



Embedded Control Handbook Update 1995 / 1996

**SERVING A COMPLEX AND COMPETITIVE
WORLD WITH FIELD-PROGRAMMABLE
EMBEDDED CONTROL
SYSTEM SOLUTIONS**

"All rights reserved. Copyright ©1995, Microchip Technology Incorporated, USA. Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights."

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A

The Microchip logo and name are registered trademarks of Microchip Technology Inc.

PICMASTER, PICSTART, PRO MATE, PICSEE, TrueGauge name and logo, *fuzzyLAB*, Smart Serial, Total Endurance, UniMouse, The Emerging World Standard, and The Embedded Control Solutions Company are trademarks of Microchip Technology Inc.

SQTP is a service mark of Microchip Technology Inc.

ACCESS.bus is a trademark of the ACCESS.bus Industry Group (ABIG). *fuzzyTECH* is a registered trademark of Inform Software Corporation. I²C is a trademark of Philips Corporation. IBM, IBM PC-AT, IBM PC/XT, and PS/2 are registered trademarks of International Business Machines Corp. Pentium is a trademark of Intel Corporation. MS-DOS, Microsoft Windows and Visual Basic are registered trademarks of Microsoft Corporation. Windows and Excel are trademarks of Microsoft Corporation. CompuServe is a registered trademark of CompuServe Incorporated. Apple and Macintosh are registered trademarks of Apple Computer, Inc. Apple Desktop Bus (ADB) is a trademark of Apple Computer Inc. National Semiconductor is a registered trademark of National Semiconductor. Microwire is a trademark of National Semiconductor. Motorola is a registered trademark of Motorola, Inc. VESA is a registered trademark of the Video Electronics Standards Association. DDC is a trademark of VESA. Panasonic is a registered trademark of Panasonic. Inform is a registered trademark of Inform Software.

All other trademarks mentioned herein are property of their respective companies.

MICROCHIP TECHNOLOGY INC. COMPANY PROFILE	1
PIC16/17 MICROCONTROLLER APPLICATION NOTES	2
APPLICATION SPECIFIC STANDARD PRODUCTS APPLICATION NOTES	3
SERIAL EEPROM APPLICATION NOTES	4
QUALITY AND RELIABILITY APPLICATION NOTES	5
DEVELOPMENT SYSTEMS AND SOFTWARE TOOLS	6
SALES AND SERVICE LOCATIONS	7

TO OUR VALUED CUSTOMERS:

Welcome to the **1995/1996 Embedded Control Handbook (ECHB) Update**. The *1995/1996 ECHB Update* is the first in the series of supplemental publications from Microchip Technology Inc. It includes all new application notes which have been written and published since the **1994/1995 Embedded Control Handbook** (released in September 1994).

With the 1995/1996 update, Microchip is introducing a library system of 'Volumes' and 'Updates' to PIC16/17, Non-Volatile Memory and other product application notes. *Volume 1* will be published in the fall of 1996, replacing the existing 1994/1995 ECHB. Thereafter, updates will be published annually, providing an uninterrupted flow of current application notes for our customers' convenience and use. These updates, with revised and new application notes, will be incorporated into future volumes as appropriate.

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip product. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced. We welcome your feedback.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department at facsimile 602.917.4150.

Table of Contents

		<u>PAGE</u>
<hr/>		
SECTION 1	MICROCHIP TECHNOLOGY INC. COMPANY PROFILE	
	Company Profile	1-1
<hr/>		
SECTION 2	PIC16/17 MICROCONTROLLER APPLICATION NOTES	
AN597	Implementing Ultrasonic Ranging.....	2-1
AN600	Air Flow Control Using Fuzzy Logic.....	2-9
AN606	Low Power Design Using PIC16/17	2-31
AN607	Power-up Trouble Shooting.....	2-47
AN611	Resistance and Capacitance Meter Using a PIC16C622	2-69
AN615	Clock Design Using Low Power/Cost Techniques.....	2-103
AN616	Digital Signal Processing with the PIC16C74	2-121
AN617	Fixed Point Routines	2-155
<hr/>		
SECTION 3	APPLICATION SPECIFIC STANDARD PRODUCTS APPLICATION NOTES	
AN599	Energy Management Control System.....	3-1
<hr/>		
SECTION 4	SERIAL EEPROM APPLICATION NOTES	
AN601	EEPROM Endurance Tutorial.....	4-1
AN602	How to get 10 Million Cycles out of your Microchip Serial EEPROM.....	4-5
AN603	Continuous Improvement	4-27
AN608	Converting to 24LCXXB and 93LCXX Serial EEPROMs.....	4-7
AN609	Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller	4-9
AN610	Using the 24LC21 Dual Mode Serial EEPROM.....	4-17
AN613	Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports	4-23
AN614	Interfacing the 8051 with 2-wire Serial EEPROMs	4-37
<hr/>		
SECTION 5	QUALITY AND RELIABILITY APPLICATION NOTES	
AN595	Improving the Susceptibility of an Application to ESD	5-1
AN598	Plastic Packaging and the Effects of Surface Mount Soldering Techniques	5-19



Table of Contents (continued)

PAGE

SECTION 6 DEVELOPMENT SYSTEMS AND SOFTWARE TOOLS

DEVELOPMENT SYSTEMS:

System Support	Development System Selection Chart	6-1
Microchip BBS	Microchip Bulletin Board Service	6-3
Design Kit	Microchip Serial EEPROM Designer's Kit	6-5
PICMASTER™	Universal In-Circuit Emulator System with MPLAB IDE	6-7
PRO MATE™	Universal Device Programmer	6-13
PICSEE™ Tools	PICSEE Product Development Tools	6-17
PICSTART™-16B1	PIC16CXX Low-Cost Microcontroller Development System	6-25
PICSTART-16C	PIC16CXX Low-Cost Microcontroller Development System	6-27
PICDEM-1	Low-Cost PIC16/17 Demonstration Board	6-29
PICDEM-2	Low-Cost PIC16CXX Demonstration Board.....	6-31

SOFTWARE TOOLS:

<i>fuzzyTECH</i> ®-MP	Fuzzy Logic Development System for PIC16/17	6-33
MPASM	Universal PIC16/17 Microcontroller Assembler Software.....	6-35
MPSIM	PIC16/17 Microcontroller Simulator	6-37
MP-C	C Compiler	6-39
Total Endurance™	Microchip Serial EEPROM Endurance Model.....	6-41
TrueGauge™ Tools	TrueGauge Intelligent Battery Management Development Tool	6-43

SECTION 7 SALES AND SERVICE LOCATIONS

Factory Representatives	7-1
Distributors	7-7
Factory Sales	7-19

Cross Reference Guides

CROSS REFERENCE GUIDE TO UPDATE 1995/1996 APPLICATION NOTES - ALPHABETICAL

	<u>PAGE</u>
Clock Design Using Low Power/Cost Techniques	AN615.....2-103
Continuous Improvement	AN603.....4-27
Converting to 24LCXXB and 93LCXX Serial EEPROMs	AN608.....4-7
Digital Signal Processing with the PIC16C74.....	AN616.....2-121
EEPROM Endurance Tutorial	AN601.....4-1
Energy Management Control System	AN599.....3-1
Fixed Point Routines	AN617.....2-155
Air Flow Control Using Fuzzy Logic	AN600.....2-9
How to get 10 Million Cycles out of your Microchip Serial EEPROM	AN602.....4-5
Implementing Ultrasonic Ranging	AN597.....2-1
Improving the Susceptibility of an Application to ESD.....	AN595.....5-1
Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Micocontroller	AN609.....4-9
Interfacing the 8051 with 2-wire Serial EEPROMs.....	AN614.....4-37
Low Power Design Using PIC16/17	AN606.....2-31
Plastic Packaging and the Effects of Surface Mount Soldering Techniques.....	AN598.....5-19
Power-up Trouble Shooting	AN607.....2-47
Resistance and Capacitance Meter Using a PIC16C622	AN611.....2-69
Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports.....	AN613.....4-23
Using the 24LC21 Dual Mode Serial EEPROM	AN610.....4-17

CROSS REFERENCE GUIDE TO UPDATE 1995/1996 APPLICATION NOTES - NUMERICAL

	<u>PAGE</u>	
AN595	Improving the Susceptibility of an Application to ESD.....5-1	
AN597	Implementing Ultrasonic Ranging	2-1
AN598	Plastic Packaging and the Effects of Surface Mount Soldering Techniques.....	5-19
AN599	Energy Management Control System	3-1
AN600	Air Flow Control Using Fuzzy Logic	2-9
AN601	EEPROM Endurance Tutorial	4-1
AN602	How to get 10 Million Cycles out of your Microchip Serial EEPROM	4-5
AN603	Continuous Improvement	4-27
AN606	Low Power Design Using PIC16/17.....	2-31
AN607	Power-up Trouble Shooting.....	2-47
AN608	Converting to 24LCXXB and 93LCXX Serial EEPROMs	4-7
AN609	Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Micocontroller	4-9
AN610	Using the 24LC21 Dual Mode Serial EEPROM	4-17
AN611	Resistance and Capacitance Meter Using a PIC16C622.....	2-69
AN613	Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports.....	4-23
AN614	Interfacing the 8051 with 2-wire Serial EEPROMs.....	4-37
AN615	Clock Design Using Low Power/Cost Techniques	2-103
AN616	Digital Signal Processing with the PIC16C74.....	2-121
AN617	Fixed Point Routines	2-155

Cross Reference Guides

CROSS REFERENCE GUIDE TO UPDATE 1995/1996 APPLICATION NOTES - BY SUBJECT

		<u>PAGE</u>
Clocks.....	AN615.....	2-103
Digital Signal Processing.....	AN616.....	2-121
EEPROM, 24 Series.....	AN608.....	4-7
	AN610.....	4-17
EEPROM, 93 Series.....	AN608.....	4-7
	AN613.....	4-23
EEPROM, Endurance.....	AN601.....	4-1
	AN602.....	4-5
	AN603.....	4-27
EEPROM, Interfacing.....	AN609.....	4-9
	AN613.....	4-23
	AN614.....	4-37
Energy Management.....	AN599.....	3-11
ESD.....	AN595.....	5-1
Fixed Point Routines.....	AN617.....	2-155
Fuzzy Logic.....	AN600.....	2-9
Improvement.....	AN603.....	4-27
Low Power Design.....	AN606.....	2-31
	AN615.....	2-103
Math.....	AN617.....	2-155
PIC16/17.....	AN607.....	2-47
PIC16C622.....	AN611.....	2-69
PIC16C74.....	AN616.....	2-121
RC Meter.....	AN611.....	2-69
Surface Mount Soldering.....	AN598.....	5-19
Ultrasonic Ranging.....	AN597.....	2-1

Cross Reference Guides

ADDITIONAL APPLICATION NOTES AVAILABLE FROM MICROCHIP

The following is a list of application notes that are available in the Microchip Technology Inc *1994/1995 Embedded Control Handbook*. Please see your local Microchip Sales Representative, Distributor or Sales Office for the latest copy (order number DS00092C).

AN510	Implementation of an Asynchronous Serial I/O
AN511	PLD Replacement
AN512	Implementing Ohmmeter/Temperature Sensor
AN513	Analog to Digital Conversion
AN514	Software Interrupt Techniques
AN515	Communicating with the I ² C Bus Using the PIC16C5X
AN517	24C01A Compatibility Issues and Its Mobility for Memory Upgrade
AN519	Implementing a Simple Serial Mouse Controller
AN520	A Comparison of Low End 8-Bit Microcontrollers
AN521	Interfacing to AC Power Lines
AN522	Power-Up Considerations
AN526	PIC16C5X/16CXX Math Utility Routines
AN527	Software Stack Management
AN528	Implementing Wake-Up on Key Stroke
AN529	Multiplexing LED Drive and a 4x4 Keypad Sampling
AN530	Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers
AN531	Intelligent Remote Positioner (Motor Control)
AN532	Servo Control of a DC-Brush Motor
AN534	Saving and Restoring Status on Interrupt (Implementing a Parameter Stack)
AN535	Logic Powered Serial EEPROMs
AN536	Basic Serial EEPROM Operation
AN537	Everything a System Engineer Needs to Know About Serial EEPROM Endurance
AN538	Using PWM to Generate Analog Output
AN539	Frequency and Resolution Options for PWM Outputs
AN540	Implementing IIR Digital Filters
AN541	Using a PIC16C5X as a Smart I ² C Peripheral
AN542	Implementation of Fast Fourier Transforms
AN543	Tone Generation
AN544	Math Utility Routines
AN545	Using the Capture Module
AN546	Using the Analog to Digital Converter
AN547	Serial Port Utilities
AN548	Implementing Table Read and Table Write
AN550	1.8 Volt Technology - Benefits
AN551	Serial EEPROM Solutions vs. Parallel Solutions
AN552	Implementing Wake Up on Keystroke
AN554	Software Implementation of I ² C Bus Master
AN555	Software Implementation of Asynchronous Serial I/O
AN556	Implementing a Table Read
AN557	Four Channel Digital Volt Meter with Display and Keyboard

Cross Reference Guides

AN558	Using the 24XX65 and 24XX32 with Stand Alone PIC16C54 Code
AN559	Optimizing Serial Bus Operations with Proper Write Cycle Times
AN560	Using the 93LC56 and 93LC66
AN562	Using the Microchip Endurance Predictive Software
AN563	Using PIC16C5X Microcontrollers as LCD Drivers
AN564	Using the PWM
AN566	Using the PortB Interrupt on Change as an External Interrupt
AN567	Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54
AN569	Hardware and Software Resolution For a Pointing Device
AN570	TrueGauge™ Calibrating the MTA11200 System
AN571	Communicating with EEPROM in MTA85XXX
AN572	Questions and Answers Concerning Serial EEPROMs
AN575	IEEE 754 Compliant Floating-Point Routines
AN576	Techniques to Disable Global Interrupts
AN577	PIC16C54A EMI Results
AN578	Use of the SSP Module in the I ² C Multi-Master Environment
AN579	Using the 8-Bit Parallel Slave Port
AN580	Using Timer1 in Asynchronous Clock Mode
AN581	Implementing Long Calls
AN582	Low Power Real-Time Clock
AN583	Implementation of the Data Encryption Standard Using PIC17C42
AN584	PICMASTER™ Support of Microsoft® Windows® DDE
AN585	A Real-Time Operating System for PIC16/17
AN586	Macros for Page and Bank Switching
AN587	Interfacing to an LCD Module
AN588	PIC16/17 Oscillator Design Guide
AN589	A PC-Based Development Programmer for the PIC16C84
AN590	A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs
AN591	Apple® Desktop Bus (ADB™)
AN592	Frequency Counter Using PIC16C5X
AN593	Serial Port Routines Without Using the RTCC
AN594	Using the CCP Modules



SECTION 1 MICROCHIP TECHNOLOGY INC. COMPANY PROFILE

Company Profile1-1



Microchip Technology Inc.

Company Profile

INTRODUCTION TO THE EMBEDDED CONTROL SOLUTIONS COMPANY™

Microchip Technology's mission is to offer industry leading semiconductor products for embedded control system applications. To do this we have focused our technology, engineering, manufacturing and marketing resources on two synergistic product lines: 8-bit PIC16/17 microcontrollers and Serial EEPROMs. These product lines provide the solutions to many of the problems facing designers of embedded control systems.

We publish the *Microchip Data Books* and *Embedded Control Handbook* to assist our customers, existing and new, in their efforts to design and produce state-of-the-art embedded control systems.

HIGHLIGHTS

Inside Microchip Technology you'll find:

- A focus on providing high-performance, cost-effective, field-programmable embedded control solutions
- An experienced executive team focused on innovation and committed to listening to our customers
- 8-bit RISC field-programmable microcontrollers and supporting logic products
- Serial and Parallel EEPROMs and EPROMs



Chandler, Arizona:

Company headquarters near Phoenix, Arizona; executive offices, R&D, and wafer fabrication occupy this 242,000 square-foot multi-building facility.

- A variety of end-user Application-Specific Standard Products
- Fully integrated manufacturing capabilities
- A global network of manufacturing and customer support facilities
- A unique corporate culture dedicated to continuous improvement
- Distributor network support worldwide including certified distribution FAEs

BUSINESS SCOPE

Microchip Technology Inc. manufactures and markets a variety of VLSI CMOS semiconductor components to support the market for cost-effective embedded control solutions. In particular, the company specializes in highly integrated, field-programmable RISC microcontrollers, application-specific standard products and related non-volatile memory products to meet growing market requirements for high performance, yet economical embedded control capability in products. Microchip's products feature the industry's most economical OTP (One-Time-Programmable) EPROM, reprogrammable EEPROM, and ROM capability, along with the compact size, integrated functionality, ease of development and technical support so essential to timely and cost-effective product development by our customers.



Tempe, Arizona:

Microchip's 170,000 square-foot wafer fabrication facility provides increased manufacturing capacity today and for the future.

Microchip Technology Inc.



MICROCHIP

- Mission Statement -

Microchip Technology Incorporated is a leading supplier of field-programmable embedded control solutions by providing RISC microcontrollers and related non-volatile memory products. In order to contribute to the ongoing success of customers, shareholders and employees, our mission is to focus resources on high-value, high-quality products and to continuously improve all aspects of our business, providing a competitive return on investment.

- Guiding Values -

Customers Are Our Focus: We establish successful customer partnerships by exceeding customer expectations for products, services and attitude. We start by listening to our customers, earning our credibility by producing quality products, delivering comprehensive services and meeting commitments. We believe each employee must effectively serve their internal customers in order for Microchip's external customers to be properly served.

Quality Comes First: We will perform correctly the first time, maintain customer satisfaction and measure our quality against requirements. We practice effective and standardized improvement methods, such as statistical process control to anticipate problems and implement root cause solutions. We believe that when quality comes first, reduced costs follow.

Continuous Improvement Is Essential: We utilize the concept of "Vital Few" to establish our priorities. We concentrate our resources on continuously improving the Vital Few while empowering each employee to make continuous improvements in their area of responsibility. We strive for constructive and honest self-criticism to identify improvement opportunities.

Employees Are Our Greatest Strength: We design jobs and provide opportunities promoting employee teamwork, productivity, creativity, pride in work, trust, integrity, fairness, involvement, development and empowerment. We base recognition, advancement and compensation on an employee's achievement of excellence in team and individual performance. We provide for employee health and welfare by offering competitive and comprehensive employee benefits.

Products And Technology Are Our Foundation: We make ongoing investments and advancements in the design and development of our manufacturing process, device, circuit, system and software technologies to provide timely, innovative, reliable and cost effective products to support current and future market opportunities.

Total Cycle Times Are Optimized: We focus resources to optimize cycle times to our internal and external customers by empowering employees to achieve efficient cycle times in their area of responsibility. We believe that cycle time reduction is achieved by streamlining processes through the systematic removal of barriers to productivity.

Safety Is Never Compromised: We place our concern for safety of our employees and community at the forefront of our decisions, policies and actions. Each employee is responsible for safety.

Profits And Growth Provide For Everything We Do: We strive to generate and maintain competitive rates of company profits and growth as they allow continued investment for the future, enhanced employee opportunity and represent the overall success of Microchip.

Communication Is Vital: We encourage appropriate, honest, constructive, and ongoing communication in company, customer and community relationships to resolve issues, exchange information and share knowledge.

Suppliers, Representatives, And Distributors Are Our Partners: We strive to maintain professional and mutually beneficial partnerships with suppliers, representatives, and distributors who are an integral link in the achievement of our mission and guiding values.

Professional Ethics Are Practiced: We manage our business and treat customers, employees, shareholders, investors, suppliers, distributors, representatives, community and government in a manner that exemplifies our honesty, ethics and integrity. We recognize our responsibility to the community and are proud to serve as an equal opportunity employer.

MARKET FOCUS

Microchip targets selected markets where our advanced designs, progressive process technology, and industry-leading product performance enable us to deliver decidedly superior performance. The company has positioned itself to maintain a dominant role as a supplier of high-performance, field-programmable microcontrollers and associated memory and logic products for embedded control applications which are found throughout the consumer, automotive, telecommunication, office automation and industrial control markets.

FULLY INTEGRATED MANUFACTURING

Microchip delivers fast turnaround and consistent quality through total control over all phases of production. Research and development, design, mask making, wafer fabrication, and the major part of assembly and quality assurance testing are conducted at facilities wholly-owned and operated by Microchip. Our integrated approach to manufacturing along with rigorous use of advanced Statistical Process Control (SPC) and a continuous improvement culture has resulted in high and consistent yields which have positioned Microchip as a quality leader in its global markets. Microchip's unique approach to SPC provides customers with excellent costs, quality, reliability and on-time delivery.

A GLOBAL NETWORK OF PLANTS AND FACILITIES

Microchip is a global competitor providing local service to the world's technology centers. The Company's design and technology advancement facility is located in Chandler, Arizona. Product and technology development is located here, along with front-end wafer fabrication and wafer probe and sort.

In 1994, Microchip purchased a second wafer fabrication facility in Tempe, Arizona – thirteen miles from its Chandler, Arizona, headquarters. The additional 170,000 square foot facility meets the increased production requirements of a growing customer base, and provides production capacity which more than doubles that of Chandler. Assembly and test facilities, predominantly located in the Philippine Islands, Kaohsiung, Taiwan, and Bangkok, Thailand, house the technology and assembly and test equipment necessary for modern plastic and ceramic packaging.

Sales and application offices are located in key cities throughout the Americas, Asia/Pacific, Japan and Europe. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical and business support.

EMBEDDED CONTROL OVERVIEW

Unlike "processor" applications such as personal computers and workstations, the computing or controlling elements of embedded control applications are buried inside the application. The user of the product is only concerned with the very top-level user interface (such as keypads, displays and high-level commands). Very rarely does an end-user know (or care to know) the embedded controller inside (unlike the conscientious PC users, who are intimately familiar not only with the processor type, but also its clock speed, DMA capabilities and so on).

It is, however, most vital for designers of embedded control products to select the most suitable controller and companion devices. Embedded control products are found in all market segments: consumer, commercial, PC peripherals, telecommunications (including fast-emerging personal telecommunication products), automotive and industrial. Most often embedded control products must meet special requirements: cost-effectiveness, low power, small footprint, and a high level of system integration.

Typically, most embedded control systems are designed around a microcontroller which integrates on-chip program memory, data memory (RAM) and various peripheral functions, such as timers and serial communication. In addition, these systems also usually require complementary Serial EEPROM memories, display drivers, keypads or small displays.

Microchip Technology has established itself as a leading supplier of field-programmable embedded control solutions. The combination of high-performance microcontrollers from the PIC17CXX, PIC16CXX and PIC16C5X families, along with industry leading non-volatile memory products, provide the basis for this leadership.

Microchip is committed to continuous innovation and improvement in design, manufacturing and technical support to provide the best possible embedded control solutions to you.

Microchip Technology Inc.

MICROCONTROLLERS

PIC16/17 microcontrollers from Microchip combine high performance, low cost, and small package size, offering the best price/performance ratio in the industry. More than 200 million of these devices have been used in cost-sensitive consumer products, computer peripherals, office automation, automotive control systems, security and telecommunication applications.

PIC16/17 MICROCONTROLLER OVERVIEW AND ROADMAP

Microchip offers three families of 8-bit microcontrollers to best fit your needs:

- PIC16C5X: Base-Line 8-bit Family
- PIC16CXX: Mid-Range 8-bit Family
- PIC17CXX: High-End 8-bit Family

All families offer One-Time-Programmable, low-voltage and low-power options, as well as various packaging options. Selected members are available in ROM and reprogrammable versions.

The widely-accepted PIC16C5X, PIC16CXX and PIC17CXX families are the industry's only 8-bit microcontrollers using a high-speed RISC architecture. Microchip pioneered the use of RISC architecture to obtain high speed and instruction efficiency.

PIC16C5X: BASE-LINE FAMILY

PIC16C5X is the well established base-line family offering the most cost-effective solution. These PIC16C5X products have a 12-bit wide instruction set and are currently offered in 18-, 20- or 28-pin packages. In the SOIC and SSOP packaging options, these are the smallest footprint controllers. Low-voltage operation down to 2.0V makes this family ideal for battery operated applications.

PIC16CXX: MID-RANGE FAMILY

PIC16CXX mid-range family offers a wide-range of options, from 18-pin to 44-pin packages as well as low to high levels of peripheral integration. This family has a 14-bit wide instruction set, interrupt handling capability and a deeper 8-level hardware stack. The PIC16CXX family provides the performance and versatility to meet the requirements of more demanding, yet cost-sensitive, mid-range 8-bit applications.

The PIC16CXX mid-range family is rapidly gaining acceptance with several of its members introduced: PIC16C620, PIC16C621, PIC16C622, PIC16C61, PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C71, PIC16C73, PIC16C74 and PIC16C84.

PIC17CXX: HIGH-END FAMILY

The PIC17CXX high-end family offers the world's fastest execution performance of any 8-bit microcontroller family in the industry. The PIC17CXX family extends the PIC16/17 microcontroller's high-performance RISC architecture with a 16-bit instruction word, enhanced instruction set and powerful vectored interrupt handling capabilities. A powerful array of precise on-chip peripheral features provide the performance for the most demanding 8-bit applications.

All three members of the PIC17CXX family have been announced and are available in production.

Current PIC16/17 microcontroller product families include advanced features such as sophisticated timers, embedded Analog-to-Digital converters, extended instruction/data memory, inter-processor communication (I²C™ bus, SPI and USARTs) and ROM, RAM, EPROM and EEPROM memories.

All three families; PIC16C5X, PIC16CXX and PIC17CXX, are supported by user-friendly development systems including; assembler, software simulator, C Compiler, fuzzy logic development software, programmers and in-circuit emulators.

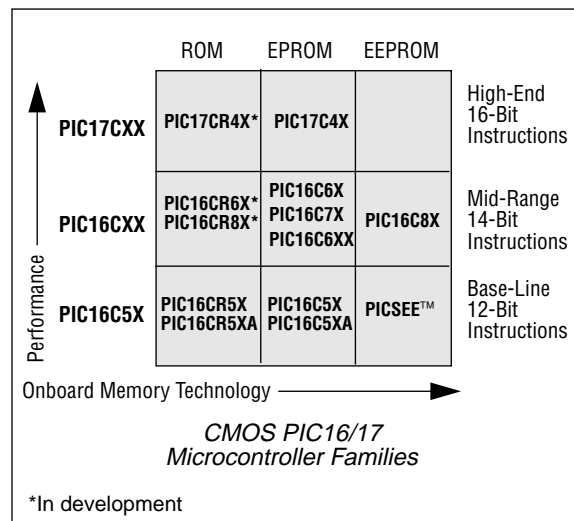


FIGURE 1: PIC16/17 MICROCONTROLLER MIGRATION PATH

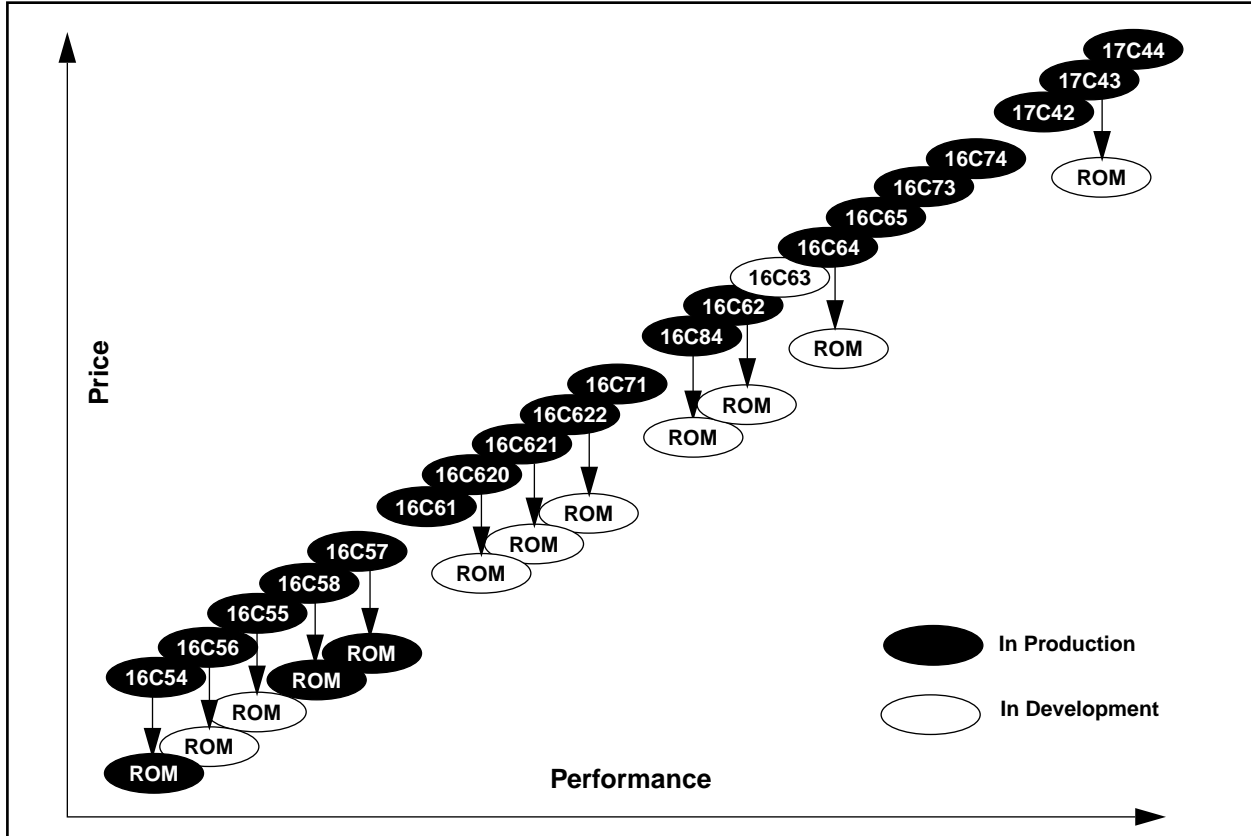


FIGURE 2: PIC16/17 SYNERGISTIC DEVELOPMENT TOOLS

Development Tool	Name	PIC16C5X	PIC16CXX	PIC17CXX
Assembler	MPASM	✓	✓	✓
Software Simulator	MPSIM	✓	✓	✓
C Compiler*	MP-C	✓	✓	✓
Entry Level Development Kit	PICSTART™	✓	✓	Planned
Universal Programmer	PRO MATE™	✓	✓	✓
Universal In-Circuit Emulator	PICMASTER™	✓	✓	✓
Fuzzy Logic Development Tool	fuzzyTECH®-MP	✓	✓	✓

* Available from Byte Craft Limited in Canada and supported by Microchip.

Microchip Technology Inc.

PIC16/17 NAMING CONVENTION

The PIC16/17 architecture offers users a wider range of cost/performance options of any 8-bit microcontroller family. In order to identify the families, the following naming conventions have been applied to the PIC16/17 microcontrollers.

TABLE 1: PIC16/17 NAMING CONVENTION

Family		Architectural Features	Name	Technology	Products
PIC16C5X	Base-Line 8-bit Microcontroller Family	<ul style="list-style-type: none"> 12-bit wide instruction set DC - 20 MHz clock speed 200 ns instruction cycle 	PIC16C5X PIC16C5XA (Note 1)	OTP program memory, digital only	PIC16C54 PIC16C54A PIC16C55 PIC16C56 PIC16C57 PIC16C58A
			PIC16CR5X PIC16CR5XA (Note 1)	ROM program memory, digital only	PIC16CR54 PIC16C54A PIC16CR57A PIC16CR58A
PIC16CXX	Mid-Range 8-bit Microcontroller Family	<ul style="list-style-type: none"> 14-bit wide instruction set Internal/external interrupts DC - 20 MHz clock speed (Note 3) 200 ns instruction cycle (@ 20 MHz) 	PIC16C6X	OTP program memory, digital	PIC16C61 PIC16C62 PIC16C63 PIC16C64 PIC16C65
			PIC16CR6X	ROM program memory, digital only	Planned
			PIC16C62X	OTP program memory with comparators	PIC16C620 PIC16C621 PIC16C622
			PIC16C7X	OTP program memory, with analog functions (i.e., A/D)	PIC16C71 PIC16C73 PIC16C74
			PIC16C8X	EEPROM program and data memory	PIC16C84
			PIC16CR8X	ROM program and EEPROM data memory	Planned
PIC17CXX	High-End 8-bit Microcontroller Family	<ul style="list-style-type: none"> 16-bit wide instruction set Internal/external interrupts DC - 25 MHz clock speed 160 ns instruction cycle 	PIC17C4X	OTP program memory, digital only	PIC17C42 PIC17C43 PIC17C44
			PIC17CR4X	ROM program memory, digital only	Planned

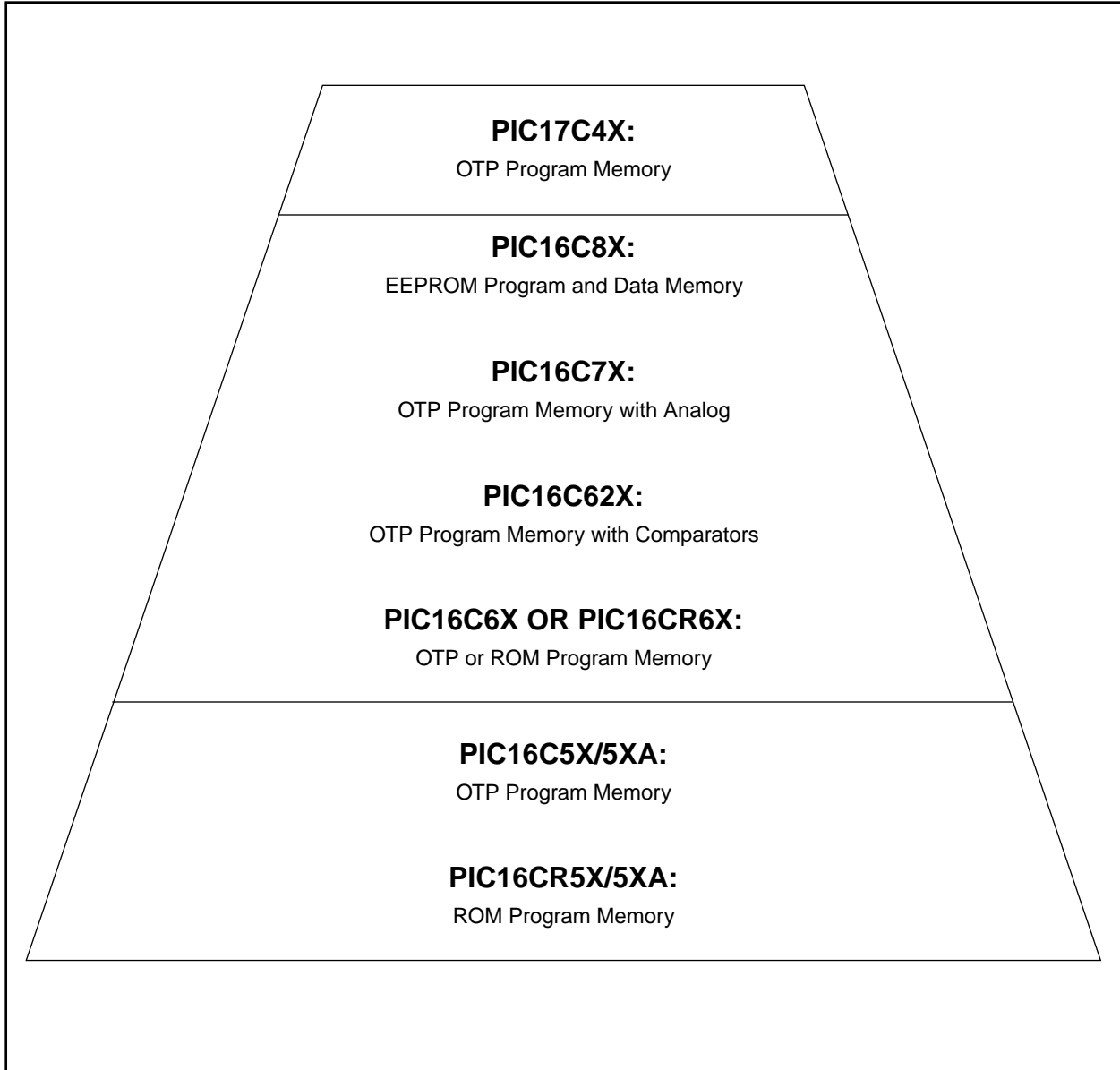
Note 1: "A" designates a more advanced process technology, generally offering customers the benefits of lower power, higher speed, etc. (example: PIC16C54, PIC16C54A). Sometimes it designates additional functions such as the addition of Brown-out Detect.

Note 2: The numbering system within each family is not necessarily significant.

Note 3: The maximum clock speed for some devices is less than 20 MHz.

Please check with your local Microchip distributor, sales representative or sales office for the latest product information.

FIGURE 3: PIC16/17 8-BIT MICROCONTROLLER FAMILY



Microchip Technology Inc.

THE ADVANTAGE OF FIELD PROGRAMMABILITY

The PIC16/17 microcontroller family provides a unique combination of a high-performance RISC processor with cost-effective One-Time-Programmable (OTP) technology. Cost-effective OTP provides many benefits to the user at prices which can be comparable to competing ROM solutions. The benefits include: 1) quick time-to-market, 2) ease of code changes, 3) ability to provide adaptable solutions to end-customer requirements, 4) ability to meet upside potential via inventory positions at Microchip or worldwide distribution, 5) reduced scrappage in manufacturing, 6) reduced inventory in manufacturing, and 7) reduced work-in-process liability.

For most manufacturers, getting the product to market quickly has become the number one goal as global markets have become more competitive. Time-to-market puts pressure on all functions within the manufacturing process: development, purchasing, production, marketing and sales. Field-programmable OTP technology streamlines the process for all stages in the product life cycle.

In the early product development stages, a programmable microcontroller allows much of the functionality to be implemented in software which can be modified more easily than hardware-only solutions.

In the manufacturing stage, the compression of the product life cycle curve puts pressure on the management of inventory and manufacturing cycle times. Minimizing inventory reduces the ability to meet upside demand. Using a traditional ROM-based microcontroller limits the ability to respond to the market with product enhancements or semi-customized products for specific customers. Using the standard OTP-based PIC16/17 microcontroller solves all these issues. Inventory can be managed effectively by using the same device in several systems. Costs can be reduced due to volume purchasing. Upsides can be met from either safety stock, directly from Microchip, or local distributors who regularly inventory all PIC16/17 microcontroller devices. A sudden decline in demand means no work-in-process ROM-based inventory and any excess safety stock can be consumed by the other products using the same standard device.

OTP is the 'Flexible Manufacturing' technology of the microcontroller world. As competition intensifies, the demand for customer-specific products increases. Having the ability to change (for example, the appearance of LCD displays or add extra features in a timely manner) can be a key competitive advantage. Programming the OTP device on the manufacturing floor allows easy customizing and internal tracking of the devices for each specific customer. Customization can significantly increase the overall product life cycle to provide better return on investment and help minimize the threat of competition.

DEVELOPMENT SYSTEMS

Microchip is committed to providing useful and innovative solutions to your embedded system designs. Among support products offered are the PICMASTER™ Real-Time Universal In-circuit Emulator running under the Windows® environment. PICMASTER is designed to provide product development engineers with an optimized design tool for developing target applications. This universal in-circuit emulator provides a complete microcontroller design tool set for all microcontrollers in the PIC16C5X, PIC16CXX and PIC17CXX families. PRO MATE™, the full-featured device programmer, enables you to quickly and easily program user software into PIC16C5X, PIC16CXX and PIC17CXX CMOS microcontrollers. The PRO MATE operates as a stand-alone unit or in conjunction with a PC compatible host system. The PICSTART™ development kit, a low-cost development system for the PIC16C5X/16CXX families of microcontrollers, includes an assembler for code development, a simulator for debug, and a development programmer board. PICSEEKIT and PICSEESTART provide product development engineers with a cost-effective and timely design tool solution for the MTA8XXXX family of ASSP products.

The Serial EEPROM Designer's Kit includes everything necessary to read, write, erase, or program special features of any Microchip Serial EEPROM product including *Smart Serials*™ and secure serials. The *Total Endurance*™ Disk is included to aid in trade-off analysis and reliability calculations. The total kit can significantly reduce time-to-market and result in an optimized system.

The *TrueGauge*™ development tool supports system development with the MTA11200 TrueGauge Intelligent Battery Management IC.

SOFTWARE SUPPORT

Microchip's PIC16/17 microcontroller families are supported by an assembler, compiler, software simulator and fuzzy logic development software. MPASM is a universal macro assembler supporting Microchip's entire product line of microcontrollers. MPSIM, a discrete event software simulator, is designed to imitate operation of PIC16C5X, PIC16CXX and PIC17CXX microcontrollers. It allows the user to debug software that will use any of these microcontrollers.

A full-featured C-Compiler and Fuzzy Logic tools are also available for all three microcontroller families.

Microchip endeavors at all times to provide the best service and responsiveness possible to its customers. The Microchip Systems Bulletin Board Service (BBS) is one service to facilitate this service. It's a multi-faceted tool that can provide you with information on a number of different topics.

The Microchip Internet Home Page can provide you with technical information, application notes and promotional news on Microchip products and technology. The Microchip Web address is <http://www.mchip.com/biz/mchip>.

Special Interest Groups available through the BBS can provide you with the opportunity to discuss issues and topics of interest with others that share your interest or questions. The BBS is regularly used to distribute technical information, application notes, source code, errata sheets, bug reports, interim patches for Microchip systems products, and user contributed files for distribution. Please see Microchip BBS connection information (Section 6, Page 6-3).

APPLICATION-SPECIFIC STANDARD PRODUCTS (ASSPs)

Microchip's Application-Specific Standard Products (ASSP) provide value-added embedded control solutions by combining PIC16/17 microcontroller architecture, non-volatile memory, and innovative software technology for vertical applications. These products incorporate technology that offers a complete solution that is both unique to the customer and standard in manufacture to Microchip. In addition, Microchip ASSPs reduce or remove the barriers for customers to use Microchip solutions, in their products, through the use of software, embedded in secure OTP- or ROM-based microcontrollers. These microcontrollers are packaged to provide the highest integration, to the customer, at the best overall system cost.

The MTA11200 family is the most accurate and most integrated battery management and charging solution available today. The TrueGauge family incorporates Microchip/SPAN patented technology which digitally integrates battery charge and discharge current to provide an accurate (>97% typical) state of charge indication. The family operates with NiCd and

NiMH and lead acid battery packs from 3VDC to 25VDC. These products are ideal for portable PC, cellular phone, and portable consumer product applications.

The MTA14000 programmable Intelligent Battery Management IC allows engineers to design intelligent controllers for smart batteries, battery chargers, battery status monitoring, uninterruptible power supplies, HVAC, and other data acquisition and processing required for managing energy. The MTA14000's programmable 4K words of program memory and 192 bytes of RAM allows it to support any battery technology including Li Ion, NiMH, NiCd, Pb acid, Zinc Air. In addition, the product's I²C™ port enables any system OEM, battery pack VAR, and battery manufacturer to design, build, and market SBD-compliant products supporting the System Management Bus standard.

The MTE1122 Energy Management Controller combines Microchip's proprietary PIC16/17 8-bit RISC microcontroller technology with a unique, patent pending power management firmware algorithm in a single package. This device, by monitoring and controlling the supply requirements into an AC induction motor, effectively reduces the power consumed by the motor. The MTE1122 is available in both plastic DIP and space-saving SOIC packages, and operates over commercial and industrial ranges.

Ease-of-use, low voltage, and low cost make the MTA41XXX mouse and trackball MCU firmware solutions ideal for implementing new designs for both PCs and Apple® computers. The products in the MTA41XXX family are 18-lead, low-power, CMOS microcontroller ICs combined with application-specific software. By adding a few external components, the user can easily realize a complete mouse or trackball system.

The MTA8XXXX PICSEE™ family of cost-effective system solutions integrates PIC16/17 microcontrollers with EEPROM technology. These PICSEE devices are ideally suited for automotive security, keyless entry, remote control, telecommunication applications and data acquisition. The combined product assembly techniques provide the user the highest performance solution in a compact and cost-effective package.

Future ASSP products will include advanced features such as mixed analog and digital capability as well as an ever broadening family of turnkey software solutions for the embedded control market.

Microchip Technology Inc.

SERIAL EEPROM OVERVIEW

Microchip offers one of the broadest selections of CMOS Serial EEPROMs on the market for embedded control systems. Serial EEPROMs are available in a variety of densities, operating voltages, bus interface protocols, operating temperature ranges and space saving packages.

Densities:

Currently range from 1K to 64K with higher density devices in development.

Bus Interface Protocols:

All major protocols are covered: 2-wire, 3-wire and 4-wire.

Operating Voltages:

In addition to standard 5V devices there are two low voltage families. The "LC" devices operate down to 2.5V, while the breakthrough "AA" family operates, in both read and write mode, down to 1.8V, making these devices highly suitable for alkaline and NiCd battery powered applications.

Temperature Ranges:

Like all Microchip devices, Serial EEPROMs are offered in Commercial (0°C to 70°C), Industrial (-40°C to 85°C) and Automotive (-40°C to 125°C) operating temperature ranges.

Packages:

The focus is on small packages. Small footprint packages include: 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths, and 14-lead SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Technology Leadership:

Microchip's Serial EEPROMs are backed by a 10 million Erase/Write cycle guarantee — an endurance breakthrough unmatched by its competitors. Microchip's erase/write cycle endurance is among the best in the world, and only Microchip offers such unique and powerful development tools as the Total Endurance disk. This mathematical software model is an innovative tool used by system designers to optimize Serial EEPROM performance and reliability within the application.

The Company has also developed the world's first 64K Smart Serial EEPROM which provides four times the speed, four times the memory, and four times the features of any competitive 2-wire Serial EEPROM. Device densities range from 256 bits up to 64K bits. Another first is the 24LC21, the only single chip DDC1/DDC2™-compatible solution for plug-and-play video monitors.

Microchip is a high-volume supplier of Serial EEPROMs to all the major markets worldwide including consumer, automotive, industrial, computer, and communications. To date, more than 300 million units have been produced. Microchip continues to develop new Serial EEPROM solutions for embedded control applications.

PARALLEL EEPROM OVERVIEW

CMOS Parallel EEPROM devices from Microchip are available in 4K, 16K and 64K densities. The manufacturing process used for these EEPROMs ensures 10,000 to 100,000 write and erase cycles typical. Data retention is more than 10 years. Fast write times are less than 200 μ s. These EEPROMs work reliably under demanding conditions and operate efficiently at temperatures from -40°C to $+85^{\circ}\text{C}$. Microchip's expertise in advanced SOIC, TSOP and VSOP surface mount packaging supports our customers' needs in space-sensitive applications.

Typical applications include computer peripherals, engine control, telecommunications and pattern recognition.

OTP EPROM OVERVIEW

Microchip's CMOS EPROM devices are produced in densities from 64K to 512K. High-speed EPROMs have access times as low as 55 ns. Typical applications include computer peripherals, instrumentation, and automotive devices. Microchip's expertise in surface mount Packaging on SOIC, TSOP and VSOP packages led to the development of the Surface Mount one-time-programmable (OTP) EPROM market where Microchip is a leading supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.

Microchip Technology Inc.

EASE OF PRODUCTION UTILIZING QUICK TURN PROGRAMMING (QTP) AND SERIALIZED QUICK TURN PROGRAMMING (SQTPSM)

Recognizing the needs of high-volume manufacturing operations, Microchip has developed two programming methodologies which make the OTP products as easy to use in manufacturing as they are efficient in the system development stage.

Quick Turn Programming allows factory programming of OTP products prior to delivery to the system manufacturing operation. PIC16/17, EPROM and Serial EEPROM products can be automatically programmed, with the users program, during the final stages of the test operation at Microchip's assembly and test operations in the Philippine Islands, Taiwan and Thailand. This low-cost programming step allows the elimination of programming during system manufacturing and essentially allows the user to treat the PIC16/17 and memory products as custom ROM products. With one- to four-week lead times on QTP products, the user no longer needs to plan for the extended ROM masking lead times and masking charges associated with custom ROM products. This capability, combined with the off-the-shelf availability of standard OTP product, ensures the user of product availability and the ability to reduce his time-to-market once product development has been completed.

Unique in the 8-bit microcontroller market is Microchip's ability to enhance the QTP capability with Serialized Quick Turn Programming (SQTP). SQTP allows for the programming of devices with unique, random or serialized identification codes. As each PIC16/17 device is programmed with the customers program code, a portion of the program memory space can be programmed with a unique id, accessible from normal program memory, which will allow the user to provide each device with a unique identification. This capability is ideal for embedded systems applications where the transmission of key codes or identification of the device as a node within a network is essential. Taking advantage of this capability allows the system designer to eliminate the requirement for expensive off-chip code implementation using DIP switches or non-volatile memory components. The SQTP offering, pioneered by Microchip, provides the embedded systems designer with a low cost means of putting a unique and custom device into every system or node.

FUTURE PRODUCTS AND TECHNOLOGY

New process technology is constantly being developed for microcontroller, ASSP, EEPROM, and high-speed EPROM products. Advanced process technology modules and products are being developed that will be integrated into present product lines to continue to achieve a range of compatible processes. Current production technology utilizes lithography dimensions down to 0.9 microns. Products using 0.7 microns technology are in development.

Microchip's research and development activities include exploring new process technologies and products that have industry leadership potential. Particular emphasis is placed on products that can be put to work in high-performance broad-based markets.

Equipment is continually updated to bring the most sophisticated process, CAD and testing tools online. Cycle times for new technology development are continuously reduced by using in-house mask generation, a high-speed pilot line within the manufacturing facility and continuously improving methodologies.

More advanced technologies are under development, as well as advanced CMOS RISC-based microcontroller, ASSP, and CMOS EEPROM and EPROM products. Objective specifications for new products are developed by listening to our customers and by close co-operation with our many customer-partners worldwide.

SECTION 2

PIC16/17 MICROCONTROLLER APPLICATION NOTES

AN597	Implementing Ultrasonic Ranging	2-1
AN600	Air Flow Control Using Fuzzy Logic	2-9
AN606	Low Power Design Using PIC16/17	2-31
AN607	Power-up Trouble Shooting	2-47
AN611	Resistance and Capacitance Meter Using a PIC16C622	2-69
AN615	Clock Design Using Low Power/Cost Techniques	2-103
AN616	Digital Signal Processing with the PIC16C74	2-121
AN617	Fixed Point Routines	2-155
	Appendix A: Algorithms	2-161
	Appendix B: Flowcharts	2-162
	Appendix C: Multiply Routines for the PIC16C5X/PIC16CXX	2-164
	Appendix D: PIC16C5X/PIC16CXX Divide Routines	2-254
	Appendix E: PIC17CXX Multiply Routines	2-348
	Appendix F: PIC17CXX Divide Routines	2-386

Implementing Ultrasonic Ranging

*Author: Robert Schreiber
Logic Products Division*

INTRODUCTION

Object ranging is essential in many types of systems. One of the most popular ranging techniques is ultrasonic ranging. Ultrasonic ranging is used in a wide variety of applications including:

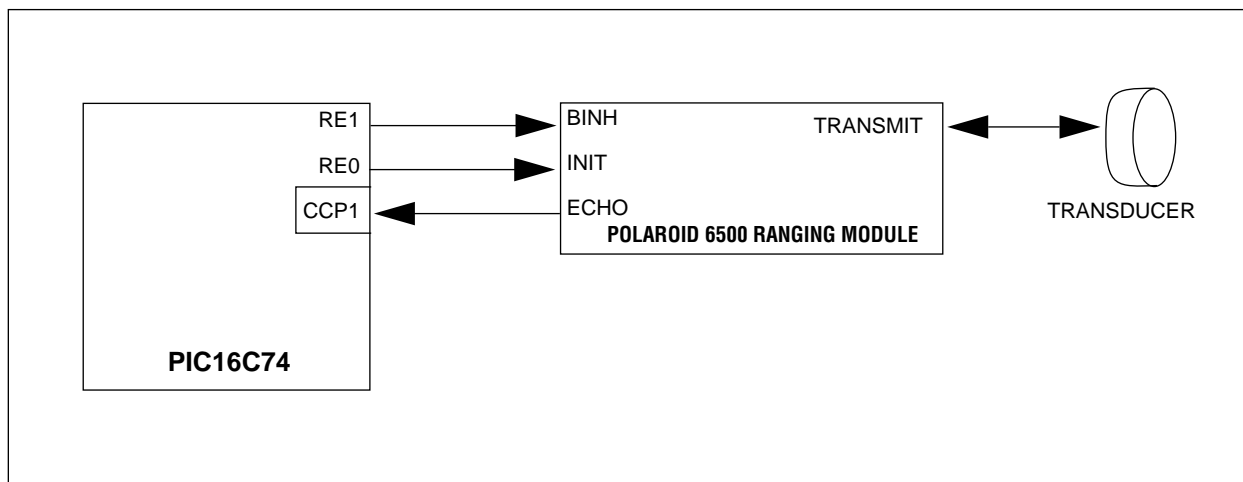
- Auto focus cameras
- Motion detection
- Robotics guidance
- Proximity sensing
- Object ranging

This application note describes a method of interfacing PIC16CXX microcontrollers to the Polaroid 6500 Ranging Module. This implementation uses a minimum of microcontroller resources, a CCP module and two I/O pins. The two major components of the system are:

- Microcontroller
- Polaroid 6500 Ranging Module

The microcontroller performs the intelligence and arithmetic functions for ultrasonic ranging, while the Polaroid 6500 Ranging Module performs the ultrasonic signal transmissions and echo detection.

FIGURE 1: RANGING MODULE INTERFACE



THEORY OF OPERATION

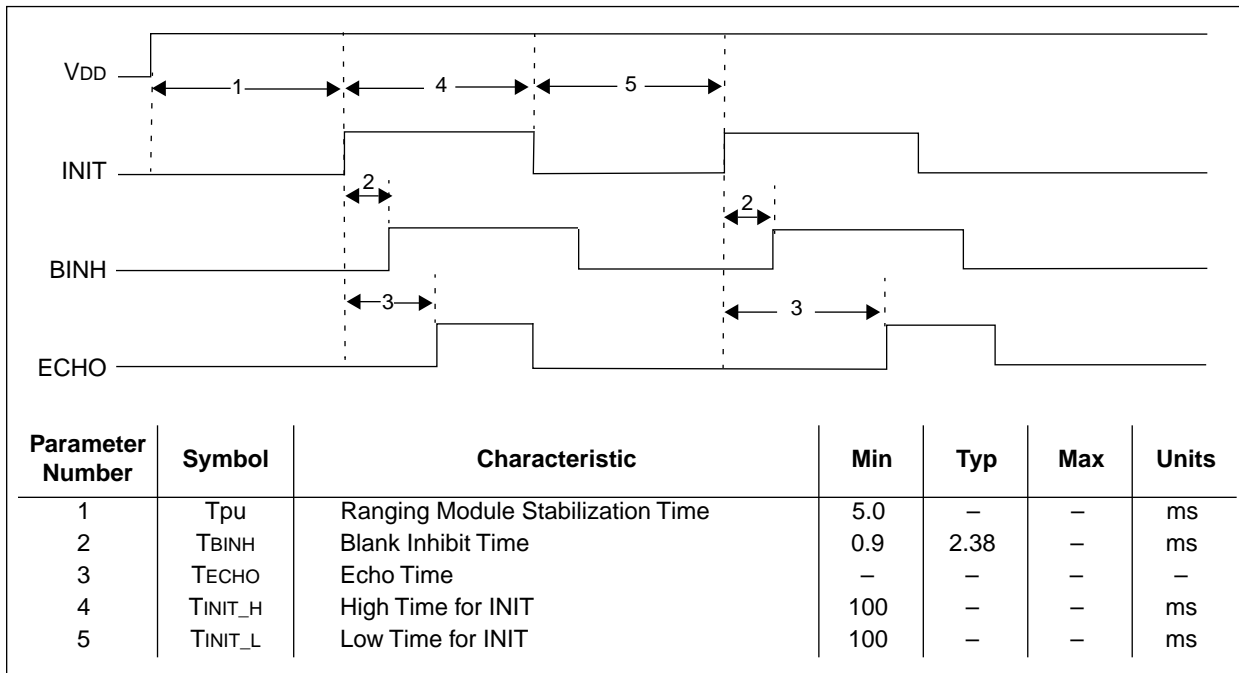
Ultrasonic ranging entails transmitting a sound wave and measuring the time that it takes for the sound wave to reflect off of an object and back to the origin. The reflection time is proportional to the distance that the object is from the source. In this implementation, the sound wave is transmitted and received from the same transducer. Therefore, a blanking interval is required between signal transmission and reception to eliminate false echoes (i.e., a transmitted signal being detected as its own echo).

CIRCUIT CONFIGURATION

In this implementation, a PIC16C74 is connected to the ranging module as shown in Figure 1. The RE0 and RE1 I/O pins are configured as digital outputs and are tied to INIT and BINH, respectively. The CCP1 pin is configured as a digital input and is tied to ECHO through a pull-up resistor. The pull-up resistor is needed since the ECHO signal is an open-collector output. The CCP1 pin is configured for capture mode (CCP1CON). Figure 2 shows the timing relationship for V_{DD} and the three signal lines (INIT, BINH, and ECHO).

Note: The ranging module requires 5.0 milliseconds to stabilize during power-up.

FIGURE 2: TIMING DIAGRAM OF RANGING MODULE CONTROL LINES



The PIC16C74 is configured to use one of its internal timers, Timer1, in capture mode to measure the time between signal transmission and echo detection. The resolution of the timer is determined by the microcontroller clock frequency. For this application, a 4 MHz external oscillator was used, giving a resolution of 1 ms per bit. The PIC16C74 initiates a ranging cycle by first clearing Timer1. Timer1 is then enabled and INIT is immediately asserted on the ranging module. When INIT is asserted, the ranging module transmits a series of 16 pulses on the transducer at 49.4 kHz. The transmitted pulses reflect off the object and are received back at the transducer.

The transducer is used for both transmitting and receiving sound waves. A blanking interval is needed to ensure that the transmitted signal has decayed on the transducer, in order not to receive false echoes. In normal operation, the ranging module has a blanking interval of 2.38 milliseconds, which corresponds to a minimum detection distance of approximately 17 inches. However, the BINH (blank inhibit) signal can be manipulated to reduce the blanking time on the transducer to allow for object ranging as close as 6 inches.

In this implementation, the PIC16C74 asserts the BINH signal approximately 0.9 milliseconds after signal transmission. This enables the transducer to receive reflections off objects at a distance of 6 inches. The ranging module asserts the ECHO signal when a valid reflection has been detected. The PIC16C74 uses the ECHO signal to trigger a capture of the Timer1 value. The capture register contains the 16-bit value

representing the elapsed time between signal transmission and echo detection. The PIC16C74 then calculates object distance based on the Timer1 value, microcontroller clock speed, and the velocity of sound in the atmosphere. The basic equation for calculating distance is given below:

$$\text{Distance (inches)} = \text{TECHO time} / 147.9 \text{ microseconds}$$

Note: The minimum high and low time for INIT is 100 milliseconds, as seen in Figure 2.

DESIGN CONSIDERATIONS

There are several design considerations which must be taken into account and are listed below.

The absolute measuring distance supported by the ranging module is 6 inches to 35 feet with an accuracy of +/- 1%.

The distance output from the ranging module can be averaged over time to filter distance calculations.

In some applications, the gain of the receiver amplifier may be too low or too high and may need to be adjusted. For example, if the transducer is mounted in a cylinder, the gain may need to be lowered to reduce false echoes within the cylinder. In this case, R1 (refer to the Polaroid Ultrasonic Ranging System manual) may be replaced with a 20 kΩ potentiometer to tweak the gain of the receiver amplifier to reduce false echoes.

In order for the Polaroid 6500 ranging module to operate properly, the power supply must be capable of handling high current transients (2.5 A) during the

transmit pulse. The instantaneous drain on the power supply can be mitigated by installing a storage capacitor across the power lines at the ranging module. A value of 500 microfarads is recommended.

A 200 millisecond interval is recommended between ranging cycles (Figure 2) to allow the transducer to clear.

The ECHO line requires a pull-up resistor (4.7 k Ω was used in this application).

There must be a common ground between the PIC16C74 circuitry and the ranging module.

Some applications may not need the resources of the higher end PIC16CXX devices. It is still possible to do this application using a device that does not contain a CCP module (for ECHO timing). The capture function can be implemented in firmware. The effect of a firmware implementation is that the resolution of the ECHO time would be 3 Tcy cycles versus 1 Tcy cycle for the CCP module. Also, the firmware implementation would not allow other tasks to be performed while the capture function was occurring.

Refer to Appendix A for general ranging module specifications.

APPENDIX A: POLAROID MODULE SPECIFICATIONS

Note: This appendix contains general specifications from the Polaroid Ultrasonic Ranging System Manual. Please refer to the current Polaroid Ultrasonic Ranging System Manual for current information regarding ranging module design considerations.

DESIGN CONSIDERATIONS IN ULTRASONICS

Range: (with user custom designed processing electronics)

Farther

- a) Use an acoustic horn to “focus” the sound (narrowing the beamwidth).
- b) Use two transducers – 1 receiver and 1 transmitter – facing each other.
- c) Lower the transmitting frequency (which will decrease the attenuation in air).

Closer

- a) Use a shorter transmit signal (such as four cycles).
- a) Use two transducers – one to transmit, one to receive (eliminates waiting for damping time).

Resolution

- a) Above all, know the target and range well, and design a system with them in mind.
- b) Use a higher transmit frequency.
- c) Look at phase differences of a given cycle of the transmitted signal and received echo (as opposed to using an integration technique).
- d) Increase the clock frequency of the timer.

Accuracy: (again, you must have a well defined target)

Temperature Compensate

- a) Use a second small target, as a reference, at a known distance in the ranging path (such as a 1/4” rod several feet away), process both echoes, then normalize the second distance with respect to the first, since $t1/d1 = t2/d2$.
- b) Incorporate a temperature sensing integrated circuit to drive a VCO to do the distance interval clocking.
- c) To increase sensitivity of detection circuit change the value of C4 from 3300 pF to 1000 pF on the 6500 Series Ranging Module.

Beam Width:

Increase

- a) Use an acoustic lens (to disperse the signal).
- b) Decrease the transmitting frequency.
- c) Use several transducers to span an area.

Decrease

- a) Use an acoustic horn (to focus the sound).
- b) Increase the transmitting frequency.

TABLE 1: RECOMMENDED OPERATING CONDITIONS

		Min.	Max.	Unit
Supply Voltage, Vcc		4.5	6.8	V
High-level input voltage, VIH	BINH, INIT	2.1		V
Low-level input voltage, VIL	BINH, INIT		0.6	V
ECHO and OSC output voltage			6.8	V
Delay time, power up to INIT high		5		ms
Recycle period		80		ms
Operating free-air temperature, TA		0	40	°C

TABLE 2: ELECTRICAL CHARACTERISTICS OVER RECOMMENDED RANGES OF SUPPLY VOLTAGE AND OPERATING FREE-AIR TEMPERATURE (UNLESS OTHERWISE NOTED)

Parameter		Test Conditions	Min.	Typ.	Max.	Unit
Input current	BINH, INIT	V1 = 2.1V			1	mA
High-level output current, I _{OH}	ECHO, OSC	V _{OH} = 5.5V			100	μA
Low-level output voltage, V _{OL}	ECHO, OSC	I _{OL} = 1.6 mA			0.4	V
Transducer bias voltage		T _A = 25°C		200		V
Transducer output voltage (peak-to-peak)		T _A = 25°C		400		V
Number of cycles for XDCR output to reach 400V		C = 500 pF			7	
Internal blanking interval				2.38*		ms
Frequency during 16-pulse transmit period	OSC output			49.4*		kHz
	XMIT output			49.4*		
Frequency after 16 pulse transmit period	OSC output			93.3*		kHz
	XMIT output			0		
Supply current, I _{CC}	During transmit period				2000	mA
	After transmit period				100	

* These typical values apply for a 420 kHz ceramic resonator.

AN597

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX B: FIRMWARE LISTING

```
MPASM 01.02 Released      XDCR.ASM  11-14-1994  9:29:15          PAGE  1
LOC OBJECT CODE          LINE SOURCE TEXT
VALUE

0001 ; XDCR.ASM
0002 ;
0003 ; This routine continually executes ranging cycles in the
0004 ; following order:
0005 ;
0006 ;     1) Timers and Flags are cleared
0007 ;     2) Ranging Cycle Executes
0008 ;     3) Distance is Calculated (to 0.5 inch)
0009 ;     4) HW is re-initialized for next cycle
0010 ;
0011 ; The processor uses a 4MHz oscillator, so all timing
0012 ; calculations are referenced to that. The calculated
0013 ; distance is a 16-bit result in the ACCbHI:ACCbLO registers.
0014 ;
0015
0016          LIST P=16C74, F=INHX8M
0017 ;
0029
0030 ;*****
0031 ; Bank 0 Registers
0032 ;*****
0033 ;
0034 ; TMR1 is off, Prescaler is 1 for a capture timeout of 65 msec
0000 0190 0035 clrf  T1CON
0036 ; Set to capture on every rising edge
0001 3005 0037 movlw  0x05
0002 0097 0038 movwf  CCP1CON
0039 ; Clear the Ports
0003 0185 0040 clrf  PORT_A
0004 0186 0041 clrf  PORT_B
0005 0187 0042 clrf  PORT_C
0006 0188 0043 clrf  PORT_D
0007 0189 0044 clrf  PORT_E
0045 ;
0046 ;*****
0047 ; Bank 1 Registers
0048 ;*****
0049 ;
0008 1683 0050 bsf   STATUS,RP0      ; Set RP0
0051 ; Port A is Digital, Port E is Digital
0009 3007 0052 movlw  0x07
000A 009F 0053 movwf  ADCON1
0054 ; Configure CCP1 (RC2) as an input, and all other ports
0055 ; as Outputs, (RE0 = INIT, RE1 = BINH)
000B 0185 0056 clrf  TRIS_A
000C 0186 0057 clrf  TRIS_B
000D 3004 0058 movlw  0x04
000E 0087 0059 movwf  TRIS_C
000F 0188 0060 clrf  TRIS_D
0010 0189 0061 clrf  TRIS_E
0011 1283 0062 bcf   STATUS,RP0      ; Clear RP0
0012
0063 Xdcr
0064 ;
0065 ; Initialize Timers and Flags
0066 ;
0012 1010 0067 bcf   T1CON,0          ; Disable TMR1
0013 018C 0068 clrf  PIR1          ; Clear Timer1 Overflow Flag & Timer1 Capture Flag
```



```

0014 018E 0069  clrf  TMR1L          ; Clear TMR1L
0015 018F 0070  clrf  TMR1H          ; Clear TMR1H
0016 0195 0071  clrf  CCP1L         ; Clear CCP1L
0017 0196 0072  clrf  CCP1H         ; Clear CCP1H
0018 1409 0073  bsf   PORT_E,0      ; Set INIT High on Ranging Module
0019 1410 0074  bsf   T1CON,0       ; Enable TMR1
001A 21F3 0075  call  DEL_9         ; Delay 0.9 msec for transducer to stabilize
001B 1489 0076  bsf   PORT_E,1      ; Enable Transducer to Receive (BINH)
001C      0077  chk_t1
001C 190C 0078  btfsc PIR1,2       ; Check for Capture
001D 2822 0079  goto  chk_done      ; Jump if Capture
001E 1C0C 0080  btfss PIR1,0       ; Check for TMR1 Overflow
001F 281C 0081  goto  chk_t1        ; Loop if nothing happened
0020 1010 0082  bcf   T1CON,0       ; Turn off TMR1
0021 2833 0083  goto  ovr_flo       ; Capture event did not occur
0022      0084  chk_done
                0085 ;
                0086 ; Calculate distance to 0.5 inch resolution
                0087 ;
0022 1010 0088  bcf   T1CON,0       ; Turn off TMR1
0023 0815 0089  movf  CCP1L,W       ; Move LSB into W
0024 00A2 0090  movwf ACCbLO       ; Move LSB into ACCbLO
0025 0816 0091  movf  CCP1H,W       ; Move MSB into W
0026 00A3 0092  movwf ACCbHI       ; Move MSB into ACCbHI
0027 304A 0093  movlw 0x4A         ; Move 75usec/0.50in into W
0028 00A0 0094  movwf ACCaLO       ; Move LSB into ACCaLO
0029 01A1 0095  clrf  ACCaHI       ; Clear MSB (ACCaHI)
002A 208F 0096  call  D_divF         ; Call 16-bit/8-bit routine
                0097 ;
                0098 ; Application Note 544
002B 3025 0099  movlw 0x25         ; Check remainder to see if
002C 0224 0100  subwf ACCcLO,W       ; we should round up...
002D 1803 0101  btfsc STATUS,CARRY ; If Remainder < (0.5 * Divisor), skip
002E 0AA2 0102  incf  ACCbLO,F     ; Round up
002F 1903 0103  btfsc STATUS,Z     ; Check low byte for wrap around
0030 0AA3 0104  incf  ACCbHI,F     ; If LSB wrapped, increment high byte
0031 1D03 0105  btfss STATUS,Z     ; Check high byte for wrap around
0032 2835 0106  goto  done         ; High byte didn't wrap
0033      0107  ovr_flo
0033 01A2 0108  clrf  ACCbLO       ;
0034 01A3 0109  clrf  ACCbHI       ;
0035      0110  done
0035 21FD 0111  call  DEL_100        ; Wait 100 msec before clearing HW.
0036 1009 0112  bcf   PORT_E,0       ; Disable INIT
0037 1089 0113  bcf   PORT_E,1       ; Disable BINH
0038 21FD 0114  call  DEL_100        ; Wait 100 msec before enabling HW.
0039 2812 0115  goto  Xdcr
                0116
                0120
                0149
                0150  end
                0151

```

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```

0000 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX-----
0040 : -----

```

All other memory blocks unused.

```

Errors      : 0
Warnings    : 0
Messages    : 0

```

AN597

NOTES:

Air Flow Control Using Fuzzy Logic

*Author: Robert Schreiber
Logic Products Division*

INTRODUCTION

Fuzzy logic control can be used to implement a wide variety of intelligent functions including everything from consumer electronic goods and household appliances to auto electronics, process control, and automation.

Typically, fuzzy logic control applications fall into two categories. First, it can be used to enhance existing products with intelligent functions. Second, it can utilize sensors that continuously respond to changing input

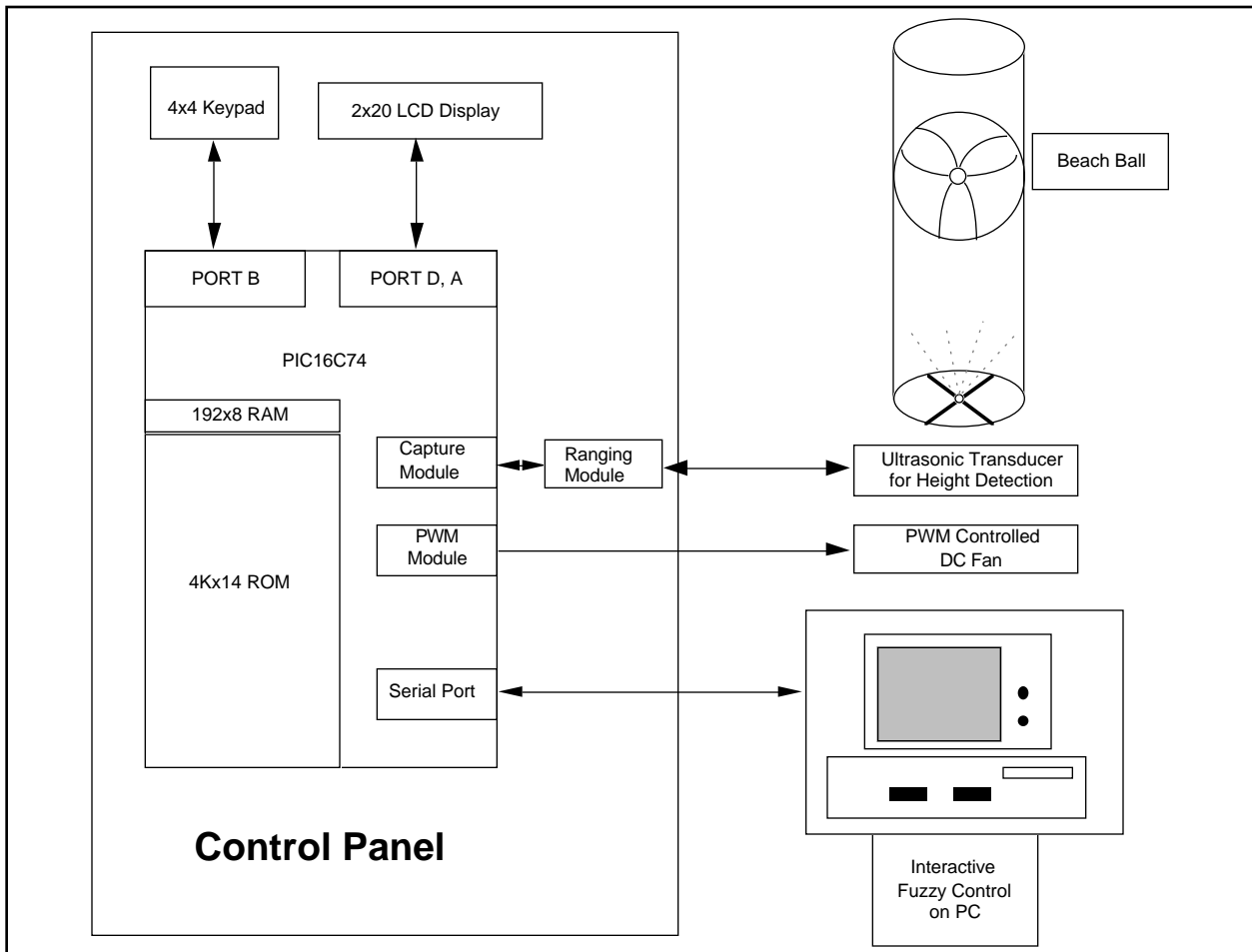
conditions. In addition, fuzzy logic simplifies dealing with non-linearities in systems, and allows for quicker product development cycles.

This application note will step the user through a fuzzy logic control design utilizing sensors. The development tool used is Inform[®] Software's *fuzzyTECH[®]-MP*. The development tool allows for an all-graphical editor, analyzers, and debug capability.

PROJECT DESCRIPTION

The block diagram of the project is shown in Figure 1 and operates as follows.

FIGURE 1: BLOCK DIAGRAM



The control panel prompts the user to enter the desired beach ball height on the 16-key keypad. The keypad input is echoed on the LCD module and the user is prompted for confirmation. Upon confirmation of user input, the control panel initiates a ranging cycle to calculate the current height of the beach ball. The desired height and current height are continually displayed on the LCD module. From the current height, the control panel calculates both the velocity and the delta height (difference in desired height from current height). This information, along with the desired height, is transmitted to the PC via an RS-232 link. The fuzzy logic algorithm, running on the PC, calculates the appropriate duty cycle of the DC fan and transmits this information to the control panel. This emulates a “real world” environment in which system level debug can be done on the PC in real-time. The control panel controls the duty cycle of the DC fan with this input. The above listed ranging process continues indefinitely until interrupted by the user.

The control panel houses an ultrasonic ranging module and the microcontroller. The microcontroller handles all of the peripheral interfaces including the 16-key keypad, the LCD display, the ultrasonic ranging module, and the RS-232 serial link. The project required a microcontroller that could handle the data throughput and all of these peripherals with little or no external components. The microcontroller used was the PIC16C74, which contains 4K of on-chip program memory and 192 bytes of on-chip data memory. Furthermore, the interrupt capabilities, I/O pins, PWM module, capture and compare modules, timer modules, Serial Communications Interface (SCI), and A/D converter make it an excellent fit for the application. In addition, the on-chip pulse-width-modulation (PWM) module allows for a single component (FET) interface for the DC fan control and the ranging module can interface directly to the microcontroller (refer to Application Note AN597, “Implementing Ultrasonic Ranging”).

FUZZY DESIGN

Fuzzy logic first translates the crisp inputs from the sensor into a linguistic description. Then it evaluates the control strategy contained in fuzzy logic rules and translates the result back into a crisp value.

The first step in fuzzy logic control design is system definition. The only possible sources of inputs to the fuzzy logic control algorithm are the ultrasonic transducer, the user, and the DC fan. The key is to decide which of these inputs are significant and which are not. Basically, the behavior of the beach ball was characterized by asking the following questions from the beach ball’s perspective:

- Where am I?
- How far am I from where I want to be?
- How fast am I getting there?
- What external force will get me there?

The nice thing about fuzzy logic control is that the linguistic system definition becomes the control algorithm.

The variables were defined as follows:

- **Current Height** [Where am I?]
- **Delta Height** [How far am I from where I want to be?]
- **Velocity** [How fast am I getting there?]
- **Duty Cycle** [What external force will get me there?]

Defining the variables was the starting point, but for the algorithm to work smoothly, it isn’t good enough to say “the beach ball has velocity;” you need to know to what degree the beach ball has velocity. This is accomplished by defining terms that more fully describe the variable. The combination of variables and terms gives a linguistic description of what is happening to the system. From this, the Velocity variable can be described as having a “positive small velocity” or a “positive big velocity,” not just a “velocity.”

There is no fixed rule on how many terms to define per variable. Typically, three to five terms are defined, but more or less may be needed based on the control algorithm. In retrospect, we probably could have reduced Current Height to three terms and Velocity to five terms. Table 1 lists the four variables that are used for the trade show demo and their associated terms.

Once the linguistic variables are defined, data types and values need to be defined. For this application, data types were defined as 8-bit integers (16-bit definition is also possible). After defining the data types, the shell and code values for each variable were specified. A shell value is used within the fuzzy logic development tool and a code value is used when the code is generated.

The best way to describe shell and code values is using the analogy of a D/A converter. If we have a 5.0V, 8-bit D/A converter, the digital input would correspond to the code value and the analog output would correspond to the shell value. This is, if we write (or pass) a value of 128 to the D/A we would get a 2.51V out. Applying this analogy to our project, we would pass a crisp value (digital) to the fuzzy world and the fuzzy world would use the fuzzy value (analog).

Therefore, when we define shell and code values, we are basically defining the "D/A converter." For example, you can define the shell value for Duty Cycle to be a minimum of 0 and a maximum of 100 (percent). Therefore, within the fuzzy logic development tool, Duty Cycle will take on a value between 0 and 100, inclusive.

The code value is limited by the data type, but can take on any or all of the digital range. That is, if the shell value is 0 to 100, the code values could be defined as 0 to 100. But to get full resolution, the code value should be defined over the entire range (i.e., 0 to 255 for 8-bit data types). The code values and shell values were defined as shown in Table 2. Note that for the height and velocity variables, the shell values are scaled by 2 (i.e., a Current Height with a crisp value of 60 would correspond to 30 inches).

TABLE 1: INPUT AND OUTPUT VARIABLES AND TERMS

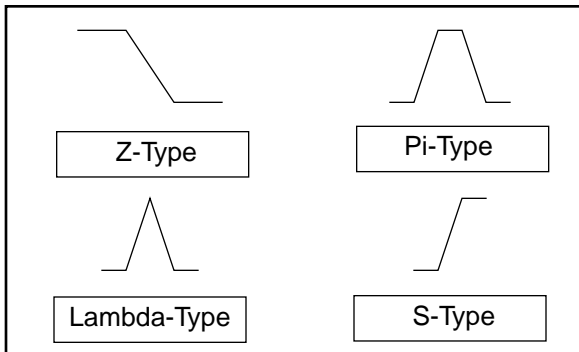
Input Variables			Output Variable
Current Height	Delta Height	Velocity	Duty Cycle
very lo	neg big	neg big	very slo
lo	neg small	neg med	slo
medium	zero	neg small	medium slo
hi	pos small	zero	medium
very hi	pos big	pos small	medium fast
		pos med	fast
		pos big	very fast

TABLE 2: SHELL AND CODE VALUES

Variable	Shell Value		Code Value	
	Min.	Max.	Min.	Max.
Current Height	0	120	0	255
Delta Height	-50	50	0	255
Velocity	-5	5	0	255
Duty Cycle	0	255	0	255

Next, the membership functions were defined to further describe the variables. The fuzzy logic development tool creates the membership functions automatically. This gives a good starting point, but the membership functions still need to be fine-tuned during the debug phase. In this application, only the linear shaped functions (Pi, Z, S and Lambda types) were used as seen in Figure 2.

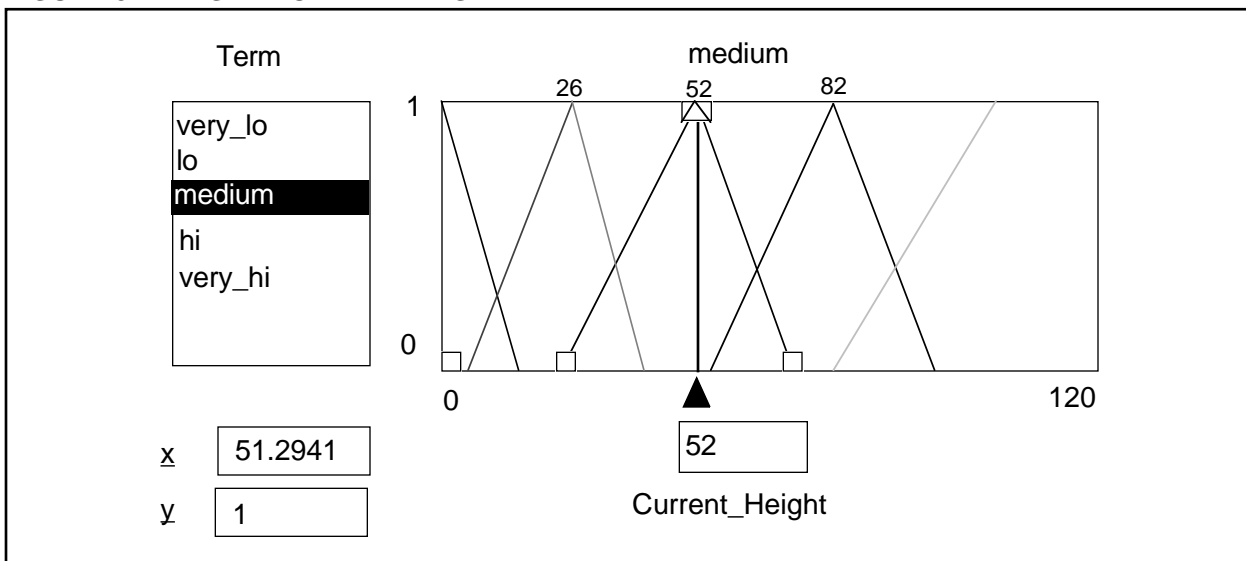
FIGURE 2: STANDARD MEMBERSHIP FUNCTION TYPES



FUZZIFICATION

Fuzzification entails translating a crisp value into a fuzzy value. Once all of the variables have been defined, the interfaces between the variables need to be defined. The interfaces for the input variables contain the fuzzification procedures. In defining the interfaces, the input variable's fuzzification method needs to be defined. The computation of fuzzification is carried out at runtime for code efficiency. The type of fuzzification used in this project is membership function computation. This is largely due to the code space efficiency and accuracy associated with this method. Once fuzzification has taken place, the algorithm is performed in the fuzzy world according to the rule base.

FIGURE 3: DEGREE OF MEMBERSHIP



FUZZY RULE BASE

The entire fuzzy inference is contained within the rule blocks of a system. For example, if the beach ball is near the top of the tube and it was commanded to be near the bottom of the tube, the rule that described the situation would be:

IF CURRENT HEIGHT = VERY HI

AND DELTA HEIGHT = NEGATIVE BIG

THEN DUTY CYCLE = SLOW

The above rule describes one situation, but the rule definition would continue until the system was adequately described. The rule block is the collection of all rules that describe the system.

The rules of the rule block can also be defined in terms of how much a specific rule is supported when calculating inference. The support of a rule, or plausibility, is known as the degree of support for that rule. A plausible rule is defined by a 1.0, a totally implausible rule is defined by 0.0. In this project all rules are fully supported.

The degree to which a crisp value belongs to a term is known as the degree of membership. For example, the terms Medium and Hi for the variable Current Height were defined as a Lambda-type membership function centered around the crisp values 52 (26 inches) and 82 (41 inches), respectively, as shown in Figure 3.

Therefore, if the beach ball was at 26 inches, the degree of membership would be 1.0 for Medium and 0.0 for Hi. However, as the beach ball rises in height, the degree of membership for the term Medium would decrease and the degree of membership for the term Hi would increase. The interplay of these linguistic variable terms is controlled by the rule base. The rule base defines not only the relationship between the terms, but also how much each rule is supported, as described previously.

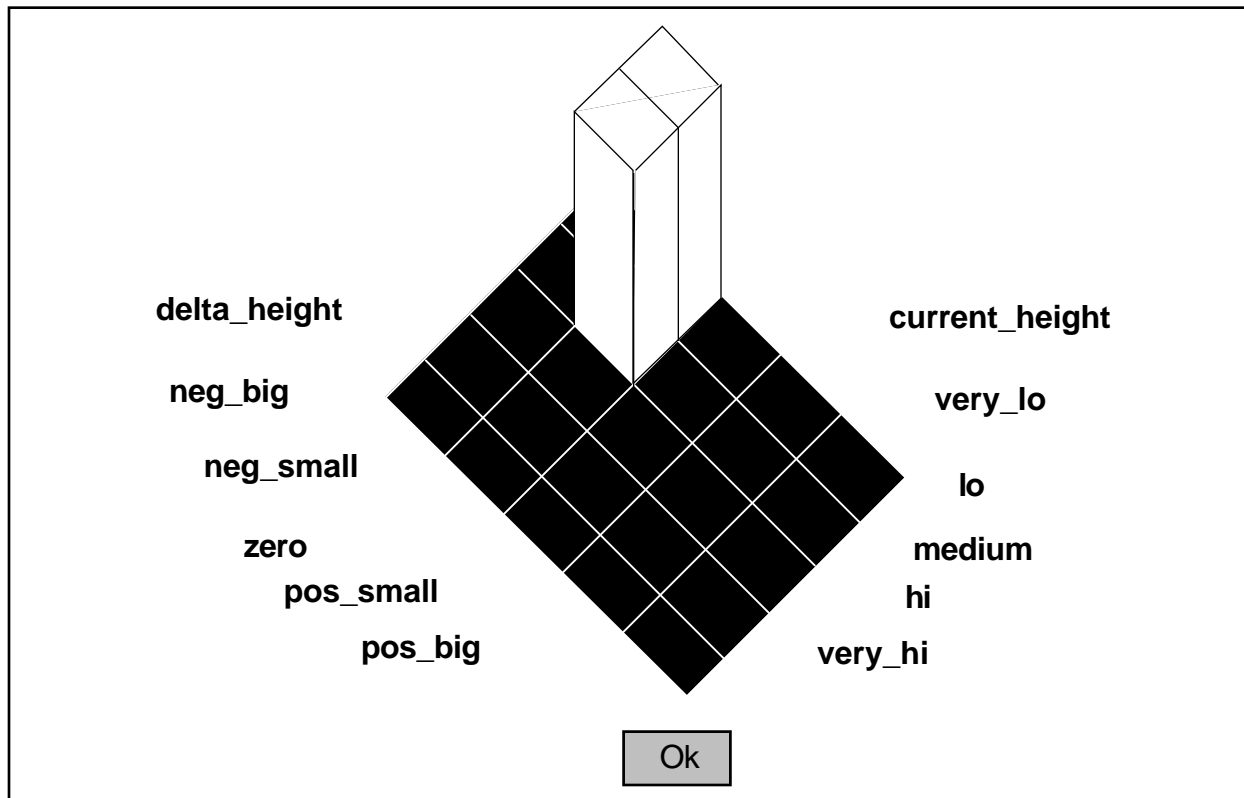
From the list of rules, a Fuzzy Associative Map (FAM) is constructed (see below). The FAM shows the plausibility (degree of support) of each rule as seen in Figure 4 and Figure 5.

FIGURE 4: MATRIX RULE EDITOR WITH FAM RULES

The screenshot displays the Matrix Rule Editor interface. On the left, a matrix is defined with 'delta_height' and 'current_height' as inputs and 'neg_big', 'neg_small', 'pos_big', and 'pos_small' as outputs. The matrix cells are represented by a grid of black and white squares. A small icon is visible in the top-left cell. To the right of the matrix is a 'Show ...' menu with three radio button options: 'Degree of Support' (selected), 'Input Aggregation', and 'Composition with Degree of Support'. Below this menu is a 'Degree of Support' slider control, currently set to 0.000. At the bottom, the 'IF' section contains three vertical lists of linguistic terms: 'current_height' (very_lo, lo, medium, hi, very_hi), 'delta_height' (neg_big, neg_small, zero, pos_small, pos_big), and 'velocity' (neg_big, neg_med, neg_small, zero, pos_small). Each list has a vertical scrollbar and a small icon. The 'THEN' section contains a single vertical list of linguistic terms: 'duty_cycle' (very_slow, slow, med_slow, medium, med_fast), also with a vertical scrollbar and a small icon.

AN600

FIGURE 5: 3-D RULE DISPLAY



DEFUZZIFICATION

Defuzzification entails translating a fuzzy value to a crisp value. The interface for the output variables contains the defuzzification procedures. For most control applications (and this project), the center-of-maximum (CoM) method is used for defuzzification. CoM evaluates more than one output term as valid and compromises between them by computing a weighted mean of the term membership maxima. Example 1 and Figure 6 show the defuzzification of the linguistic variable Duty Cycle using CoM.

EXAMPLE 1: DEFUZZIFICATION OF DUTY CYCLE

The crisp values of the three input variables are as follows:

Current Height: 30

Delta Height: 0

Velocity: 0

The crisp value can be calculated using the CoM method with the following equation.

$$C = \frac{\sum_i [I \cdot \max_x (M) \cdot \arg (\max_x (M))]}{\sum_i I}$$

c = crisp output value

i = linguistic term

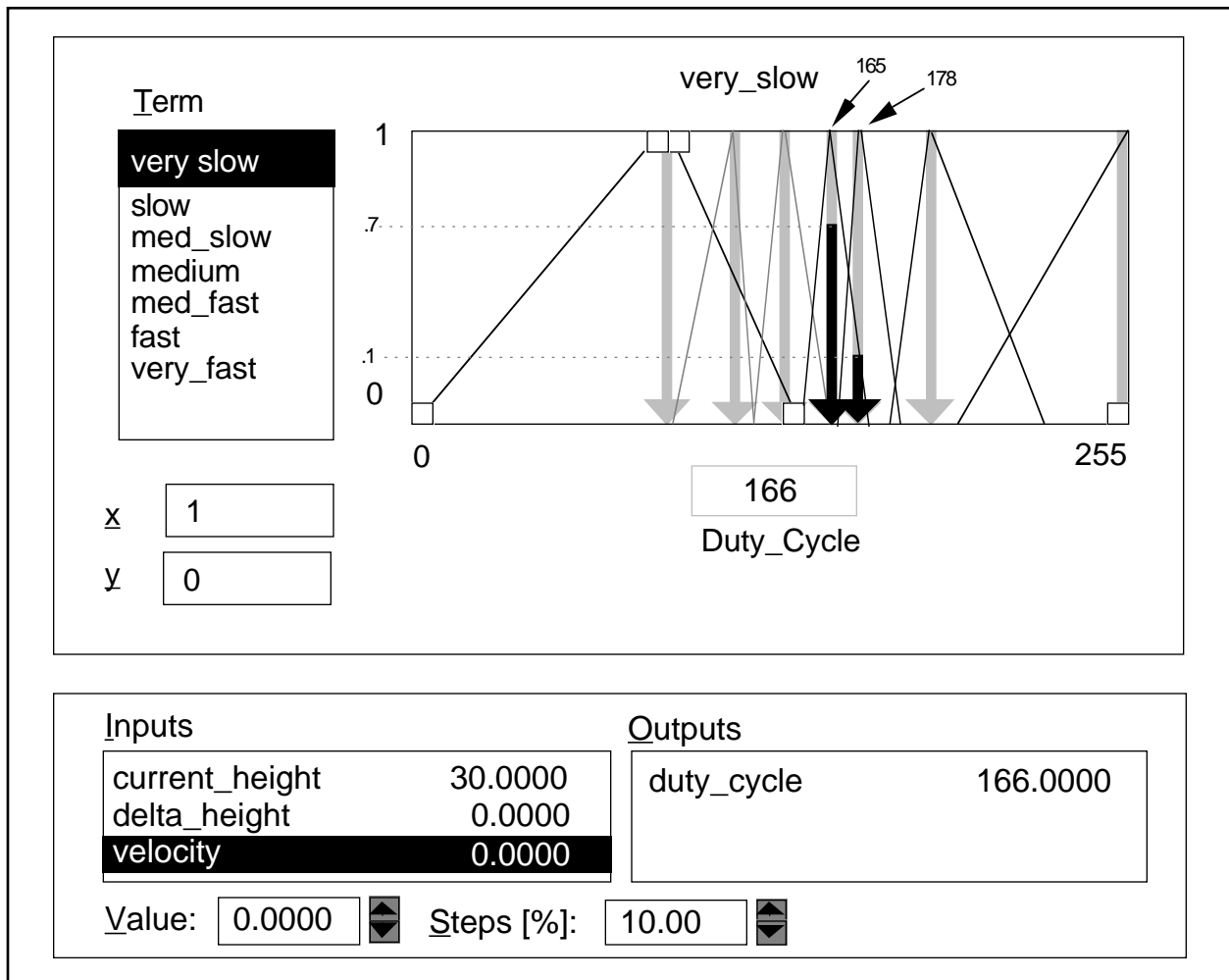
I = inference result

M = membership function of linguistic term

For this example, when the crisp values are fuzzified, the Duty Cycle variable is defined to be mostly "medium" (degree of membership of 0.7) and somewhat "medium fast" (degree of membership 0.1). The arguments for the "medium" and "medium fast" term membership maxima are 165 and 178, respectively.

$$\frac{((0.7 \cdot 1.0 \cdot 165) + (0.1 \cdot 1.0 \cdot 178))}{(0.7 + 0.1)} = 166$$

FIGURE 6: DEFUZZIFICATION OF DUTY CYCLE



AN600

DEBUGGING

In serial debug mode, one can graphically adjust the variable terms and see the results in "real time." On this project, the first variable adjusted was the Duty Cycle variable. Duty Cycle was adjusted so that the beach ball reached 30 inches (Figure 7). The Delta Height terms were fine-tuned -- negative small, zero, and positive small were bunched together -- and the beach ball stabilized at 30 inches (Figure 8). There was virtually no fluctuation in the height. In order for the system to self-correct for environmental (external) changes, the Velocity variable was used. The velocity variable is calculated by the difference in height between consecutive height calculations. A few rules were added that used the Velocity variable to nudge the ball into place when the environmental conditions changed (Figure 9).

Another advantage of fuzzy logic is that it simplifies dealing with non-linearities of the system. The system was highly non-linear, so it was tested at the extremes and moving the beach ball at different rates from one extreme to the other. The Current Height variable needed almost no adjustment (Figure 10). The variable that required the most work was the Duty Cycle variable, but in less than a day, the algorithm was working well within specifications. The beach ball could go from a resting position, with the DC fan off, to the maximum allowable height of 42 inches in less than 8 seconds with no overshoot. Operation between the minimum and maximum height was much quicker, also with no overshoot.

The final graphical representation of the linguistic variables are shown in Figure 7 through Figure 10.

FIGURE 7: DUTY CYCLE VARIABLE

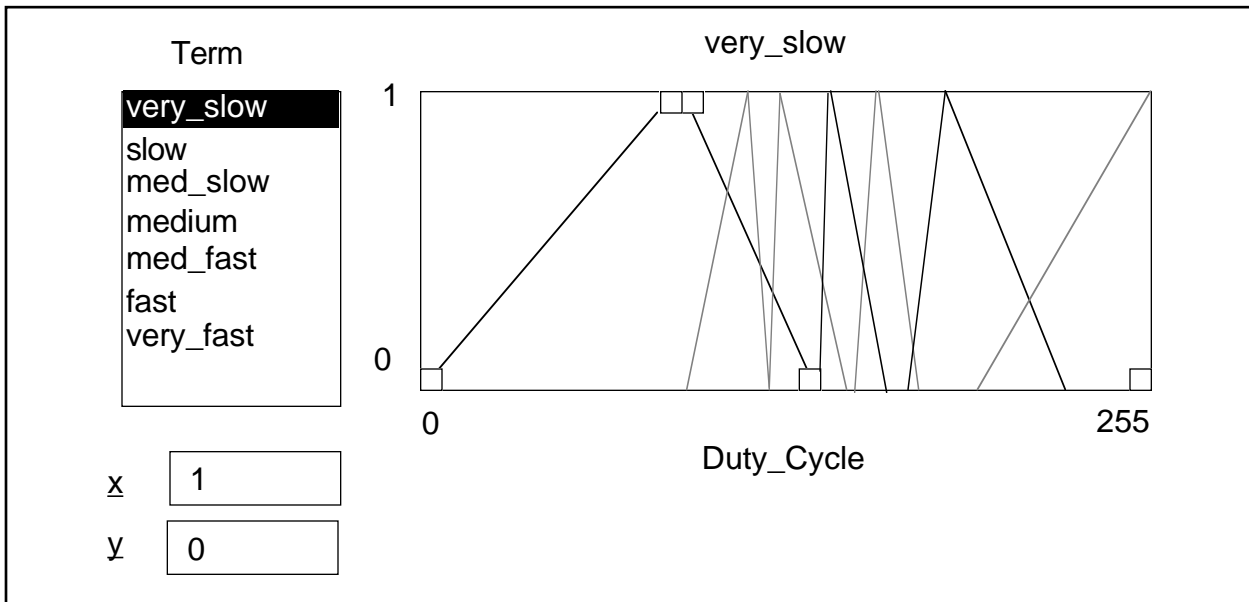


FIGURE 8: DELTA HEIGHT VARIABLE

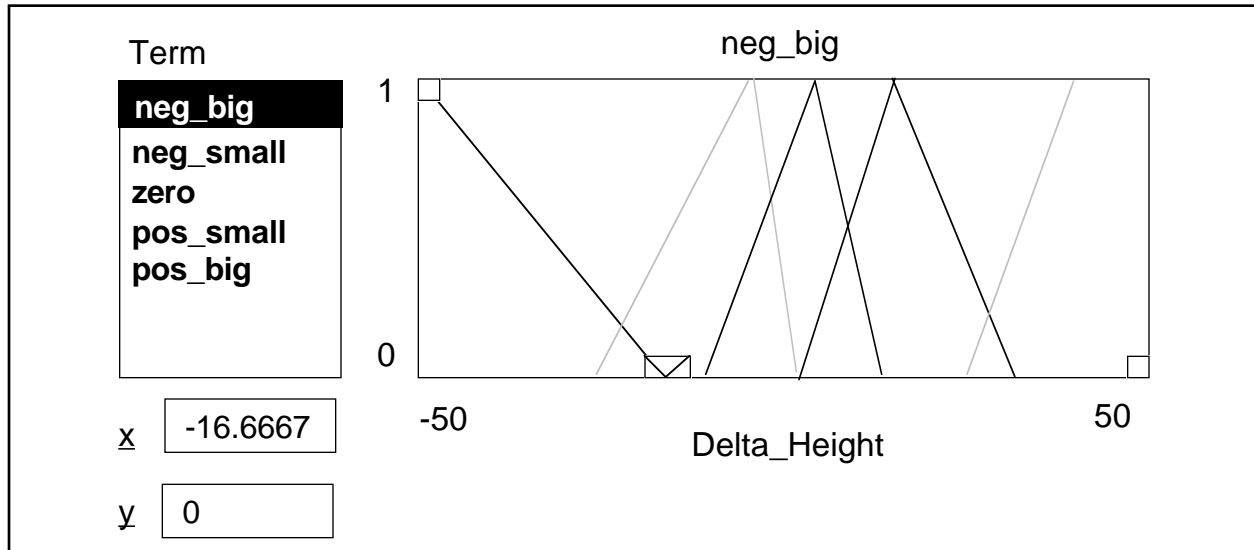


FIGURE 9: VELOCITY VARIABLE

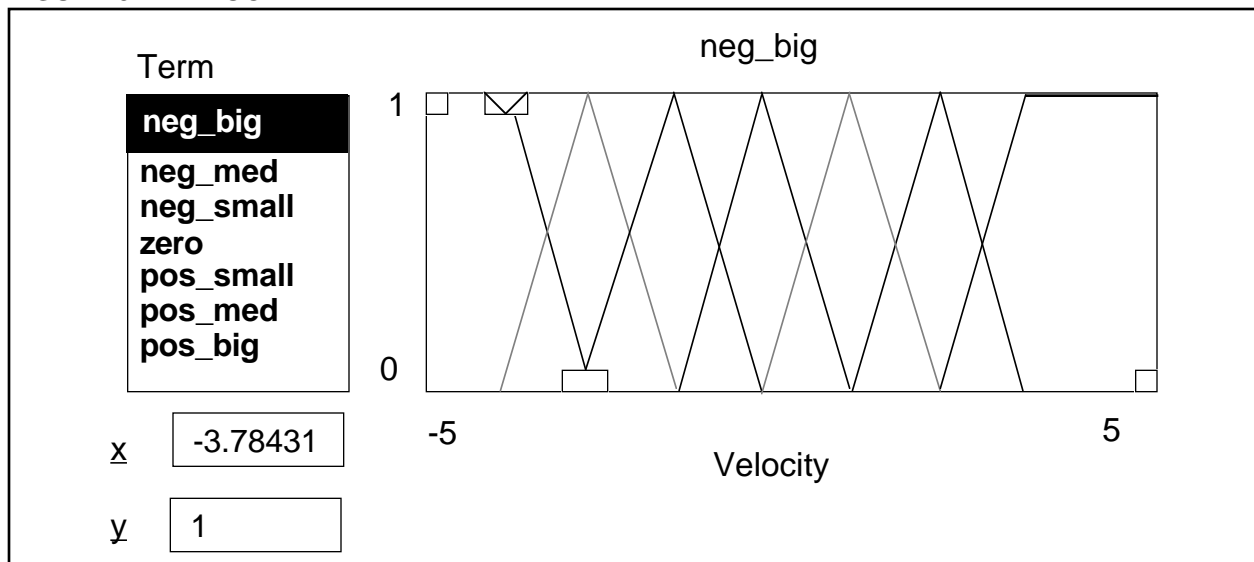
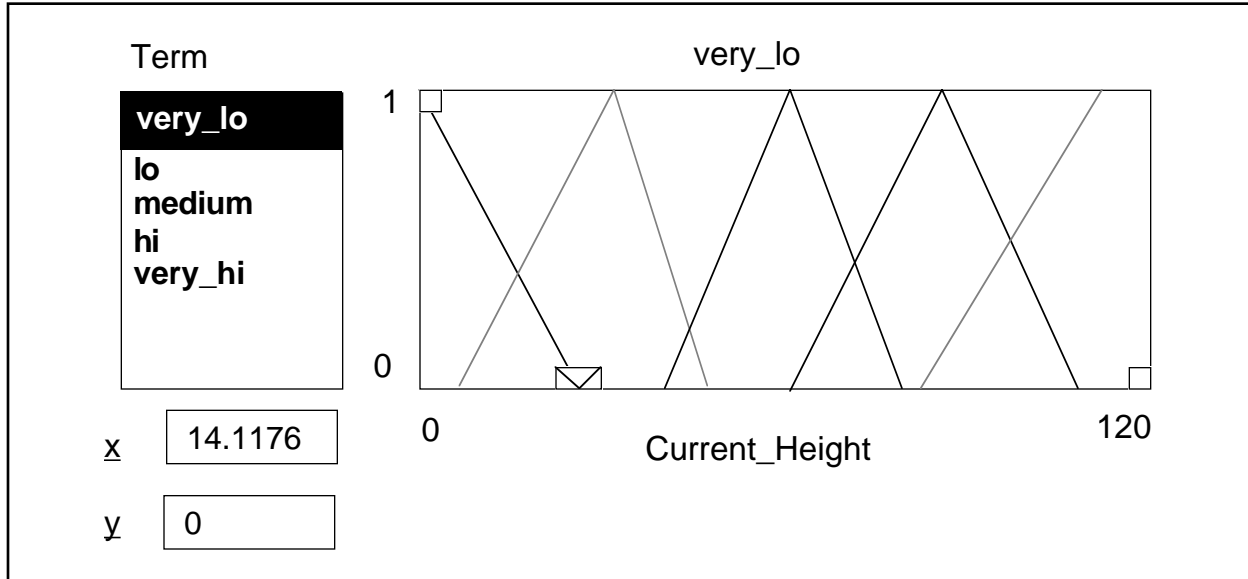


FIGURE 10: CURRENT HEIGHT VARIABLE



INTEGRATION

The system parameters and graphical variable representations are captured in a Fuzzy Technology Language (FTL) file. The FTL file is a vendor and hardware independent language which defines the fuzzy logic based system. The FTL file for this project can be seen in Appendix A.

The FTL file is used to generate the public variable definitions and code which can be embedded in the microcontroller. The appropriate device family from the pre-assembler code are generated by simply selecting the compile pull-down menu. Once the pre-assembler file is generated, the "hooks" to the main program must be added.

The best way to embed the code is to use the template MYMAIN.ASM. The template for each of the families of devices (PIC16C5X, PIC16CXX and PIC17CXX) is included in the *fuzzyTECH[®]-MP* development kit. The template shown in Appendix B is for the PIC16CXX family.

The file MYMAIN.ASM should contain your program in the "main_loop" section. The only other modifications required to the template are listed below and are specified in the left hand column of Appendix B.

1. Processor Type definition
2. Code Start Address
3. Fuzzy RAM Start Address
4. Include Public Variable Definition file (myproj.var), which was created by *fuzzyTECH[®]-MP*
5. Include Pre-Assembler Code (myproj.asm) which was created by *fuzzyTECH[®]-MP*
6. Call Initialization (initmyproj) which was created by *fuzzyTECH[®]-MP*
7. Set Crisp Input Value(s)
8. Call Fuzzy Logic System (myproj)
9. Read Crisp Output Value(s)

For this project, the fuzzy logic algorithm assembled to 704 words of program memory and 41 bytes of data memory.

SUMMARY

This project demonstrates many aspects of fuzzy logic control - quick development cycle, real-time debug, sensor integration, and non-linear system control. The total development time for the application took less than a week and performed well within system specifications.

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: FUZZY TECHNOLOGY LANGUAGE FILE

```

PROJECT {
  NAME = B_BALL.FTL;
  AUTHOR = ROBERT SCHREIBER;
  DATEFORMAT = M.D.YY;
  LASTCHANGE = 9.16.94;
  CREATED    = 9.14.94;
  SHELL = MP;
  COMMENT {
} /* COMMENT */
SHELLOPTIONS {
  ONLINE_REFRESHTIME = 55;
  ONLINE_TIMEOUTCOUNT = 0;
  ONLINE_CODE = OFF;
  TRACE_BUFFER = (OFF, PAR(10000));
  BSUM_AGGREGATION = OFF;
  PUBLIC_IO = ON;
  FAST_CMBF = ON;
  FAST_COA = OFF;
  SCALE_MBF = OFF;
  FILE_CODE = OFF;
  BTYPE = 8_BIT;
} /* SHELLOPTIONS */
MODEL {
  VARIABLE_SECTION {
    LVAR {
      NAME      = current_height;
      BASEVAR = Current_Height;
      LVRANGE = MIN(0.000000), MAX(120.000000),
                MINDEF(0), MAXDEF(255),
                DEFAULT_OUTPUT(120.000000);
      RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                SHOWGRID (ON), SNAPTOGRID(ON);
    }
    TERM {
      TERMNAME = very_lo;
      POINTS = (0.000000, 1.000000),
                (14.117647, 0.000000),
                (120.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (255), GREEN (0), BLUE (0);
    }
    TERM {
      TERMNAME = lo;
      POINTS = (0.000000, 0.000000),
                (5.176471, 0.000000),
                (24.941176, 1.000000),
                (40.941176, 0.000000),
                (120.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (0), GREEN (255), BLUE (0);
    }
    TERM {
      TERMNAME = medium;
      POINTS = (0.000000, 0.000000),
                (27.294118, 0.000000),
                (51.294118, 1.000000),

```

AN600

```
        (66.352941, 0.000000),
        (120.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (0), BLUE (255);
}
TERM {
    TERMNAME = hi;
    POINTS = (0.000000, 0.000000),
             (55.529412, 0.000000),
             (82.352941, 1.000000),
             (106.352941, 0.000000),
             (120.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (128), GREEN (0), BLUE (0);
}
TERM {
    TERMNAME = very_hi;
    POINTS = (0.000000, 0.000000),
             (73.411765, 0.000000),
             (113.411765, 1.000000),
             (120.000000, 1.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (128), BLUE (0);
}
} /* LVAR */
LVAR {
    NAME      = delta_height;
    BASEVAR   = Delta_Height;
    LVRANGE   = MIN(-50.000000), MAX(50.000000),
              MINDEF(0), MAXDEF(255),
              DEFAULT_OUTPUT(-50.000000);
    RESOLUTION = XGRID(0.000000), YGRID(1.000000),
              SHOWGRID (ON), SNAPTOGRID(ON);
    TERM {
        TERMNAME = neg_big;
        POINTS = (-50.000000, 1.000000),
                (-16.666667, 0.000000),
                (50.000000, 0.000000);
        SHAPE = LINEAR;
        COLOR = RED (255), GREEN (0), BLUE (0);
    }
    TERM {
        TERMNAME = neg_small;
        POINTS = (-50.000000, 0.000000),
                (-21.764706, 0.000000),
                (-6.470588, 1.000000),
                (-0.588235, 0.000000),
                (50.000000, 0.000000);
        SHAPE = LINEAR;
        COLOR = RED (0), GREEN (255), BLUE (0);
    }
    TERM {
        TERMNAME = zero;
        POINTS = (-50.000000, 0.000000),
                (-12.352941, 0.000000),
                (0.196078, 1.000000),
                (13.529412, 0.000000),
                (50.000000, 0.000000);
        SHAPE = LINEAR;
        COLOR = RED (0), GREEN (0), BLUE (255);
    }
}
```

```

}
TERM {
    TERMNAME = pos_small;
    POINTS = (-50.000000, 0.000000),
            (0.196078, 0.000000),
            (10.000000, 1.000000),
            (10.392157, 1.000000),
            (32.745098, 0.000000),
            (50.000000, 0.000000);

    SHAPE = LINEAR;
    COLOR = RED (128), GREEN (0), BLUE (0);
}
TERM {
    TERMNAME = pos_big;
    POINTS = (-50.000000, 0.000000),
            (26.470588, 0.000000),
            (39.803922, 1.000000),
            (50.000000, 1.000000);

    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (128), BLUE (0);
}
} /* LVAR */
LVAR {
    NAME      = duty_cycle;
    BASEVAR   = Duty_Cycle;
    LVRANGE   = MIN(0.000000), MAX(255.000000),
              MINDEF(0), MAXDEF(255),
              DEFAULT_OUTPUT(0.000000);
    RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                SHOWGRID (ON), SNAPTOGRID(ON);

    TERM {
        TERMNAME = very_slow;
        POINTS = (0.000000, 0.000000),
                (1.000000, 0.000000),
                (103.000000, 1.000000),
                (113.000000, 1.000000),
                (147.000000, 0.000000),
                (255.000000, 0.000000);

        SHAPE = LINEAR;
        COLOR = RED (255), GREEN (0), BLUE (0);
    }
    TERM {
        TERMNAME = slow;
        POINTS = (0.000000, 0.000000),
                (108.000000, 0.000000),
                (127.000000, 1.000000),
                (131.000000, 0.000000),
                (255.000000, 0.000000);

        SHAPE = LINEAR;
        COLOR = RED (0), GREEN (255), BLUE (0);
    }
    TERM {
        TERMNAME = med_slow;
        POINTS = (0.000000, 0.000000),
                (133.000000, 0.000000),
                (142.000000, 1.000000),
                (162.000000, 0.000000),
                (255.000000, 0.000000);

        SHAPE = LINEAR;
        COLOR = RED (0), GREEN (128), BLUE (128);
    }
}

```

AN600

```
}
TERM {
  TERMNAME = medium;
  POINTS = (0.000000, 0.000000),
           (151.000000, 0.000000),
           (164.000000, 1.000000),
           (166.000000, 1.000000),
           (174.000000, 0.000000),
           (255.000000, 0.000000);

  SHAPE = LINEAR;
  COLOR = RED (0), GREEN (0), BLUE (255);
}
TERM {
  TERMNAME = med_fast;
  POINTS = (0.000000, 0.000000),
           (166.000000, 0.000000),
           (178.000000, 1.000000),
           (193.000000, 0.000000),
           (255.000000, 0.000000);

  SHAPE = LINEAR;
  COLOR = RED (255), GREEN (0), BLUE (128);
}
TERM {
  TERMNAME = fast;
  POINTS = (0.000000, 0.000000),
           (189.000000, 0.000000),
           (202.000000, 1.000000),
           (232.000000, 0.000000),
           (255.000000, 0.000000);

  SHAPE = LINEAR;
  COLOR = RED (128), GREEN (0), BLUE (0);
}
TERM {
  TERMNAME = very_fast;
  POINTS = (0.000000, 0.000000),
           (206.000000, 0.000000),
           (255.000000, 1.000000);

  SHAPE = LINEAR;
  COLOR = RED (0), GREEN (128), BLUE (0);
}
} /* LVAR */
LVAR {
  NAME      = velocity;
  BASEVAR  = Velocity;
  LVRANGE  = MIN(-5.000000), MAX(5.000000),
            MINDEF(0), MAXDEF(255),
            DEFAULT_OUTPUT(0.000000);
  RESOLUTION = XGRID(0.000000), YGRID(1.000000),
              SHOWGRID (OFF), SNAPTOGRID(ON);

  TERM {
    TERMNAME = neg_big;
    POINTS = (-5.000000, 1.000000),
             (-3.784314, 1.000000),
             (-2.529412, 0.000000),
             (5.000000, 0.000000);

    SHAPE = LINEAR;
    COLOR = RED (255), GREEN (0), BLUE (0);
  }
  TERM {
    TERMNAME = neg_med;

```



```
POINTS = (-5.000000, 0.000000),
          (-3.784314, 0.000000),
          (-2.529412, 1.000000),
          (-1.274510, 0.000000),
          (5.000000, 0.000000);
SHAPE = LINEAR;
COLOR = RED (0), GREEN (255), BLUE (0);
}
TERM {
  TERMNAME = neg_small;
  POINTS = (-5.000000, 0.000000),
          (-2.568627, 0.000000),
          (-1.313725, 1.000000),
          (-0.058824, 0.000000),
          (5.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (0), GREEN (0), BLUE (255);
}
TERM {
  TERMNAME = zero;
  POINTS = (-5.000000, 0.000000),
          (-1.000000, 0.000000),
          (-0.019608, 1.000000),
          (0.960784, 0.000000),
          (5.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (128), GREEN (0), BLUE (0);
}
TERM {
  TERMNAME = pos_small;
  POINTS = (-5.000000, 0.000000),
          (-0.137255, 0.000000),
          (1.117647, 1.000000),
          (2.372549, 0.000000),
          (5.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (0), GREEN (128), BLUE (0);
}
TERM {
  TERMNAME = pos_med;
  POINTS = (-5.000000, 0.000000),
          (1.078431, 0.000000),
          (2.333333, 1.000000),
          (3.588235, 0.000000),
          (5.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (0), GREEN (0), BLUE (128);
}
TERM {
  TERMNAME = pos_big;
  POINTS = (-5.000000, 0.000000),
          (2.294118, 0.000000),
          (3.549020, 1.000000),
          (5.000000, 1.000000);
  SHAPE = LINEAR;
  COLOR = RED (255), GREEN (0), BLUE (128);
}
} /* LVAR */
} /* VARIABLE_SECTION */
```

AN600

```
OBJECT_SECTION {
  INTERFACE {
    INPUT = (current_height, FCMBF);
    POS = -213, -137;
    RANGECHECK = ON;
  }
  INTERFACE {
    INPUT = (delta_height, FCMBF);
    POS = -216, -83;
    RANGECHECK = ON;
  }
  INTERFACE {
    OUTPUT = (duty_cycle, COM);
    POS = 158, -79;
    RANGECHECK = ON;
  }
  RULEBLOCK {
    INPUT = current_height, delta_height, velocity;
    OUTPUT = duty_cycle;
    AGGREGATION = (MIN_MAX, PAR (0.000000));
    COMPOSITION = (GAMMA, PAR (0.000000));
    POS = -39, -113;
    RULES {
      IF current_height = very_lo
        AND delta_height = neg_big
      THEN duty_cycle = slow WITH 1.000;
      IF current_height = very_lo
        AND delta_height = neg_small
      THEN duty_cycle = med_slow WITH 1.000;
      IF current_height = very_lo
        AND delta_height = zero
      THEN duty_cycle = medium WITH 1.000;
      IF current_height = very_lo
        AND delta_height = pos_small
      THEN duty_cycle = fast WITH 1.000;
      IF current_height = very_lo
        AND delta_height = pos_big
      THEN duty_cycle = very_fast WITH 1.000;
      IF current_height = lo
        AND delta_height = neg_big
      THEN duty_cycle = slow WITH 1.000;
      IF current_height = lo
        AND delta_height = neg_small
      THEN duty_cycle = med_slow WITH 1.000;
      IF current_height = lo
        AND delta_height = zero
      THEN duty_cycle = medium WITH 1.000;
      IF current_height = lo
        AND delta_height = pos_small
      THEN duty_cycle = fast WITH 1.000;
      IF current_height = lo
        AND delta_height = pos_big
      THEN duty_cycle = very_fast WITH 1.000;
      IF current_height = medium
        AND delta_height = neg_big
      THEN duty_cycle = very_slow WITH 1.000;
      IF current_height = medium
        AND delta_height = neg_small
      THEN duty_cycle = med_slow WITH 1.000;
      IF current_height = medium

```

```
    AND delta_height = zero
THEN  duty_cycle = med_fast   WITH 1.000;
IF    current_height = medium
    AND delta_height = pos_small
THEN  duty_cycle = fast     WITH 1.000;
IF    current_height = medium
    AND delta_height = pos_big
THEN  duty_cycle = very_fast WITH 1.000;
IF    current_height = hi
    AND delta_height = neg_big
THEN  duty_cycle = very_slow WITH 1.000;
IF    current_height = hi
    AND delta_height = neg_small
THEN  duty_cycle = med_slow  WITH 1.000;
IF    current_height = hi
    AND delta_height = zero
THEN  duty_cycle = med_fast  WITH 1.000;
IF    current_height = hi
    AND delta_height = pos_small
THEN  duty_cycle = fast     WITH 1.000;
IF    current_height = hi
    AND delta_height = pos_big
THEN  duty_cycle = very_fast WITH 1.000;
IF    current_height = very_hi
    AND delta_height = neg_big
THEN  duty_cycle = very_slow WITH 1.000;
IF    current_height = very_hi
    AND delta_height = neg_small
THEN  duty_cycle = slow     WITH 1.000;
IF    current_height = very_hi
    AND delta_height = zero
THEN  duty_cycle = med_slow  WITH 1.000;
IF    current_height = very_hi
    AND delta_height = pos_small
THEN  duty_cycle = medium   WITH 1.000;
IF    current_height = very_hi
    AND delta_height = pos_big
THEN  duty_cycle = very_fast WITH 1.000;
IF    current_height = very_lo
    AND delta_height = neg_small
    AND velocity = zero
THEN  duty_cycle = very_slow WITH 1.000;
IF    current_height = very_lo
    AND delta_height = neg_small
    AND velocity = pos_small
THEN  duty_cycle = very_slow WITH 1.000;
IF    current_height = very_lo
    AND delta_height = neg_small
    AND velocity = pos_med
THEN  duty_cycle = very_slow WITH 1.000;
IF    current_height = very_lo
    AND delta_height = neg_small
    AND velocity = pos_big
THEN  duty_cycle = very_slow WITH 1.000;
IF    current_height = very_lo
    AND delta_height = pos_small
    AND velocity = zero
THEN  duty_cycle = fast     WITH 1.000;
IF    current_height = very_lo
    AND delta_height = pos_small
```

AN600

```
    AND velocity = neg_small
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = very_lo
    AND delta_height = pos_small
    AND velocity = neg_med
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = very_lo
    AND delta_height = pos_small
    AND velocity = neg_big
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = lo
    AND delta_height = neg_small
    AND velocity = zero
THEN duty_cycle = very_slow WITH 1.000;
IF   current_height = lo
    AND delta_height = neg_small
    AND velocity = pos_small
THEN duty_cycle = very_slow WITH 1.000;
IF   current_height = lo
    AND delta_height = neg_small
    AND velocity = pos_med
THEN duty_cycle = very_slow WITH 1.000;
IF   current_height = lo
    AND delta_height = neg_small
    AND velocity = pos_big
THEN duty_cycle = very_slow WITH 1.000;
IF   current_height = lo
    AND delta_height = pos_small
    AND velocity = zero
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = lo
    AND delta_height = pos_small
    AND velocity = neg_small
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = lo
    AND delta_height = pos_small
    AND velocity = neg_med
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = lo
    AND delta_height = pos_small
    AND velocity = neg_big
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = medium
    AND delta_height = neg_small
    AND velocity = zero
THEN duty_cycle = slow    WITH 1.000;
IF   current_height = medium
    AND delta_height = neg_small
    AND velocity = pos_small
THEN duty_cycle = slow    WITH 1.000;
IF   current_height = medium
    AND delta_height = neg_small
    AND velocity = pos_med
THEN duty_cycle = slow    WITH 1.000;
IF   current_height = medium
    AND delta_height = neg_small
    AND velocity = pos_big
THEN duty_cycle = slow    WITH 1.000;
IF   current_height = medium
    AND delta_height = pos_small
```

```
    AND velocity = zero
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = medium
    AND delta_height = pos_small
    AND velocity = neg_small
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = medium
    AND delta_height = pos_small
    AND velocity = neg_med
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = medium
    AND delta_height = pos_small
    AND velocity = neg_big
THEN duty_cycle = fast    WITH 1.000;
IF   current_height = hi
    AND delta_height = neg_small
    AND velocity = zero
THEN duty_cycle = med_slow WITH 1.000;
IF   current_height = hi
    AND delta_height = neg_small
    AND velocity = pos_small
THEN duty_cycle = med_slow WITH 1.000;
IF   current_height = hi
    AND delta_height = neg_small
    AND velocity = pos_med
THEN duty_cycle = med_slow WITH 1.000;
IF   current_height = hi
    AND delta_height = neg_small
    AND velocity = pos_big
THEN duty_cycle = med_slow WITH 1.000;
IF   current_height = hi
    AND delta_height = pos_small
    AND velocity = zero
THEN duty_cycle = very_fast WITH 1.000;
IF   current_height = hi
    AND delta_height = pos_small
    AND velocity = neg_small
THEN duty_cycle = very_fast WITH 1.000;
IF   current_height = hi
    AND delta_height = pos_small
    AND velocity = neg_med
THEN duty_cycle = very_fast WITH 1.000;
IF   current_height = hi
    AND delta_height = pos_small
    AND velocity = neg_big
THEN duty_cycle = very_fast WITH 1.000;
IF   current_height = very_hi
    AND delta_height = neg_small
    AND velocity = zero
THEN duty_cycle = medium    WITH 1.000;
IF   current_height = very_hi
    AND delta_height = neg_small
    AND velocity = pos_small
THEN duty_cycle = medium    WITH 1.000;
IF   current_height = very_hi
    AND delta_height = neg_small
    AND velocity = pos_med
THEN duty_cycle = medium    WITH 1.000;
IF   current_height = very_hi
    AND delta_height = neg_small
```

AN600

```
        AND velocity = pos_big
    THEN duty_cycle = medium    WITH 1.000;
    IF    current_height = very_hi
        AND delta_height = pos_small
        AND velocity = zero
    THEN duty_cycle = very_fast  WITH 1.000;
    IF    current_height = very_hi
        AND delta_height = pos_small
        AND velocity = neg_small
    THEN duty_cycle = very_fast  WITH 1.000;
    IF    current_height = very_hi
        AND delta_height = pos_small
        AND velocity = neg_med
    THEN duty_cycle = very_fast  WITH 1.000;
    IF    current_height = very_hi
        AND delta_height = pos_small
        AND velocity = neg_big
    THEN duty_cycle = very_fast  WITH 1.000;
} /* RULES */
}
INTERFACE {
    INPUT = (velocity, FCMBF);
    POS = -211, -29;
    RANGECHECK = ON;
}
} /* OBJECT_SECTION */
} /* MODEL */
} /* PROJECT */
TERMINAL {
    BAUDRATE      = 9600;
    STOPBITS      = 1;
    PROTOCOL      = NO;
    CONNECTION    = PORT1;
    INPUTBUFFER   = 4096;
    OUTPUTBUFFER  = 1024;
} /* TERMINAL */
```

APPENDIX B: MYMAIN.ASM TEMPLATE FOR THE PIC16CXX FAMILY

```

1 → PROCESSOR 16C71
;-----
;- USER MAIN FILE
;-----
2 → CODE_START      EQU    0x100  ;code startadr for 16C71
   RESET_ADR       EQU    0x000  ;reset vector
3 → FUZZY_RAM_START EQU    0x00C  ;first free RAM location for 16C71
4 → include         "myproj.var" ;include preassembler variables
   CBLOCK
       user1          ;starts after fuzzy ram locations
       ;reserve 1 byte (example)
   ENDC
   ORG CODE_START    ;example start adress for code

mymain
6 → call           initmyproj   ;call init once
main_loop
   movlw           000          ;example
7 → movwf          lv0_Input_1  ;set 1st crisp input
   movlw           0A0          ;example
7 → movwf          lv1_Input_2  ;set 2nd crisp input
8 → call           myproj       ;call preassembler code
   movf            invalidflags,W
   btfss           Z            ;test if the project is completely defined
   goto            case_no_fire

case_fire
   ;proj OK
9 → movf           lv2_Output,W  ;fetch crisp output
   ;user code
   goto            main_loop

case_no_fire
   ;no rule defined for this input combination
   ;call default_handling_routine
   ;user code
   goto            main_loop
5 → INCLUDE "myproj.asm"      ;include preassembler code
;-----
;- RESET VECTOR
;-----
   ORG             RESET_ADR
   goto            mymain      ;jump to program code
   END                ;end for assembler (only here)

```

Note: Refer to the "Integration" section for the number descriptions.

AN600

NOTES:

Low Power Design Using PIC16/17

Author: *Rodger Richey*
Logic Products Division

INTRODUCTION

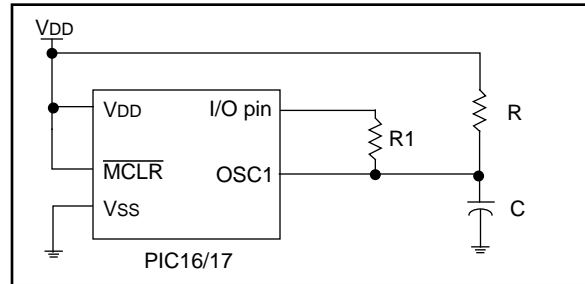
Power consumption is an important element in designing a system, particularly in today's battery powered world. The PIC16/17 family of devices has been designed to give the user a low-cost, low-power, and high performance solution to this problem. For the application to operate at the lowest possible power, the designer must ensure that the PIC16/17 devices are properly configured. This application note describes some design techniques to lower current consumption, some battery design considerations, and suggestions to assist the designer in resolving power consumption problems.

DESIGN TECHNIQUES

Many techniques are used to reduce power consumption in the PIC16/17 devices. The most commonly used methods are SLEEP Mode or external events. These modes are the best way to reduce I_{pd} in a system. The PIC16/17 device can periodically wake-up from Sleep using the Watchdog Timer or external interrupt, execute code and then go back into SLEEP Mode. In SLEEP Mode the oscillator is shut off, which causes the PIC16/17 device to consume very little current. Typical I_{pd} current in most PIC16/17 devices is on the order of a few microamps.

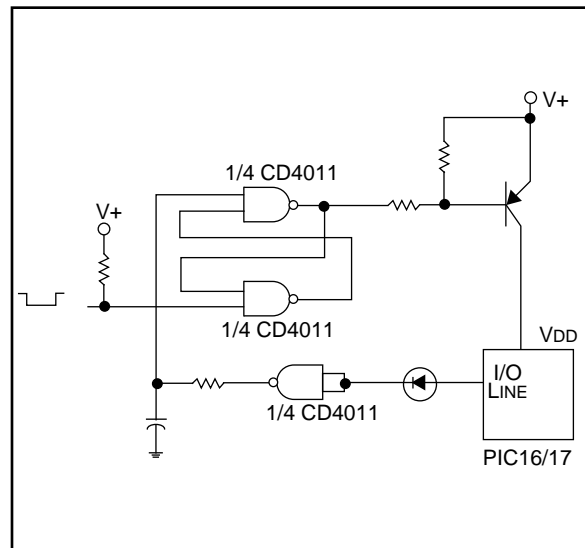
In cases where the PIC16/17 uses an RC oscillator but cannot use SLEEP Mode, another technique is used to lower power consumption. An I/O pin can remove a parallel resistance from the oscillator resistor while waiting for an event to occur. This would slow down the internal clock frequency, by increasing the resistance, and thus reduce I_{pd} . Once an event occurs the resistor can be switched in and the PIC16/17 device can process the event at full speed. Figure 1 shows how to implement this technique. The resistor R1 would be used to increase the clock frequency by making the I/O pin an output and setting it to VDD.

FIGURE 1: USING AN EXTERNAL RESISTOR TO LOWER POWER IN RC MODE



External events can be used to control the power to PIC16/17 devices. For these cases, the Watchdog Timer can be disabled to further reduce current consumption. Figure 2 shows an example circuit that uses an external event to latch power on for the PIC16/17 device. Once the device has finished executing code, it disables power by resetting the latch. The latching circuit uses a low-power 4000 series CMOS quad chip which consumes a typical of $10\mu A$ of current. The measured value of current consumption for the complete circuit with the PIC16/17 powered-down was 1 nA. Current consumption for a PIC16/17 in SLEEP Mode is typically $1\mu A$.

FIGURE 2: EXTERNAL EVENT POWER CONTROL CIRCUIT



AN606

Power consumption is dependent on the oscillator frequency of the system. The device must operate fast enough to interface with external circuitry, yet slow enough to conserve power. The designer must account for oscillator start-up time, external circuitry initialization, and code execution time when calculating device power consumption. Table 1 shows various frequency oscillators, oscillator modes and the average current consumption of each mode. A PIC16C54 was used to collect data for Table 1 and the code is shown in Example 1. A current profile for a PIC16C54 in RC oscillator mode running at 261 kHz is shown in Figure 3. Figure 4 shows a current profile for a

PIC16C54 in XT mode running at 1 MHz. The current profile includes three regions: power-up, active, and sleep. The power-up region is defined as the time the PIC16/17 device is in Power-On Reset and/or Oscillator Start-up Time. The active region is the time that the PIC16/17 device is executing code and the sleep region is the time the device is in SLEEP Mode. When using a 32.768 kHz crystal in LP oscillator mode, the designer must check that the oscillator has stabilized during the Power-On Reset. Otherwise, the device may not come out of reset properly.

TABLE 1: OSCILLATOR MODES

Osc. Type	Frequency	Osc. Mode	Power-up Region Current, Time	Active Region Current, Time	Sleep Region Current, Time
Resistor / Capacitor	261 kHz	RC	51.2 μ A, 17.5 ms	396 μ A, 12.8 ms	0.32 μ A, 140 ms
Resistor / Capacitor	1.13 MHz	RC	61.4 μ A, 17.5 ms	810 μ A, 2.5 ms	0.3 μ A, 140 ms
Crystal	32.768 kHz	LP	51.2 μ A, 19 ms	23.5 μ A, 93 ms	0.3 μ A, 140 ms
Crystal	50 kHz	LP	61.4 μ A, 16 ms	39.4 μ A, 48.5 ms	0.28 μ A, 140 ms
Crystal	1 MHz	XT	92 μ A, 17.5 ms	443 μ A, 3 ms	0.35 μ A, 140 ms
Crystal	8 MHz	HS	123 μ A, 18 ms	2.11 mA, 250 μ s	0.3 μ A, 140 ms
Resonator	455 kHz	XT	38.4 μ A, 17.3 ms	421 μ A, 7 ms	0.34 μ A, 140 ms
Resonator	8 MHz	HS	143 μ A, 18 ms	2.5 mA, 250 μ s	0.29 μ A, 140 ms

EXAMPLE 1: CURRENT PROFILE CODE

```

TITLE "Current Profiling Program"
LIST P=16C54, F=INHX8M
INCLUDE "C:\PICMASTR\P16C5X.INC"
;*****
;*****
;;      This program initializes the PIC16C54, delays for 256 counts, then goes
;      to sleep. The WDT wakes up the PIC16C54.
;*****
;*****
;Define General Purpose register locations
      LSB      EQU 0x10      ;delay control register
      Reset Vector
      ORG 0
START
      MOVLW    0x0B          ;WDT Prescaler of 1:8
      OPTION
      CLRF     PORTA         ;clear PORTA
      CLRF     PORTB         ;clear PORTB
      CLRW
      TRIS     PORTA
      TRIS     PORTB
      CLRF     LSB
LOOP   DECFSZ  LSB,1
      GOTO     LOOP
      SLEEP
      END

```

FIGURE 3: CURRENT PROFILE (261 kHz RC MODE)

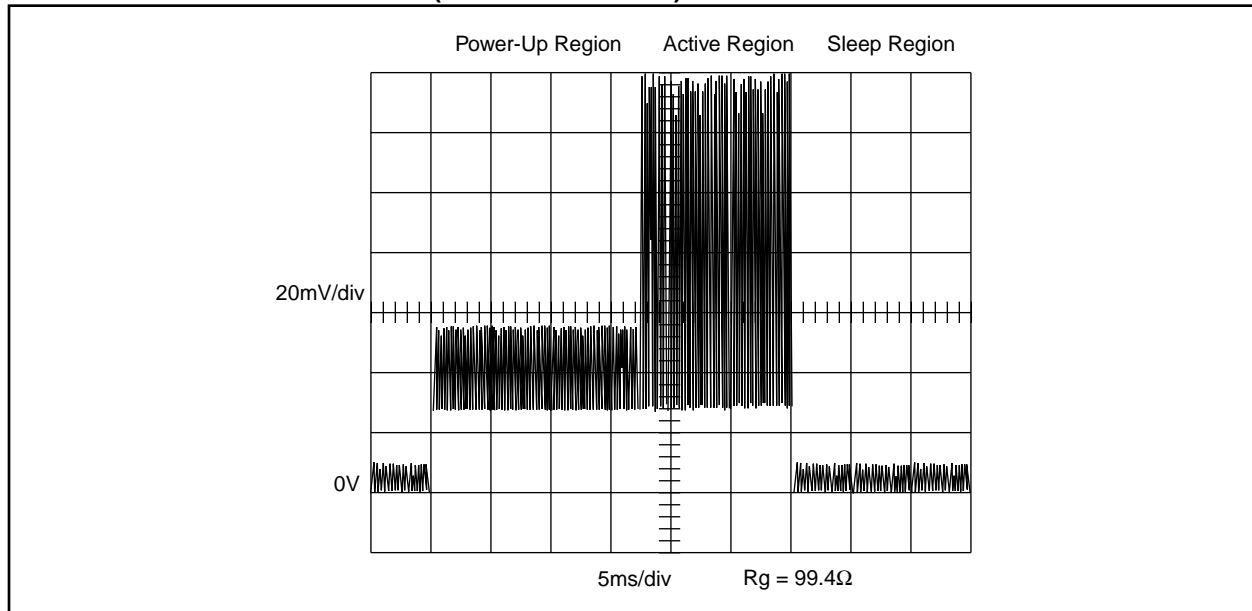
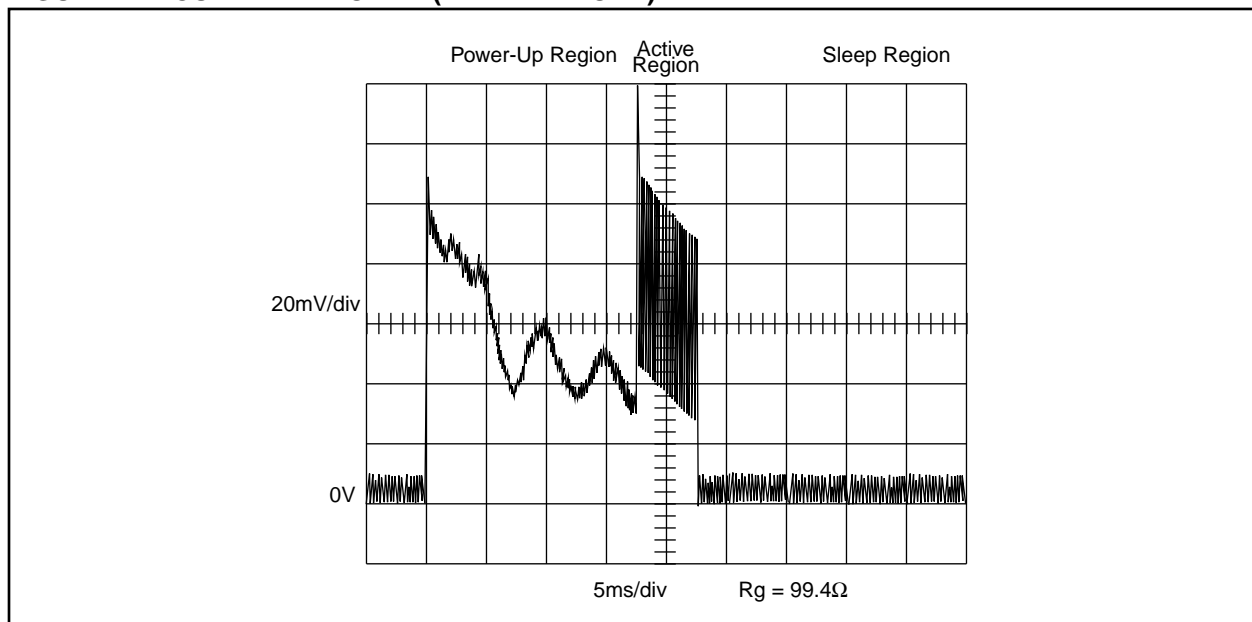


FIGURE 4: CURRENT PROFILE (1 MHz XT MODE)



Designing a system for lower supply voltages, typically 3V, is another method to reduce I_{PD}. This type of design is best utilized in a battery powered system where current consumption is very low. A wide range of devices from op-amps and Analog-to-Digital (A/D) converters to CMOS logic products are being manufactured for low voltage operation. This gives the designer the flexibility to design a low voltage system with the same type of components that are available for a 5V design. Refer to the PIC16/17 device data sheets for I_{PD} vs. V_{DD} data.

Since any I/O pin can source or sink up to 20 mA, the PIC16/17 devices can provide power to other components. Simply connect the V_{DD} pin of an external

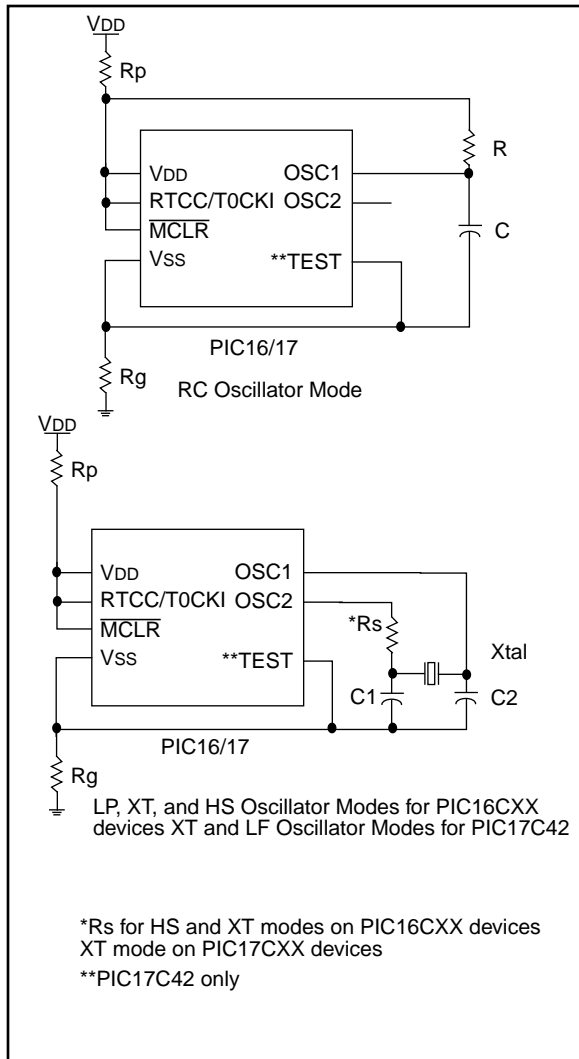
component to an I/O pin. Currently, most of the op-amps, A/D converters, and other devices manufactured today are low-power and can be powered by this technique. This provides the ability to turn off power to sections of the system during periods of inactivity.

Temperature will effect the current consumption of the PIC16/17 devices in different ways. Typically devices will consume more current at extreme temperatures and batteries will have less available current at those same temperatures. PIC16/17 devices will exhibit higher I_{pd} currents at high temperatures. Refer to the PIC16/17 device data sheets for I_{PD} vs. Temperature data.

TROUBLESHOOTING IPD

The first step in troubleshooting IPD problems is to measure the IPD that the circuit is consuming. Circuits to measure IPD for all oscillator modes are shown in Figure 5 for PIC16/17 devices. The resistor R_p is used to measure the amount of current entering the VDD pin when resistor R_g is shorted. The resistor R_g is used to measure the amount of current leaving the VSS pin when resistor R_p is shorted. The value of R_p and R_g should be approximately 100Ω for all oscillator modes. The two values of current should be approximately the same when the PIC16/17 is operating at the lowest possible power. If you find that the values of IPD measured from both configurations are not equivalent or are higher than the specifications, the following suggestions should help to find the source of extra current.

FIGURE 5: CIRCUITS TO MEASURE IPD FOR PIC16/17 DEVICES



Basically, if I_p is not equal to I_g , then an I/O pin is either sourcing ($I_p > I_g$) current or sinking ($I_p < I_g$) current.

- Is the \overline{MCLR} pin tied to VDD? Is the rate of rise of VDD slower than 0.05 V/ms? Does VDD start at VSS then rise? These conditions will not guarantee that the chip will come out of reset and function properly. Some of the circuits on PIC16/17 devices will start operating at lower voltage levels than other circuits. See Application Note AN522 "Power-Up Considerations" in the Microchip *Embedded Control Handbook*.
- Are all inputs being driven to VSS or VDD? If any input is not driven to either VSS or VDD, it will cause switching currents in the digital (i.e., flashing) input buffers. The exceptions are the oscillator pins and any pin configured as an analog input. During Power-on Reset or Oscillator Start-up time, pins that are floating may cause increased current consumption.
- All unused I/O pins should be configured as outputs and set high or low. This ensures that switching currents will not occur due to a floating input.
- Is the TMR0 (RTCC) pin pulled to VSS or VDD? The TMR0 pin of PIC16C5X devices should be tied to VSS or VDD for the lowest possible current consumption.
- If an analog voltage is present at a pin, is that pin configured as an analog input? If an analog voltage is present at a pin configured as a digital input, the digital input buffers devices will consume more current due to switching currents.
- Are all on-chip peripherals turned off? Any on-chip peripheral that can operate with an external clock source, such as the A/D converter or asynchronous timers, will consume extra current.
- Are you using the PORTB internal pull-up resistors? If so and if any PORTB I/O pin is driving or receiving a zero, the additional current from these resistors must be considered in the overall current consumption.
- Is the Power-Up Timer being used? This will add additional current drain during power-up.
- If the currents measured at the R_p and R_g resistors are not the same, then current is being sourced or sunk by an I/O pin. Make sure that all I/O pins that are driving external circuitry are switched to a low power state. For instance, an I/O pin that is driving an LED should be switched to a state where the LED is off.
- Is the window of a JW package device covered? Light will affect the current consumption of a JW package device with the window left uncovered.

IPD Analysis Using A Random Sample

The Microchip *1994 Microchip Data Book* specifies the typical I_{pd} current for a PIC16C5X part at 4 μ A and the maximum I_{pd} current at 12 μ A. These values are valid at a V_{DD} voltage of 3V and a temperature range of 0°C to 70°C with the Watchdog Timer enabled. A control group of fifty PIC16C54's were randomly selected with pre-production and production samples. I_{pd} tests were run on the group for a voltage range of 2.5V to 6.5V and for a temperature range of 0°C to 70°C. Table 2 compares the median and maximum values obtained by the I_{pd} tests to the typical and maximum values in the data book. The I_{pd} test data and the data book values are based on V_{DD} = 3.0V, Watchdog Timer Enabled, and a temperature range of 0°C to 70°C.

The values in the data book are obtained from devices in which the manufacturing process has been skewed to various extremes. This should produce devices which function close to the minimum and maximum operating ranges for each parameter shown in the data book. The typical values obtained in the data book are actually the mean value of characterization data at a temperature of 25°C. The minimum and maximum values shown in the data book are the mean value of the characterization data at the worst case temperature, plus or minus three times the standard deviation. Statistically this means that 99.5% of all devices will operate at or below the typical value and much less than the maximum value.

TABLE 2: I_{PD} COMPARISON OF CONTROL GROUP vs. DATA BOOK VALUES

Source	Typical or Median I _{PD}	Maximum
Control Group	2.349 μ A	3.048 μ A
1994 Microchip Data Book	4 μ A	12 μ A

BATTERY DESIGN

When designing a system to use batteries, the designer must consider the maximum current consumption, operating voltage range, size and weight constraints, operating temperature range, and the frequency of operation. Once the system design is finished, the designer must again ask some questions that will define what type of battery to use. What is the operating voltage range? What is the current drain rate? What are the size constraints? How long will the system be used? What type of battery costs can be tolerated? What range of temperatures will the system be operated?

It is difficult to state a rule of thumb for selecting batteries because there are many variables to consider. For example, operating voltages vary from one battery type to another. Lithium cells typically provide 3.0V while Nickel-Cadmium cells provide 1.2V. On the other hand, Lithium cells can withstand minimal discharge rates while Nickel-Cadmium can provide up to 30A of current. A designer must consider all characteristics of each battery type when making a selection. Appendix B contains a simple explanation of batteries, a characteristic table for some common battery types, and discharge curves for the common batteries.

It is very important when doing a low power design to correctly estimate the required capacity of the power source. At this point, the designer should be able to estimate the operating voltage, current drain rates and how long the system is supposed to operate. To explain how to estimate the required capacity of a system, we will use the first entry from Table 1 using an RC oscillator set at 261 kHz. Figure 3 shows the current profile for this entry. It can be seen that the profile has a period of 170.3 ms with a 17.5 ms power-up region, a 12.8 ms active region, and a 140 ms sleep region. Assuming that the system will be required to operate for six months, we can now calculate the capacity required to power this system. Example 2 will illustrate the procedure. If a system does not have a periodic current profile, then the percentages obtained in step 1 of Example 2 will have to be estimated.

EXAMPLE 2: CAPACITY CALCULATION

1. Calculate the percentage of time spent in power-up, active, and sleep regions.
power-up
 $(17.5 \text{ ms} / 170.3 \text{ ms}) \times 100 = 10.3\%$
active
 $(12.8 \text{ ms} / 170.3 \text{ ms}) \times 100 = 7.5\%$
sleep
 $(140 \text{ ms} / 170.3 \text{ ms}) \times 100 = 82.2\%$
2. Calculate the number of hours in 6 months.
6 months
 $\times (30 \text{ days} / \text{month})$
 $\times (24 \text{ hours} / \text{day}) = 4320 \text{ hours}$
3. Using the number of hours, percentages, and currents calculate the capacity for each period of time
power-up
 $4320 \text{ hours} \times 10.3\% \times 51.2 \mu\text{A} = 22.8 \text{ mAh}$
active
 $4320 \text{ hours} \times 7.5\% \times 396 \mu\text{A} = 128.3 \text{ mAh}$
sleep
 $4320 \text{ hours} \times 82.2\% \times 0.32 \mu\text{A} = 1.14 \text{ mAh}$
4. Sum the capacities of each period
 $22.8 \text{ mAh} + 128.3 \text{ mAh} + 1.14 \text{ mAh} = 152.2 \text{ mAh}$

The capacity required to operate the circuit for six months is 152.2 mAh. Example 2 does not take into consideration temperature effects or leakage currents that are associated with batteries. The load resistance of a battery is affected by temperature which in turn changes the available voltage and current; however, the self discharge rate is higher.

EXAMPLE DESIGN

A PIC16C54 with an LP oscillator of 32.768 kHz is used in this design. A Linear Technology low-power 12-bit A/D converter samples a temperature sensor. This data is transmitted via an LED at 300 baud to a receiver. The A/D converter, op-amp, and temperature sensor are powered from an I/O pin on the PIC16C54. The Watchdog Timer is enabled to periodically wake the system up from Sleep and take a sample. Figure 6 shows the schematic for the example design and Appendix A contains the source code.

This circuit has two operating modes, active and sleep. There was not a distinct power-up region in this design. In the circuit with the peripheral chips powered directly from the battery, the example design consumed 8mA of current in the active mode and 6.5 mA in SLEEP Mode. With the peripheral chips powered from an I/O pin of the PIC16C54, the example design consumed 4 mA of current in the active mode and 0.5 μA in SLEEP Mode. The advantage of using an I/O pin to provide power to

peripherals can be seen in a calculation of the capacity required to operate the circuit for one month. Example 3 details the two capacity calculations.

EXAMPLE 3: CAPACITY CALCULATION FOR THE EXAMPLE DESIGN

1. Calculate the percentage of time spent in the active and SLEEP Modes.
active - battery power
 $(210 \text{ ms} / 2.61 \text{ s}) \times 100 = 8\%$
sleep - battery power
 $(2.4 \text{ s} / 2.61 \text{ s}) \times 100 = 92\%$
active - I/O power
 $(188 \text{ ms} / 2.638 \text{ s}) \times 100 = 7.1\%$
sleep - I/O power
 $(2.45 \text{ s} / 2.638 \text{ s}) \times 100 = 92.9\%$
2. Calculate the number of hours in 1 month.
1 month
 $\times (30 \text{ days} / \text{month})$
 $\times (24 \text{ hours} / \text{day})$
 $= 720 \text{ hours}$
3. Using the number of hours, percentages and currents calculate the capacity for each period of time.
active - battery power
 $720 \text{ hours} \times 8\% \times 8 \text{ mA} = 461 \text{ mAh}$
sleep - battery power
 $720 \text{ hours} \times 92\% \times 6.5 \text{ mA} = 4306 \text{ mAh}$
active - I/O power
 $720 \text{ hours} \times 7.1\% \times 4 \text{ mA} = 205 \text{ mAh}$
sleep - I/O power
 $720 \text{ hours} \times 92.9\% \times 0.5 \mu\text{A} = 0.4 \text{ mAh}$
4. Sum the capacities of each period.
battery power
 $461 \text{ mAh} + 4306 \text{ mAh} = 4767 \text{ mAh}$
I/O power
 $205 \text{ mAh} + 0.4 \text{ mAh} = 206 \text{ mAh}$

The capacity required to operate this circuit for one month can be reduced by a factor of twenty just by powering the peripheral components from an I/O pin. The example design will use two Panasonic® BR2325 Lithium batteries in series to provide power to the circuit. This results in a V_{batt} of 6V and a capacity of 165 mAh. Using the estimation process, the circuit should function for approximately 24 days. The actual time of operation was 24.2 days with the system running in an ambient temperature of 22°C.

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: EXAMPLE DESIGN CODE

```

MPASM 01.02.05 Released      LOWPWR.ASM      1-9-1995      13:2:42      PAGE 1
Ipd/Battery Apnote Example Design
LOC OBJECT CODE      LINE SOURCE TEXT
VALUE

0001      TITLE "Ipd/Battery Apnote Example Design"
0002      LIST P=16C54, F=INHX8M
0003
0004      INCLUDE "C:\PICMASTR\PI6C5X.INC"
0002 ; PI6C5X.INC Standard Header File, Version 0.1      Microchip Technology, Inc.
0004
0005
0006 ;*****
0007 ;*****
0008 ;
0009 ;      Filename:      lowpwr.asm
0010 ;      REVISION:      9 Jan 95
0011 ;
0012 ;*****
0013 ;
0014 ;      This program initializes the PIC, takes a sample, and outputs the
0015 ;      value to PORTB pin 0 (the LED), and then goes to Sleep. The
0016 ;      Watchdog Timer wakes the device up from Sleep. PORTA pin 0 is used
0017 ;      to control power to peripherals.
0018 ;
0019 ;*****
0020 ;*****
0021
0022 ;      Define variable registers
0023      MSB      EQU      0x10
0024      LSB      EQU      0x11
0025      DELAY_CNT      EQU      0x12
0026      SHIFT      EQU      0x13
0027      COUNT      EQU      0x14
0028
0029 ;      Reset Vector
0030      ORG      0x1FFF
0031      GOTO      START
0032
0033 ;      Start of main code
0010      0010
0011      0011
0012      0012
0013      0013
0014      0014
01FF      0A00

```



```

0000
0000 0C2F
0001 0002
0002 0C02
0003 0025
0004 0066
0005 0C08
0006 0005
0007 0040
0008 0006
0009 0071
000A 0070

000B 0004
000C 0911
000D 0004
000E 0948
000F 0004
0010 0003

0011
0011 0505
0012 0943
0013 0C0B
0014 0034
0015 0C08
0016 0033
0017 0000
0018 0425
0019 0000
001A 0545
001B 0000
001C 0445
001D 0000
001E 0545
001F 0000
0020 0445
0021 0000

0034 0000
0035
0036 ;*****
0037 ; Main routine which initializes the PIC, and has main loop.
0038 ;*****
0039 START
0040 MOVW 0x2F ;1:128 WDT PRESCALAR
0041 OPTION ;RA1 SET HIGH
0042 MOVW 0x02 ;ALL PINS SET TO Vss
0043 MOVWF PORTA ;RA3-DATA INPUT
0044 CLRF PORTB ;RA0-POWER, RA1-CS, RA2-CLOCK OUTPUTS
0045 MOVW 0x08 ;PORTB ALL OUTPUTS, RBO-LED OUTPUT
0046 TRIS PORTA ;CLEAR A/D RESULT REGISTERS
0047 CLRW ;
0048 TRIS PORTB ;
0049 CLRF LSB ;
0050 CLRF MSB ;
0051
0052 CLRWDI
0053 CALL SAMPLE ;GET SAMPLE FROM A/D
0054 CLRWDI
0055 CALL OUTPUT ;OUTPUT SAMPLE TO LED AT 300 BAUD
0056 CLRWDI
0057 SLEEP
0058
0059
0060 ;*****
0061 ; Main routine for retrieving a sample from the A/D.
0062 ;*****
0063 SAMPLE
0064 BSF PORTA,0 ;TURN POWER ON TO PERIPHERALS
0065 CALL DELAY ;WAIT FOR PERIPHERALS TO STABILIZE
0066 MOVW 0x0B ;DATA COUNTER, 12 BIT A/D
0067 MOVWF COUNT
0068 MOVW 0x08 ;SET SHIFT REGISTER
0069 MOVWF SHIFT
0070 NOP
0071 BCF PORTA,1 ;ENABLE A/D
0072 NOP
0073 BSF PORTA,2 ;1ST CLOCK RISE
0074 NOP
0075 BCF PORTA,2 ;1ST CLOCK FALL
0076 NOP
0077 BSF PORTA,2 ;NULL BIT CLOCK RISE
0078 NOP
0079 BCF PORTA,2 ;NULL BIT CLOCK FALL
0080 NOP

```

```

0022 0933          CALL          ;READ DATA BIT
0023 0000          NOP
0024 0545          BSF          PORTA, 2      ;BIT CLOCK RISE
0025 0000          NOP
0026 0445          BCF          PORTA, 2      ;BIT CLOCK FALL
0027 0000          NOP
0028 02F4          DECFSZ      COUNT, F      ;CHECK LOOP COUNTER
0029 0A22          GOTO        LOOP
002A 0933          CALL          ;READ LAST BIT
002B 0000          NOP
002C 0545          BSF          PORTA, 2      ;SET CLOCK
002D 0000          NOP
002E 0525          BSF          PORTA, 1      ;SET CS
002F 0000          NOP
0030 0445          BCF          PORTA, 2      ;CLEAR CLOCK
0031 0405          BCF          PORTA, 0      ;POWER DOWN PERIPHERALS
0032 0800          RETURN

0033
0033 0004          CLRWDI
0034 0774          BTFSS      COUNT, 3      ;CHECK IF AT BIT 8 - 11
0035 0A3B          GOTO        RLOW         ;GOTO BITS 0 - 7
0036 0765          BTFSS      PORTA, 3      ;CHECK IF DATA IS CLEAR
0037 0A3F          GOTO        REND         ;GOTO EXIT
0038 0213          MOVF        SHIFT, W      ;ADD A ONE TO MSB IN THE CORRECT
0039 01F0          ADDWF      MSB, F        ;BIT POSITION
003A 0A3F          GOTO        REND
003B 0765          BTFSS      PORTA, 3      ;ADD A ONE TO LSB IN THE CORRECT
003C 0A3F          GOTO        REND         ;BIT POSITION
003D 0213          MOVF        SHIFT, W      ;SHIFT
003E 01F1          ADDWF      LSB, F        ;SHIFT
003F 0333          RRF          SHIFT, F      ;IF ONE IS IN THE CARRY
0040 0603          BTFSC      STATUS, C      ;SHIFT AGAIN
0041 0333          RRF          SHIFT, F
0042 0800          RETURN

0043
0043 0004          CLRWDI          ;RESET WATCHDOG TIMER
0044 0072          CLR         DELAY_CNT
0045 02F2          DECFSZ      DELAY_CNT, F

0081
0082 LOOP          CALL          ;READ DATA BIT
0083          NOP
0084          BSF          PORTA, 2      ;BIT CLOCK RISE
0085          NOP
0086          BCF          PORTA, 2      ;BIT CLOCK FALL
0087          NOP
0088          DECFSZ      COUNT, F      ;CHECK LOOP COUNTER
0089          GOTO        LOOP
0090          CALL          ;READ LAST BIT
0091          NOP
0092          BSF          PORTA, 2      ;SET CLOCK
0093          NOP
0094          BSF          PORTA, 1      ;SET CS
0095          NOP
0096          BCF          PORTA, 2      ;CLEAR CLOCK
0097          BCF          PORTA, 0      ;POWER DOWN PERIPHERALS
0098          RETURN
0099

0100 ;*****
0101 ; Reads a bit from PORTA, data line from the A/D.
0102 ;*****
0103 READ
0104          CLRWDI
0105          BTFSS      COUNT, 3      ;CHECK IF AT BIT 8 - 11
0106          GOTO        RLOW         ;GOTO BITS 0 - 7
0107          BTFSS      PORTA, 3      ;CHECK IF DATA IS CLEAR
0108          GOTO        REND         ;GOTO EXIT
0109          MOVF        SHIFT, W      ;ADD A ONE TO MSB IN THE CORRECT
0110          ADDWF      MSB, F        ;BIT POSITION
0111          GOTO        REND
0112          BTFSS      PORTA, 3      ;ADD A ONE TO LSB IN THE CORRECT
0113          GOTO        REND         ;BIT POSITION
0114          MOVF        SHIFT, W      ;SHIFT
0115          ADDWF      LSB, F        ;SHIFT
0116          RRF          SHIFT, F      ;IF ONE IS IN THE CARRY
0117          BTFSC      STATUS, C      ;SHIFT AGAIN
0118          RRF          SHIFT, F
0119          RETURN
0120

0121 ;*****
0122 ; Simple delay loop for 772 clock cycles.
0123 ;*****
0124 DELAY
0125          CLRWDI          ;RESET WATCHDOG TIMER
0126          CLR         DELAY_CNT
0127          DLOOPL      DELAY_CNT, F

```

```

0046 0A45      GOTO    DLOOPL
0047 0800      RETURN
0128
0129
0130
0131
0132 ; *****
0133 ; Output sample to LED at 300 baud.
0134 ; *****
0135 OUTPUT
0136      MOVLM  0x08      ; SHIFT 8 MSB BITS OUT
0137      MOVWF  COUNT
0138
0139 MSBOUT  RLF      MSB,F      ; SHIFT LSB INTO CARRY
0140      BTFSZ  STATUS,C      ; IF CARRY IS SET
0141      GOTO   MSBCLR
0142      BSF   PORTB,0      ; SET PORTB,0
0143      CALL  BAUD
0144      GOTO  MSBCHK      ; CHECK FOR ALL 8 BITS TO BE SENT
0145 MSBCLR  BCF   PORTB,0      ; OTHERWISE CLEAR PORTB,0
0146      NOP
0147      NOP
0148      CALL  BAUD
0149 MSBCHK  DECFSZ COUNT      ; CHECK FOR ALL 8 BITS TO BE SENT
0150      GOTO  MSBOUT
0151
0152      MOVLM  0x08      ; SHIFT 8 LSB BITS OUT
0153      MOVWF  COUNT
0154
0155 LSBOUT  RLF      LSB,F      ; SHIFT LSB INTO CARRY
0156      BTFSZ  STATUS,C      ; IF CARRY IS SET
0157      GOTO   LSBCLR
0158      BSF   PORTB,0      ; SET PORTB,0
0159      CALL  BAUD
0160      GOTO  LSBCHK      ; CHECK FOR 8 BITS TO BE SENT
0161 LSBCLR  BCF   PORTB,0      ; OTHERWISE CLEAR PORTB,0
0162      NOP
0163      NOP
0164      CALL  BAUD
0165 LSBCHK  DECFSZ COUNT      ; CHECK FOR 8 BITS TO BE SENT
0166      GOTO  LSBOUT
0167      BCF   PORTB,0      ; CLEAR PORTB,0
0168      CLRF  LSB
0169      CLRF  MSB
0170      RETURN
0171
0172 ; *****
0173 ; Delay loop for sending data to the LED at 300 baud.
0174 ; *****

```


APPENDIX B: BATTERY DESCRIPTIONS

Presently there are two types of batteries that are manufactured, primary and secondary. Primary batteries are those that must be thrown away once their energy has been expended. Low current drain, short duty cycles, and remote operation favor primary batteries such as Carbon Zinc and Alkaline. Secondary batteries can be recharged once they have exhausted their energy. High current drain or extended usage favors secondary batteries especially when the cost of replacement of disposable batteries is not feasible. Secondary batteries include Nickel-Cadmium and Nickel Metal Hydride.

A battery may be discharged by different means depending on the type of load. The type of load will have a significant effect on the life of the battery. The typical modes of discharge are constant resistance, constant current, and constant power. Constant resistance is when the load maintains a constant resistance throughout the discharge cycle. Constant current is the mode where the load draws the same current during discharge. Finally, constant power is defined as the current during a discharge increases as the battery voltage decreases.

The constant resistance mode results in the capacity of the battery being drained at a rapid and excessive rate, resulting in a short life. This is caused by the current during discharge following the drop in battery voltage. As a result, the levels of current and power during discharge are in excess of the minimum required.

The constant current mode has lower current and power throughout the discharge cycle than the constant resistance mode. The average current drain on the battery is lower and the discharge time to the end-voltage is longer.

The constant power discharge mode has the lowest average current drain and therefore has the longest life. During discharge, the current is lowest at the beginning of the cycle and increases as the battery voltage drops. Under this mode the battery can be discharged below its end voltage, because the current is increased as the voltage drops. The constant power mode provides the most uniform performance throughout the life of the battery and has the most efficient use of the energy in the battery.

The nominal voltage is the no-load voltage of the battery, the operating voltage is the battery voltage with a load, and the end-of-life voltage is the voltage when the battery has expended its energy. Energy Density is used to describe the amount of energy per unit of volume or mass (Wh/kg or Wh/l). Generally, energy density decreases with decreasing battery size within a particular type of battery. Most batteries are rated by an amp-hour (Ah) or milliamp-hour (mAh) rating. This rating is based on a unit of charge, not energy. A 1-amp current corresponds to the movement of 1 coulomb (C) of charge past a given point in 1 second (s). Table B-1 lists some typical characteristics of the most common types of batteries.

AN606

TABLE B-1: TYPICAL BATTERY CHARACTERISTICS

	Carbon Zinc	Alkaline	Nickel Cadmium	Lithium	Nickel Metal Hydride	Zinc Air	Silver Oxide
Cell Voltage							
Nominal	1.5	1.5	1.2	3.0	1.2	1.4	1.6
Operating	1.25-1.15	1.25-1.15	1.25-1.00	2.5-3.0	1.25-1.0	1.35-1.1	1.5
End of life	0.8	0.9	0.9	1.75	0.9	0.9	0.9
Operating Temperature	-5°C to 45°C	-20°C to 55°C	-40°C to 70°C	-30°C to 70°C	-20°C to 50°C	0°C to 45°C	-20°C to 50°C
Energy Density (Wh/kg)	70	85	30	300	55	300	100
Capacity	60mAh to 18Ah	30mAh to 45Ah	150mAh to 4Ah	35mAh to 4Ah	500mAh to 5Ah	50mAh to 520mAh	15mAh to 210mAh
Advantages		High capacity, good low temp	good low temp, good high rate discharge	good low and high temp, good high rate discharge, long shelf life	better capacity than Nicad for same size	high energy density, good shelf life	good low temp, good shelf life
Limitations	Low energy density, poor low temp, poor high rate discharge		poor low rate discharge, disposal hazards	Violent reaction to water		Cannot stop reaction once started	poor high rate discharge
Relative Cost	low	low	medium	high	high	high	high
Type	Primary	Primary	Secondary	Primary	Secondary	Primary	Primary

Typical discharge curves for alkaline, carbon zinc, lithium, nickel cadmium, nickel metal hydride, silver oxide, and zinc air are shown in Figure B-1 through Figure B-7. These curves are only typical representations of each battery type and are not specific to any battery manufacturer. Also the load and current drain are different for each type of battery.

FIGURE B-1: ALKALINE DISCHARGE CURVE (16 mA LOAD)

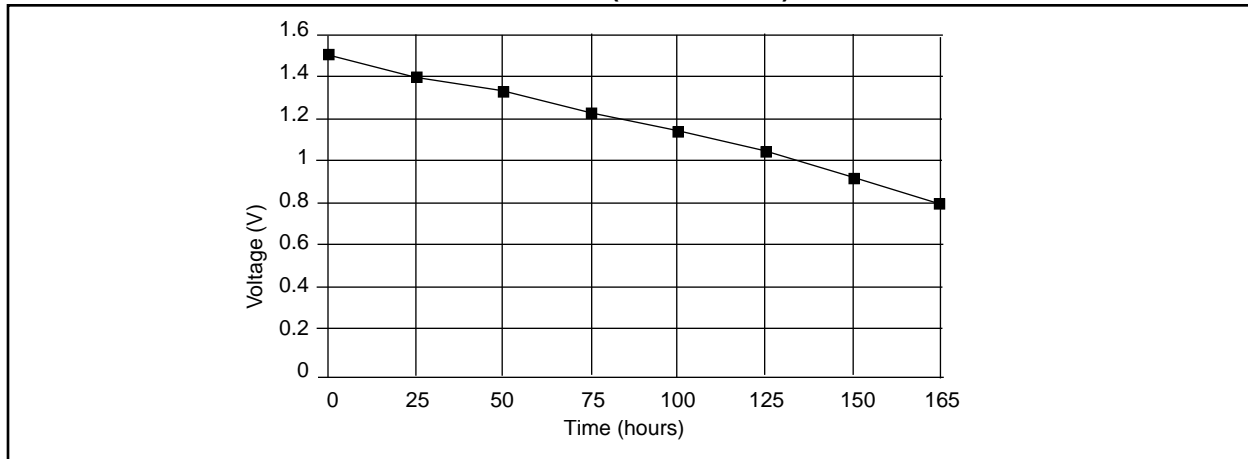


FIGURE B-2: CARBON ZINC DISCHARGE CURVE (16 mA LOAD)

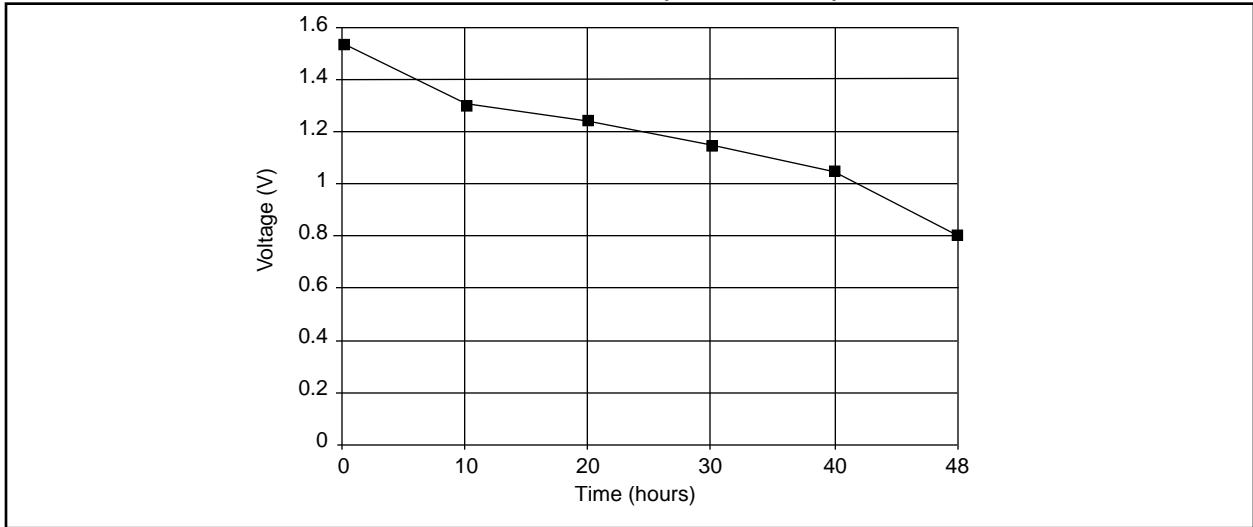


FIGURE B-3: LITHIUM DISCHARGE CURVE (2.8 mA LOAD)

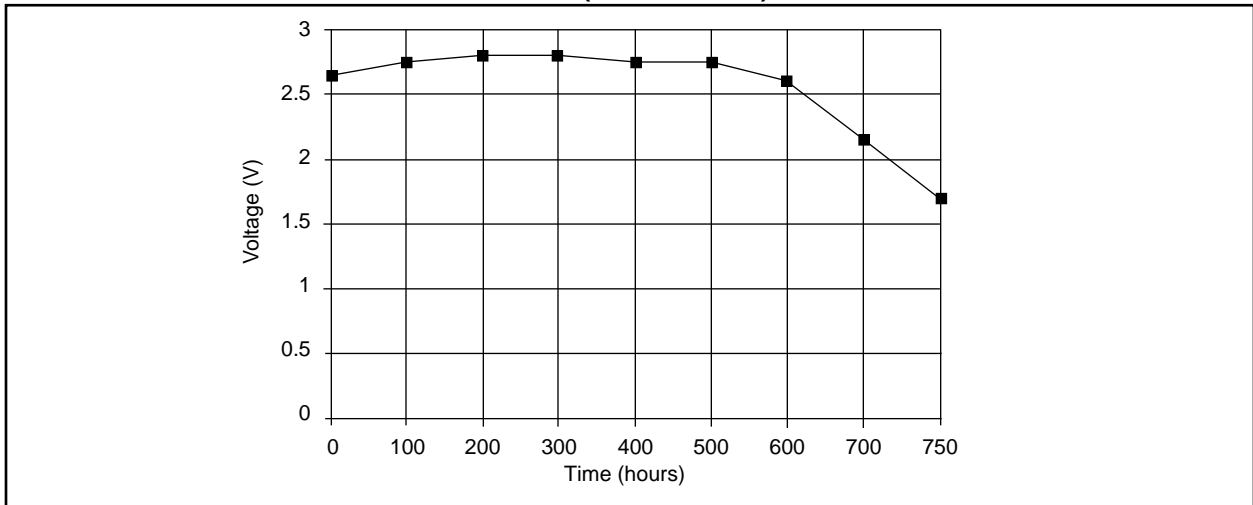
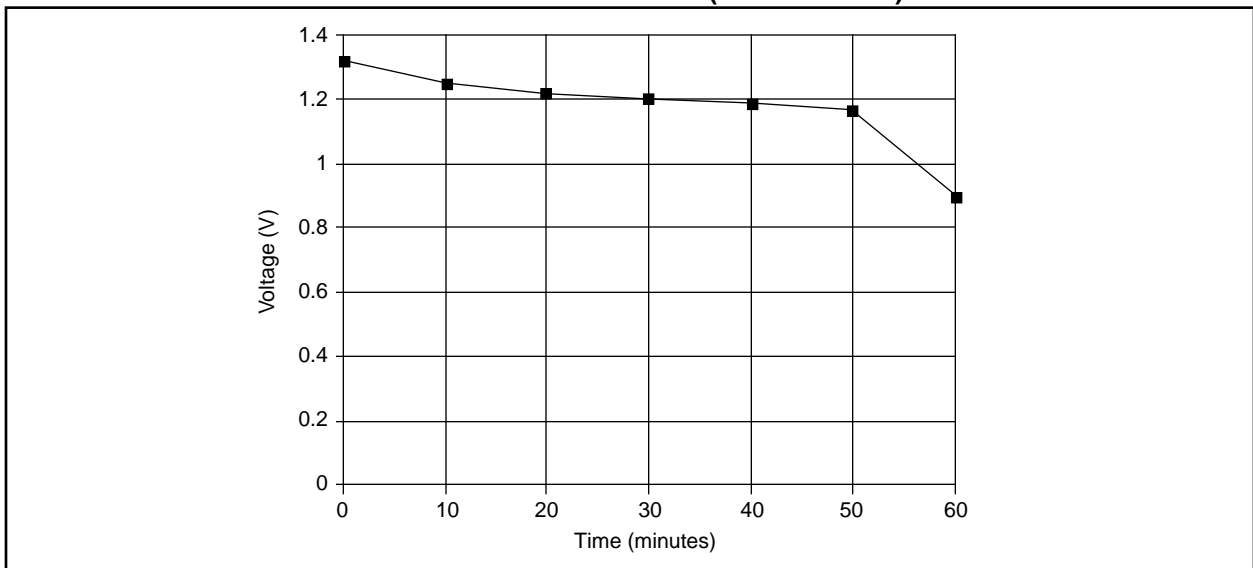


FIGURE B-4: NICKEL CADMIUM DISCHARGE CURVE (500 mA LOAD)



AN606

FIGURE B-5: NICKEL METAL HYDRIDE DISCHARGE CURVE (1500 mA LOAD)

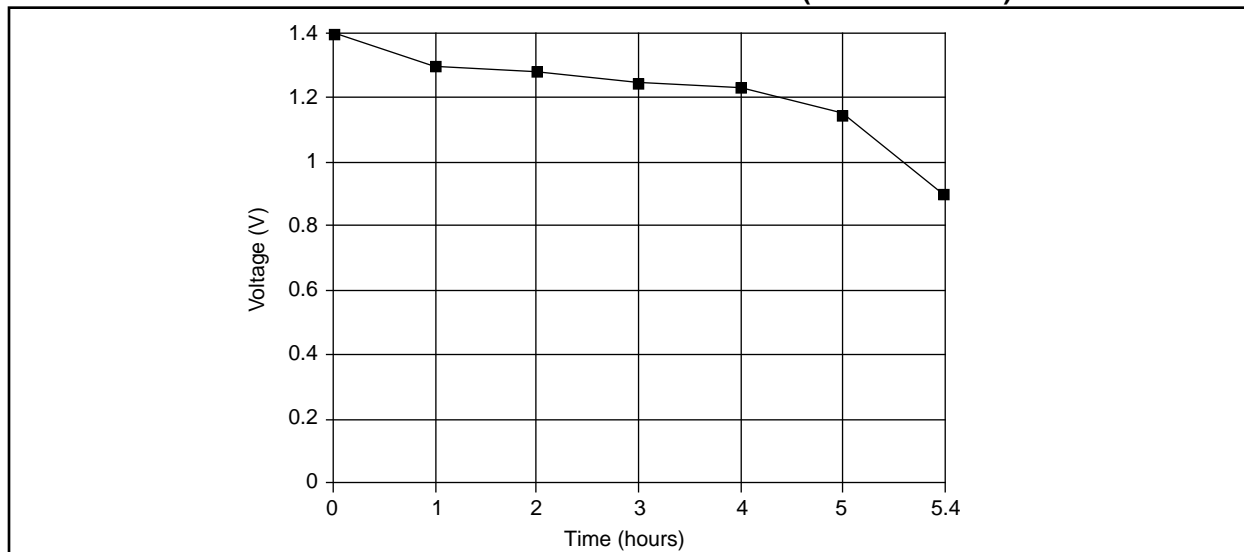


FIGURE B-6: SILVER OXIDE DISCHARGE CURVE (1 mA LOAD)

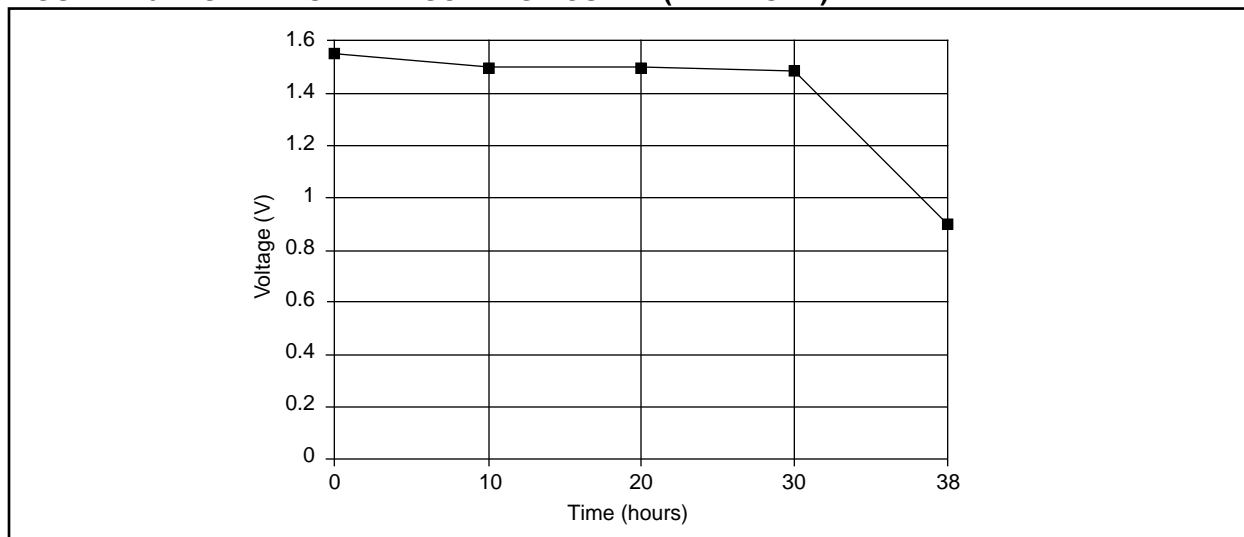
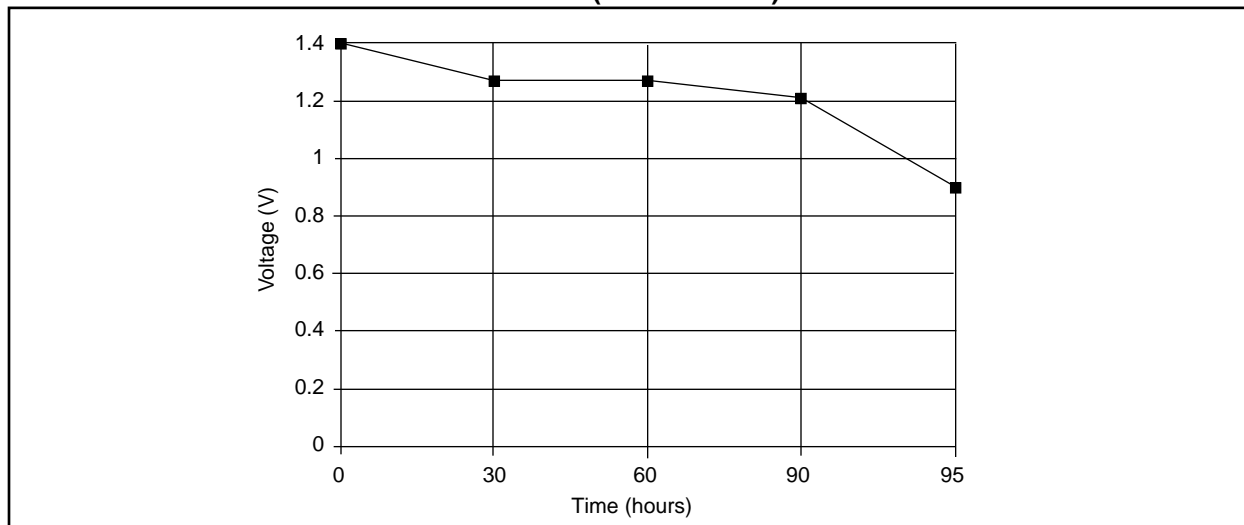


FIGURE B-7: ZINC AIR DISCHARGE CURVE (1.3 mA LOAD)



Power-up Trouble Shooting

*Author: Mark Palmer
Logic Products Division*
*Contributions: Richard Hull & Randy Yach
Logic Products Division*

INTRODUCTION

For any application to begin proper operation, the application must power-up properly. Many criteria must be taken into account to ensure this. The PIC16/17 devices integrate several features to simplify the design for the power-up sequence. These integrated features also reduce the total system cost.

This application note describes the requirements for the device to properly power-up, common pitfalls that designers encounter, and methods to assist in solving power-up problems.

THE POWER-UP SEQUENCE

There are several factors that determine the actual power-up sequence that a device will go through. These factors are:

- The Processor Family
 - PIC16C5X
 - PIC16CXX
 - PIC17CXX
- Oscillator Configuration
- Device Configuration
- MCLR pin

Note: The PIC16CXX family refers to devices with a 14-bit instruction word. This does not include the PIC16C5X family.

The Power-on Reset (POR) signal generation is discussed, followed by the power-up sequence for the specific device families.

Power-on Reset (POR) signal

The data sheets show a Power-on Reset (POR) pulse, as in Figure 1. The POR signal is a level triggered signal. This representation will help in the understanding of future devices, which may have a brown-out reset capability.

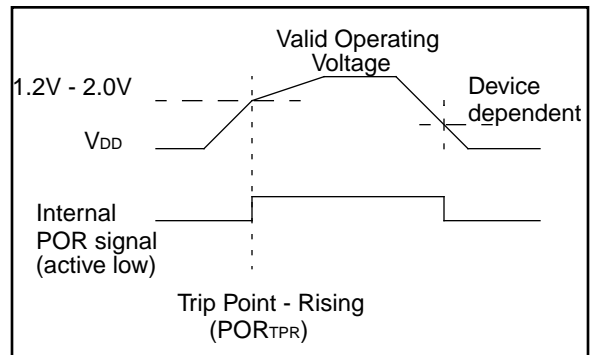
The power-up sequence begins by increasing the voltage on the V_{DD} pin (from 0V). If the slope of the V_{DD} rise time is faster than 0.05 V/ms, the internal circuitry is

capable of generating an internal reset signal. Depending on the device family, different power-up sequences will occur after this POR signal.

If the slope is less than 0.05 V/ms, the \overline{MCLR} pin should be held low, by external circuitry, until a valid operating V_{DD} level is reached.

The V_{DD} rise time specification needs to be met, until the POR signal is generated. After the POR signal is generated the slope of the V_{DD} rise can change (to a faster or slower rise). This may have other ramifications, see the "Power-up Consideration" section. In general, the POR signal will trip (POR_{TPR}) somewhere between 1.2V to 2.0V (Figure 1).

FIGURE 1: INTERNAL POR SIGNAL



When V_{DD} is falling, the voltage at which the internal POR signal returns to a low level is processor/device dependent. To ensure that a device will have a POR, the device voltage must return to V_{SS} before power is re-applied.

Note: Some devices (with EPROM program memory) have a newer POR circuit that does not require V_{DD} to return to V_{SS} . See the device data sheet for the complete specification on the POR operation.

The POR will be generated regardless of the level of the \overline{MCLR} pin. The PIC16/17 device families are different on what triggers the power-up sequence. Table 1 describes the events that cause the POR sequence to occur.

After reaching the POR trip point (POR_{TPR}), the POR sequence holds the device in reset for a given time. Once this time has elapsed, the device voltage must be valid or the \overline{MCLR} pin must be low. The time from the POR rising edge to the time that V_{DD} must be valid level is the $T_{POR2VDDV}$ time.

AN607

TABLE 1: EVENTS THAT TRIGGER POR SEQUENCE

Device	Events
PIC16C5X	Both the POR signal rising edge and any $\overline{\text{MCLR}}$ rising edge ⁽¹⁾
PIC16CXX	The POR signal rising edge
PIC17CXX	Either the POR signal rising edge or the first $\overline{\text{MCLR}}$ rising edge (if $\overline{\text{MCLR}}$ is low when the POR occurs). After this event, all following $\overline{\text{MCLR}}$ rising edges ⁽¹⁾ cause the device to start program execution immediately.

Note 1: The POR low-to-high transition causes Special Function Register (SFR) bits/registers to a specified value. The SFR bits/register are not identically affected by the $\overline{\text{MCLR}}$ signal. Refer to the device data sheet to see how the bits are affected by these two conditions.

The POR sequence for each of the PIC16/17 families is described in the following three sections:

- PIC16C5X Family
- PIC16CXX Family
- PIC17CXX Family

PIC16C5X Family

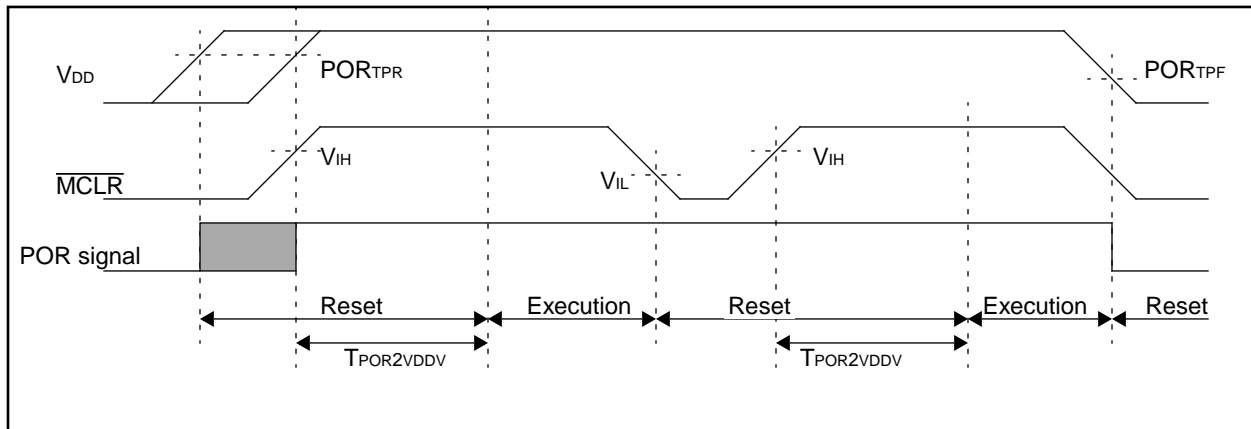
After the $\overline{\text{MCLR}}$ pin has reached a high level, the device is held in reset for typically 18 ms. This time is determined by an on-chip RC oscillator and 8-bit ripple counter. This Device Reset Timer (DRT), allows most crystals (except low frequency crystals) to start-up and stabilize. Due to the characteristics of resistors and capacitors, this time is extremely variable over temperature and voltage. There is also a device to device variation. See the data sheet for the range of this time-out.

TABLE 2: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)

Oscillator Configuration	Power-up	Wake-up from SLEEP
XT, HS, LP ⁽¹⁾	18 ms	18 ms
RC	18 ms	18 ms

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20 ms. Resonators a typically <1 ms. All these times are voltage dependent.

FIGURE 2: PIC16C5X POWER-UP SEQUENCE



PIC16CXX Family

After the POR rising edge has occurred, the device can have up to 2 time-out sequences that occur in series. The first being the Power-up Timer (PWRT), the second being the Oscillator Start-up Timer (OST).

The Power-up Timer time-out will occur if enable fuse PWRTE is read as a '1'. The PWRT uses a 10-bit counter, with the clock from an internal RC. Due to the characteristics of resistors and capacitors, this time is extremely variable over temperature and voltage. There is also a device to device variation. See the data sheet for the range of this time-out.

Note: Future devices will change the polarity of the PWRTE configuration bit. Refer to the specific data sheet for the polarity of this bit.

The OST will occur on power-up/wake-up when the device has oscillator mode selected. This allows the oscillator to stabilize before program execution begins. The OST uses a 10-bit counter, with the clock from the OSC pin. The time is dependent on the frequency of the input clock. This timer is disabled if the oscillator is configured as RC.

Figure 3 shows how the two timers work in the power-up sequence. V_{DD} must be valid when program execution starts. The $T_{PWRT} + T_{OST}$ times can be thought of as the time that the device gives for the V_{DD} to become valid ($T_{POR2VDDV}$). Figure 4 shows when device execution begins for the case of the \overline{MCLR} pin going high before $T_{POR2VDDV}$ times out. Figure 5 shows when the \overline{MCLR} pin is held low longer than the $T_{POR2VDDV}$ time. The device starts execution immediately when \overline{MCLR} goes high. Table 3 gives the typical reset times.

TABLE 3: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)

Oscillator Configuration	Power-up		Wake-up from SLEEP
	PWRTE = 1 (2)	PWRTE = 0 (2)	
XT, HS, LP ⁽¹⁾	72 ms + 1024 T_{osc}	1024 T_{osc}	1024 T_{osc}
RC	72 ms	—	—

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20ms. Resonators are typically <1 ms. All these times are voltage dependent.

Note 2: Future devices will change the polarity of this configuration bit. Refer to the specific data sheet for the polarity of the PWRT Configuration Bit.

FIGURE 3: PIC16CXX POWER-UP SEQUENCE

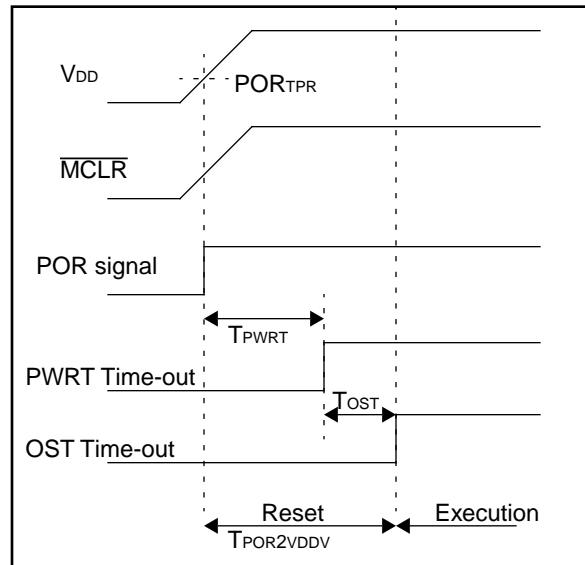


FIGURE 4: START OF DEVICE OPERATION ($\overline{MCLR} < T_{POR2VDDV}$)

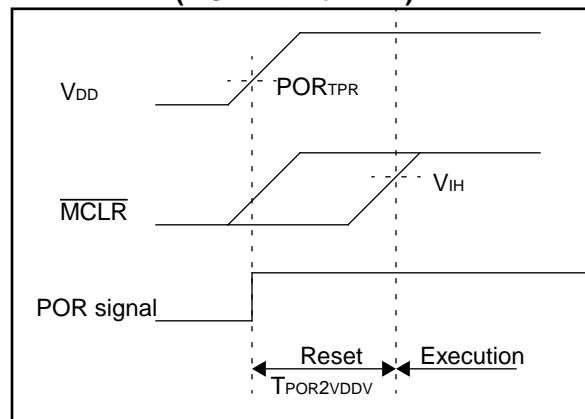
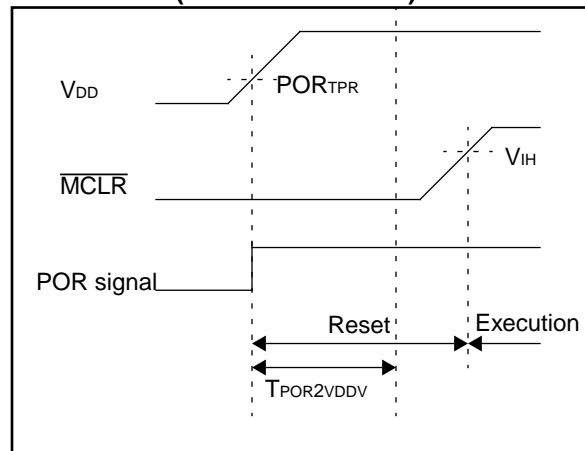


FIGURE 5: START OF DEVICE OPERATION ($\overline{MCLR} > T_{POR2VDDV}$)



AN607

PIC17CXX Family

When the $\overline{\text{MCLR}}$ pin comes to a high level, after the POR rising edge, the device has 2 time-out sequences that occur in parallel. One is the Power-up Timer (PWRT), the other is the Oscillator Start-up Timer (OST). The timer with the greater time holds the device in reset. Figure 6 shows the sequence with $\overline{\text{MCLR}}$ tied to V_{DD} . Figure 7 shows the time-out when $\overline{\text{MCLR}}$ is independent of V_{DD} . The PWRT time is generally longer, except for low frequency crystals/resonators. The OST time does not include the start-up time of the oscillator/resonator.

The PWRT uses a 10-bit counter, with the clock from an internal RC. The characteristics of the RC vary from device to device and over temperature and voltage. The specification for the time-out range can be found in the electrical specification of the data sheet.

The OST uses a 10-bit counter, with the clock from the OSC pin. The time is dependent on the frequency of the input clock.

Until $\overline{\text{MCLR}}$ has reached a high level, the POR sequence will not start. While the POR signal remains high, all following $\overline{\text{MCLR}}$ pulses will not cause the POR sequences to occur (Figure 8).

TABLE 4: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)

Oscillator Configuration	Power-up	Wake-up from Sleep
RC, EC	Greater of 80 ms and $1024 T_{osc}$	—
XT, LF ⁽¹⁾	Greater of 80 ms and $1024 T_{osc}$	$1024 T_{osc}$

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20 ms. Resonators are typically <1 ms. All these times are voltage dependent.

FIGURE 6: PIC17CXX POWER-UP SEQUENCE ($\overline{\text{MCLR}}$ TIED TO V_{DD})

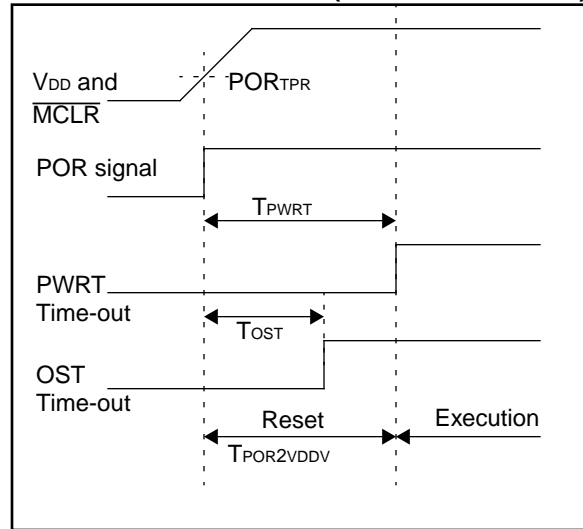


FIGURE 7: PIC17CXX POWER-UP SEQUENCE ($\overline{\text{MCLR}}$ NOT TIED TO V_{DD})

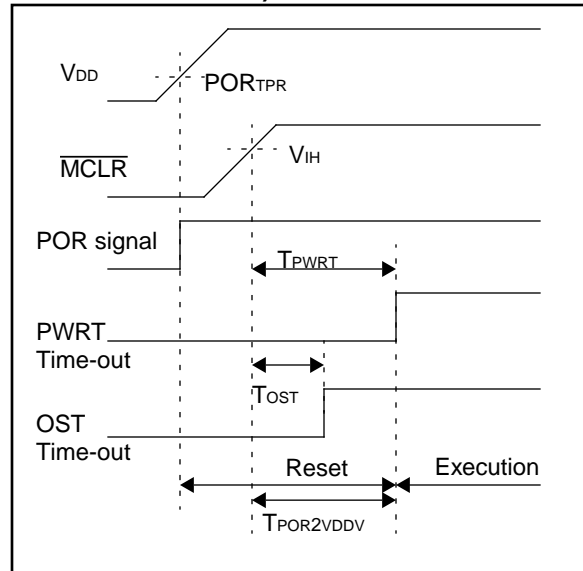
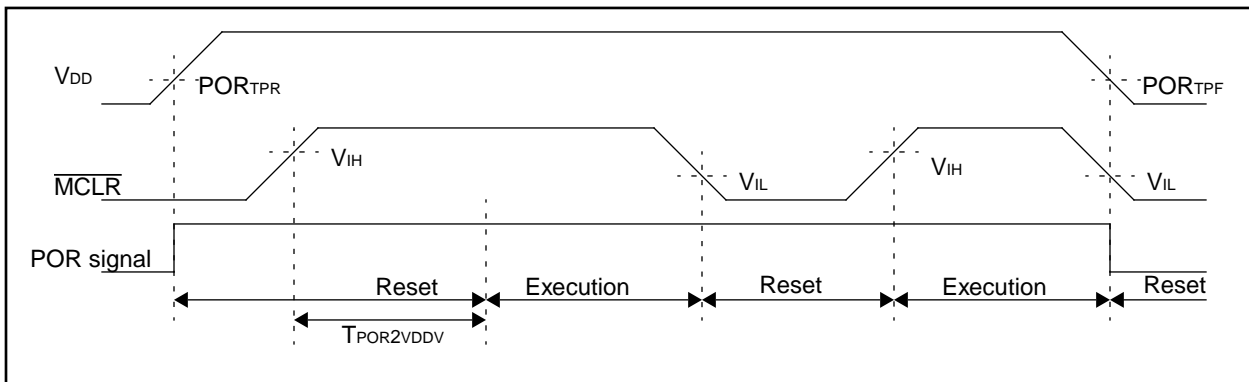


FIGURE 8: $\overline{\text{MCLR}}$ OPERATION



POWER-UP CONSIDERATIONS

The device must be at a valid operating voltage when the device exits reset. This can be done by ensuring that the power supply rise time is fast enough to guarantee an operating V_{DD} level, or by using an external reset circuit which will hold \overline{MCLR} low until the operating V_{DD} level is reached.

When the rise time of V_{DD} is very fast, there will be a time delay before the Power-on Reset (POR) signal will rise to a logic high ($T_{TP2PORH}$). This delay is in the 1-5 μ s range, as shown in Figure 9.

Figure 10, Figure 11, and Figure 12 show the maximum time from the POR sequence beginning to the device having a valid operating voltage. Table 5 gives the $T_{POR2VDDV}$ times. When determining the time at which V_{DD} must be valid, the POR trip point must be assumed to be at the minimum POR voltage trip point.

How Crystal Frequencies affect Start-up time

Both the PIC16CXX and PIC17CXX families may have start-up times that include the contributions of the oscillator. Table 5 shows how the oscillator can affect each mode of operation, with Table 6 giving the reset time that an oscillator generates. This time can be used in the equation to calculate the total reset time, at the given frequency. This time may vary slightly due to the initial start-up characteristics of the crystal/oscillator circuit.

Note 1: The rise time specification does not ensure that a valid V_{DD} operating voltage will be reached before the device exits reset. The device's V_{DD} must be within the specified operating range for proper device operation.

Note 2: The start-up characteristics of the crystal/oscillator must also be taken into account when determining the time that the device must be held in reset.

TABLE 5: MAXIMUM TIME FROM POR RISING EDGE TO VALID V_{DD} VOLTAGE

	Osc Mode	Maximum Time	Conditions
PIC16C5X	LP, XT, HS, and RC	9 ms	
	RC	28 ms	
PIC16CXX	LP, XT, and HS	28 ms + $1024 T_{Osc}$	PWRTE = 1
	LP, XT, and HS	$1024 T_{Osc}$	PWRTE = 0
PIC17CXX	LF, XT, EC, and RC	Greater of (40 ms or $1024 T_{Osc}$)	

FIGURE 9: POR DELAY FOR FAST V_{DD} RISE TIME

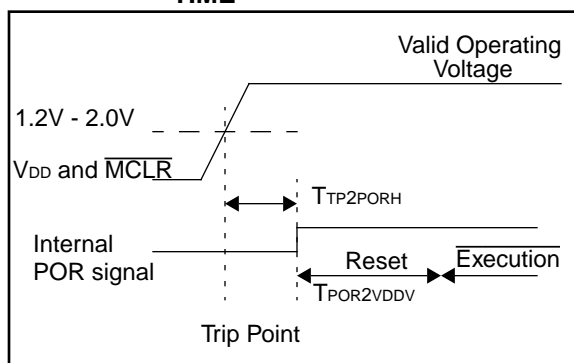


TABLE 6: RESET TIME DUE TO OSCILLATOR

	Clock Frequency								
	32 kHz	1 MHz	2 MHz	4 MHz	8 MHz	10 MHz	16 MHz	20 MHz	25 MHz
$1024 T_{Osc}$	32 ms	1.0 ms	512 μ s	256 μ s	128 μ s	102.4 μ s	64 μ s	51.2 μ s	41 μ s

AN607

FIGURE 10: MAXIMUM POWER-UP TIME, $\overline{\text{MCLR}}$ TIED TO V_{DD} (PIC16C5X, PIC16CXX, PIC17CXX)

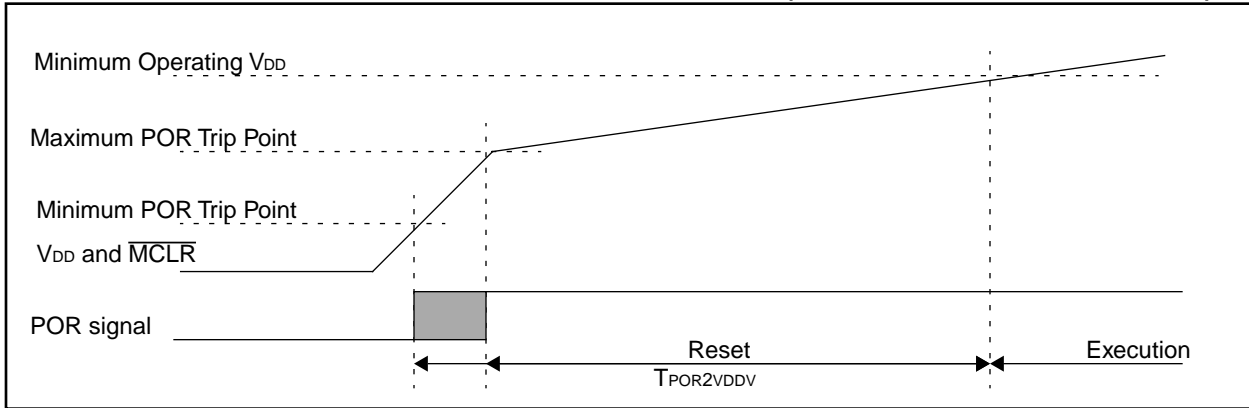


FIGURE 11: MAXIMUM POWER-UP TIME, $\overline{\text{MCLR}}$ NOT TIED TO V_{DD} (PIC16CXX)

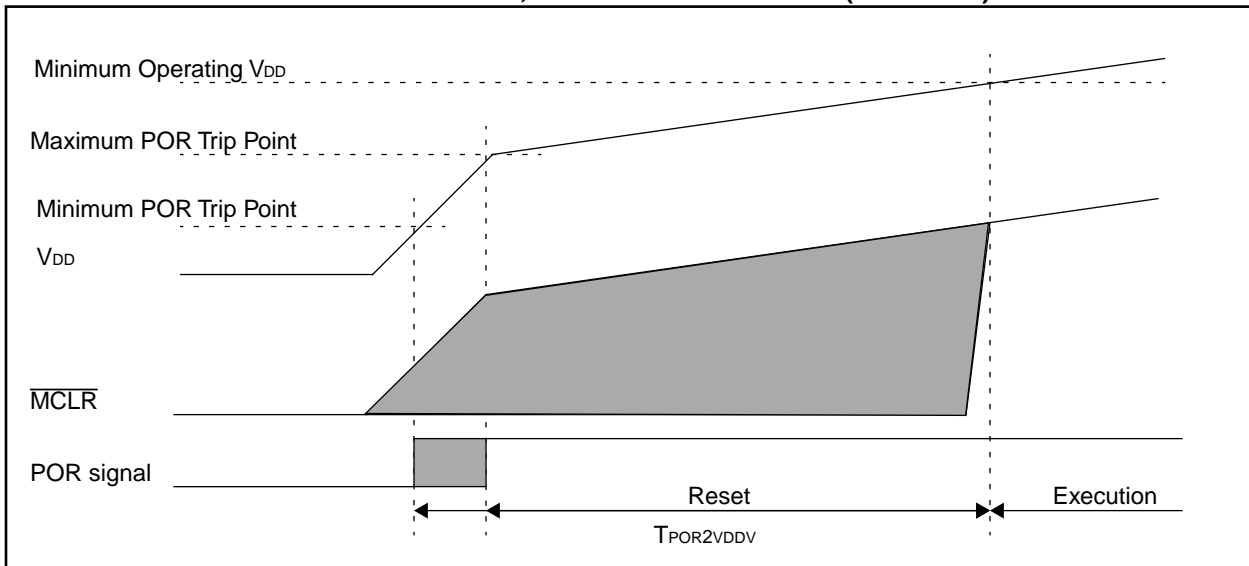
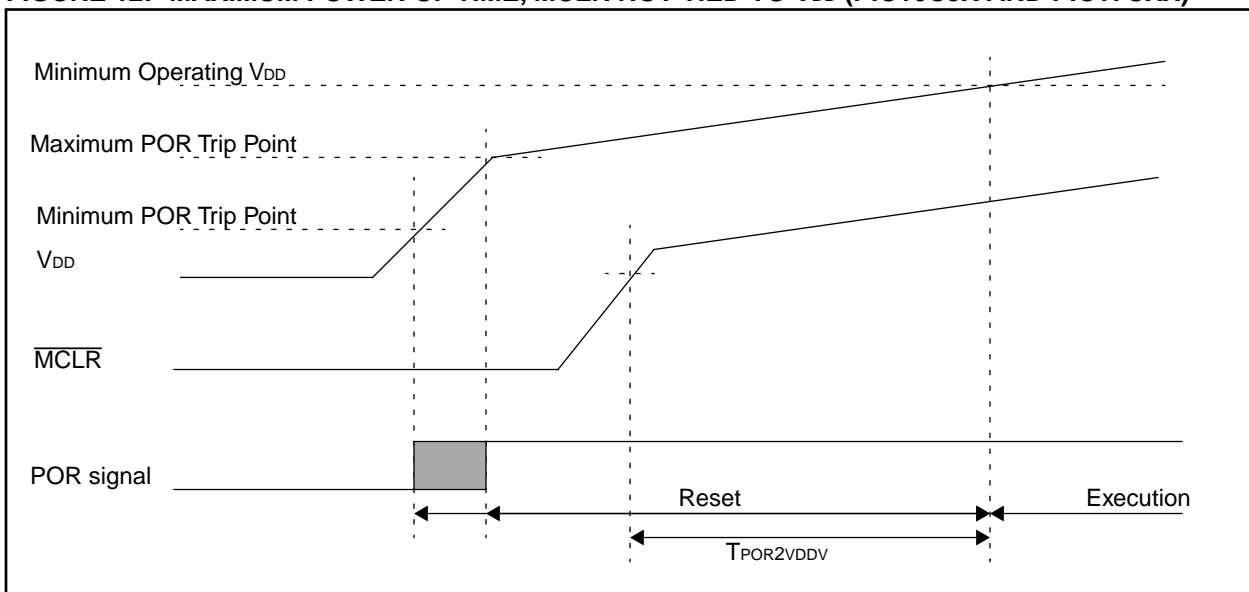


FIGURE 12: MAXIMUM POWER-UP TIME, $\overline{\text{MCLR}}$ NOT TIED TO V_{DD} (PIC16C5X AND PIC17CXX)



Oscillator and Resonator Considerations

Oscillators and resonators from different manufacturers may have different characteristics. The recommended capacitor selection can be found in each device's data sheet. When we do the capacitor selection, during the oscillator/resonator characterization, we are currently using devices from one of several manufacturers. Generally we use oscillators from either ECS, CTS, FOX or Epson, and ceramic resonators from either Murata Erie or Panasonic. Other manufacturers may be used in the future, depending on availability and other factors.

Other manufacturers devices may have significantly different characteristics. To ensure proper oscillator operation, the circuit should be verified at the lowest temperature/highest V_{DD} (to ensure that the crystal is not overdriven), and with the highest temperature/lowest V_{DD} (to ensure the device still starts up) that the device will be subjected to while in the application. This ensures a stable start-up and frequency for this device, at the extreme conditions of the application.

For production purposes, the above testing should be done with many different samples of the components selected. This is so the part to part variation of the capacitors, resistors, crystals/resonators, and PIC16/17 devices are taken into account. All PIC16/17 final data sheets supply the characterization information on the transconductance of the oscillator (measurement of gain). This information can be used to check part to part variations of the PIC16/17.

When selecting the crystal, the designer must ensure that it is a parallel cut type. Failure to use a parallel cut crystal may cause:

- Frequency operation out of the specified range of the crystal.
- Unreliable oscillator start-up.
- Device or crystal damage.

RAM and Special Function Register Initialization

After a successful Power-up Reset, the device will begin to execute the firmware program. To have expected operation, ALL RAM should be initialized by the program. This includes the Special Function Registers (SFR) and the general purpose data memory. The use (read) of an uninitialized RAM location will cause the program to do exactly what you told it, with the unexpected RAM value. It should not be expected that all devices will power-up with the same uninitialized device values.

There are many factors that contribute to how a RAM cell powers up, but the most common "gotcha" is between the Windowed and OTP device types. Many times a user forgets to cover the window after erasing the Windowed device. When the device is powering up, and the light is able to shine onto the device die, the transistor characteristics will shift. This can cause the

device RAM to have a different power-up value than a device where no light can shine onto the die (OTP or covered).

Note: RAM locations should be initialized before they are used. Use of an uninitialized location will cause proper device operation with the improper values. That is, it will do what you told it to do, not what you wanted it to do.

Valid Operating Voltage Levels

When the device is operating, the device voltage must be within the specified Min/Max limits. Operation of the device outside these limits may cause unexpected device operation.

One of the primary functional failure modes of a device is when the applied voltage is lower than the specified minimum requirement. This functional failure is called Brown-out. Brown-out causes the program memory not to be read correctly. For example, the program counter may be pointing to a MOVE instruction, but the device reads it as a GOTO instruction (with a random destination). This can have disastrous affects to the operation of the application. If brown-out conditions are possible, the application needs to be protected by using a brown-out circuit. A brown-out circuit works with the \overline{MCLR} pin to put the device in RESET before the device's actual voltage violates the minimum limit.

Figure 13 shows a low cost brown-out protection circuit. The voltage at which the circuit causes a reset is dependent upon the tolerances of the components. Figure 14 shows the use of a Dallas Semiconductor EconoReset. This device monitors the status of the power supply, and generates a reset when an out-of-tolerance condition is detected. Motorola also makes some 3-terminal devices to monitor the power supply, such as the MC34164, MC34064, MC33064. Their data sheets should be reviewed to ensure that the device is suitable for that devices application.

FIGURE 13: LOW COST BROWN-OUT PROTECTION CIRCUIT

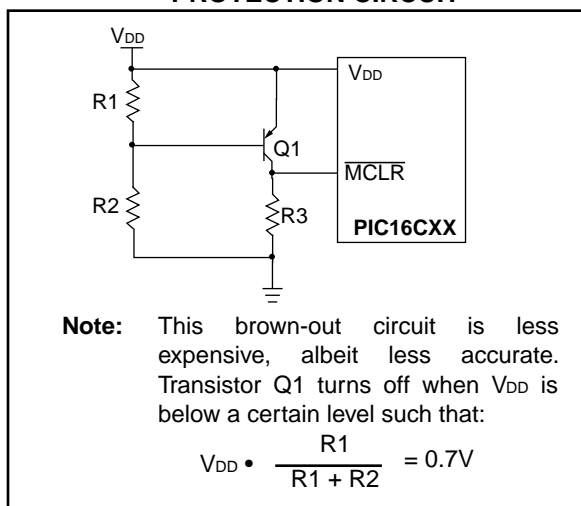
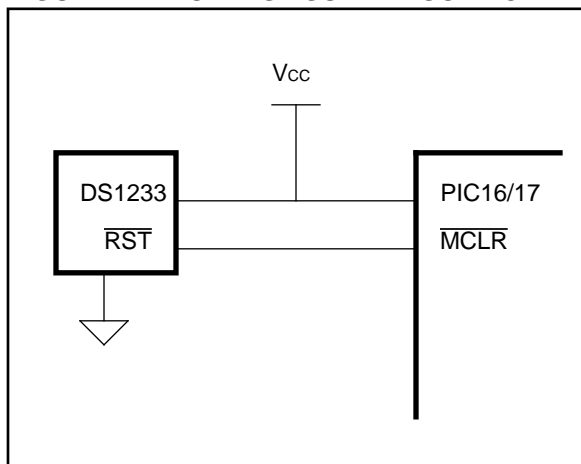


FIGURE 14: VOLTAGE SUPERVISORY CHIP



Brown-out and the WDT

The recommended solution for brown-out conditions is the use of a brown-out circuit. The brown-out circuit will keep the device in reset until a valid operating voltage is present. In some applications the additional cost of the external brown-out circuit, can be traded-off with system recovery from brown-out. Use of the Watchdog Timer (WDT) can enhance the probability of system recovery from a brown-out condition.

Note: If I/O drive conflicts can cause critical problems, this technique should not be used. This is due to the indeterminate time before a device reset could occur, which would reset all pins to inputs to eliminate any I/O conflict.

When using the WDT in brown-out conditions, care must be taken. Brown-outs may cause an unrecoverable condition, but with good design practice the probability of this can be significantly reduced.

During a brown-out, improper program execution can occur due to an EPROM read failure. This program execution can also corrupt data memory locations, which include the Special Function Registers (SFRs). Corrupting the control registers may cause hardware conflicts. For example, an input may become an output. Other conflicts are possible, but the situation is application dependent.

As the device voltage gets lower, internal logic can become corrupted. This can include the Program Counter (PC) value, Stack Pointer and contents, State machines, Data Memory, etc.

When a valid voltage is returned, the device may be at an unexpected program location, possibly using corrupted values. In this situation, the device would not be expected to operate as intended and could get into a state that appears locked-up.

For the PIC17C42 in code protected microcontroller mode, once the Program Counter (PC) exceeds the 32K-word boundary, the device will become locked-up. The PC can exceed the 32K-word boundary from the execution of incorrect instructions (due to failure reading the EPROM) or by the PC becoming corrupted.

If the WDT is to be used to reset the device, care must be used in structuring the program. Optimally, only one CLRWDT instruction should be used. This minimizes the possibility of program execution returning to a loop which clears the WDT. This loop could then lock-up the device, since other control registers are corrupted and the device is not configured as expected. An example is; if the loop was waiting for an interrupt, but the bit that enables global interrupts was disabled the device would no longer respond to the interrupts and would appear locked-up.

Example 1 shows a simple implementation of using the WDT reset for system recovery. The program loops, waiting for a WDT time-out (which clears the \overline{TO} bit). After the WDT reset, the \overline{TO} bit needs to be set (by executing a CLRWDT instruction). The program should then initialize the device. Then application code can start executing. There is a possibility of the \overline{TO} bit being corrupted by low voltage, and the device not being in a reset state when the software initializes the device.

The WDT example in Appendix B: uses a different method, independent of the \overline{TO} bit. This uses RAM locations which get loaded with a value. A WDT time-out (or other reset) needs to occur. The RAM locations are verified to contain the same values. Once the RAM is verified, it is cleared, and the device should be initialized. These RAM locations can be used by the application program.

EXAMPLE 1: USING WDT RESET

```

org   Reset_Address
      GOTO   TO_TEST      ;At any reset,
                          ;test the  $\overline{TO}$  bit

org   TO_TEST
      BTFSC STATUS, TO   ;WDT Time-Out?
      HERE  GOTO  HERE   ;NO, Wait for  $\overline{TO}$ 
      Time_Out
                          ;YES, Good Reset
      CLRWDT              ;Start here
      :                  ;Initialize Device
      :                  ;Application Code
    
```

False Power-down

In applications where power is removed from the device's supply lines, but voltage is still applied to an I/O pin, unexpected operation may occur. Power is able to be supplied to the device through this I/O pin. Since the device is still partially powered, the internal logic is never completely powered down. Figure 15 shows the general structure of an I/O pin. Figure 16 depicts the internal voltage level that is actually applied to some device logic, versus what is seen at the pin.

To guarantee a Power-on Reset (POR) rising edge, the device voltage (V_{DD}) must start from V_{SS} . When the device is inadvertently powered from an I/O pin, the voltage at the V_{DD} pin may appear to be near ground but may actually be higher in the device. With some of the internal logic powered, the characteristics of the device can be similar to a brown-out situation. Similar design practices to brown-out should be implemented.

A method for protecting the device from being powered from an I/O pin is shown in Figure 17.

FIGURE 15: TYPICAL ELECTRICAL STRUCTURE OF I/O PIN

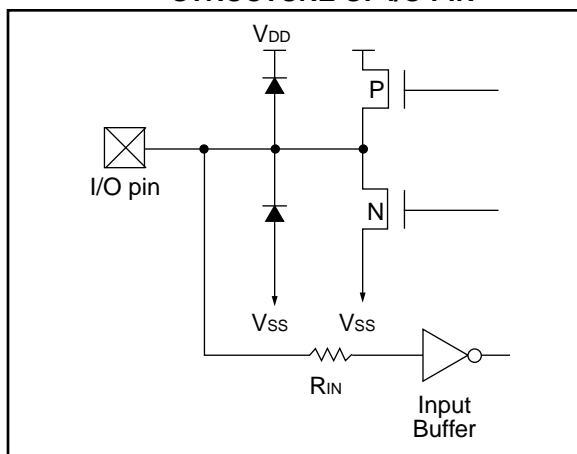


FIGURE 16: FALSE POWER-DOWN

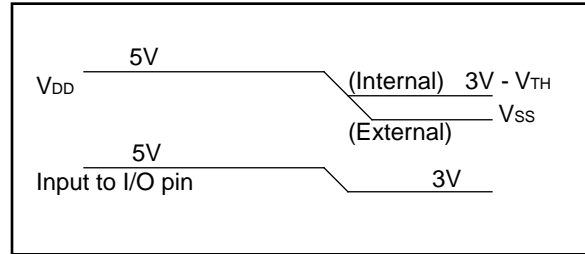
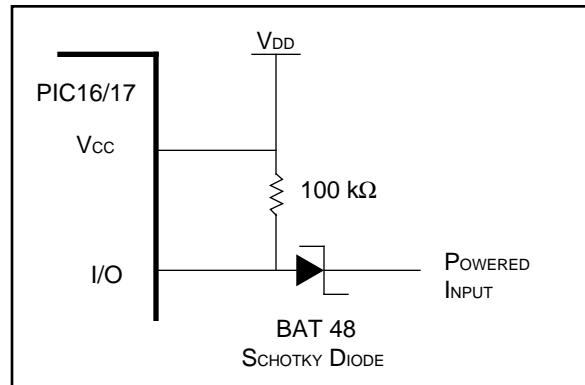


FIGURE 17: POWERED INPUT PROTECTION



In general, a brown-out detect circuit should cause the PIC16/17 to RESET (\overline{MCLR} forced low). This ensures that the internal logic is in a known state until a valid device voltage level is reached. The actual brown-out circuit depends on the voltage range of the device and the application requirements. A comprehensive brown-out circuit would use a dedicated device to monitor the voltage and force the \overline{MCLR} pin low when the voltage becomes lower than specified.

Another case of false power-down situations is when the power is removed from the system, but the capacitor loading keeps a non-zero voltage on the V_{DD} pin. When power is reapplied, the device never powered down so no power-on-reset will occur. A simple Brown-out circuit should fix this.

TROUBLESHOOTING

There are several techniques that can be used to troubleshoot problems related to powering up. First it is important to try to locate the source of the problem. These sources could be:

- No oscillation on OSC1/OSC2 pins
- Improper/no Program Execution

In cases where there is no oscillation on the OSC1/OSC2 pins, some of the following should be tried:

- a) Verify that there are good connections/the components are good.
- b) Verify that the crystal/resonator manufacturer is one that has been tested, if not try other capacitor values.
- c) See if an external clock (from a function generator) causes device operation to begin.
- d) Verify that all components are well grounded.
- e) If a scope probe is connected to the oscillator output, it must be a low capacitance/high impedance probe. If it is not, the oscillator may stop.

In cases where program execution is not as expected:

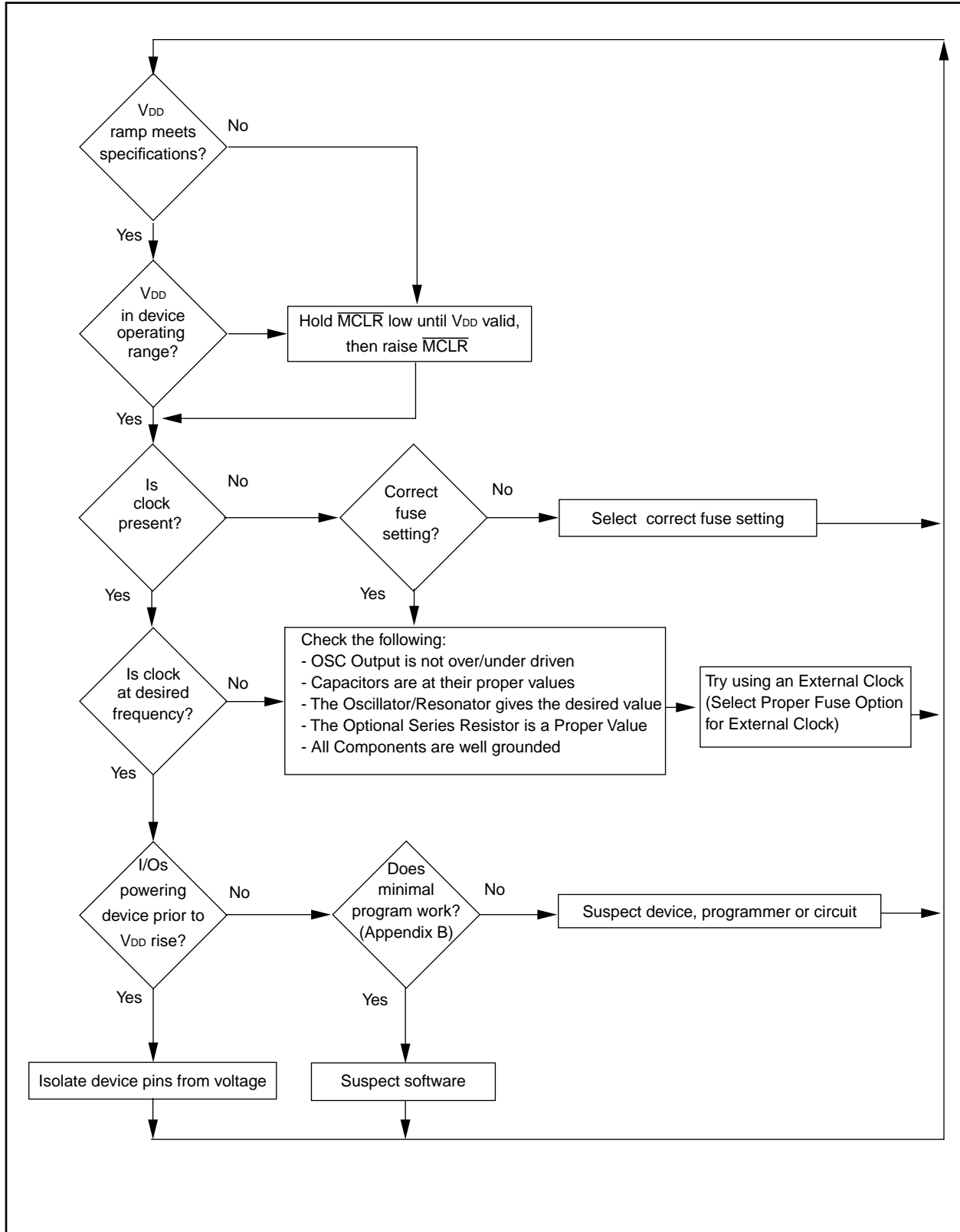
- a) Use a minimal program with external clock input.
- b) Tie \overline{MCLR} to ground until solid power is applied to the device then release \overline{MCLR} (bring high).
- c) Measure V_{DD} rise time to determine if an external reset circuit is needed, and, if so, what type of reset circuit should be used.
- d) Verify that the device program memory and configuration fuses are programmed to their expected states.

The flowchart shown in Figure 18 can be used to troubleshoot power-up problems. This flowchart is only intended to be the first level diagnostic in trying to solve a power-up problem. Many other flowcharts can be used, depending on the characteristics of the problem and the set-up of the application.

CONCLUSION

Understanding the criteria for the powering up of a device will allow you to make better design choices. If device power-up problems are still encountered, many techniques can be used to solve the problem. Appendix B contains example code which can be used to verify that a device is operating (powered-up correctly). This eliminates the possibility of the program as the cause, and allows debug on the hardware.

FIGURE 18: TROUBLESHOOTING FLOWCHART



APPENDIX A: Q & As

Q. ***When I use a windowed device (JW), my application works as expected. When I program an OTP device, it no longer works as expected. Why is this?***

A. The silicon is the same between the OTP and windowed devices. If the windowed device's window is not covered (with black tape), light shines onto the silicon. The light causes the potential levels of gates to shift. This in turn can cause RAM to be initialized to an unknown state, which could be different than in the OTP device. If RAM is not initialized by the program before it is used, these different power-up states of the RAM could be the cause of the problem. Ensure that all RAM is initialized in the device. This includes the SFRs.

Q. ***My oscillator is not oscillating, what could be wrong?***

A. There are several possibilities, some which include:

1. The wrong oscillator fuse setting is selected. The erased (default) state is RC oscillator mode.
2. The wrong capacitor values are installed. Refer to the most current data sheet for recommended values.
3. The characteristics of your manufacturers crystal are different than those that are characterized by Microchip. Generally our tests have been done with one of the following manufacturers' crystals/resonators: ECS, FOX, Murata Erie, or Panasonic.
4. The external connections to the device are wrong. Verify that all connections to the device are correct and that good signals / levels are being applied.
5. The cut of the crystal is a series type, as opposed to the specified parallel type.
6. No bypassing capacitors were used on the device. The noise on V_{DD} could be affecting the oscillator circuitry.

Q. ***The device was powered-down and then powered back up, but the device does not operate. What could be wrong.***

A. Possibilities include:

1. If power was applied to an I/O pin when the device was "powered-down", the device would be powered through the I/O pin. The internal logic is not actually powered-down, and Power-on Reset (POR) will not occur.
2. When V_{DD} was powered-down, V_{DD} was not given enough time to settle to 0V.
3. The V_{DD} ramp rate is too slow.

Q. ***My oscillator is oscillating, but the device is not working. What could be wrong?***

A. There are several possibilities, some which include:

1. Slow V_{DD} rise time, which was too slow to cause a Power-on Reset (POR). The rise time should not exceed the minimum device specification. For most devices this is 0.05 V/ms. Also the device must be at the minimum operating V_{DD} of the processor when reset is exited.
2. Ensure that the \overline{MCLR} pin is not low. This holds the device in RESET.
3. A brown-out has occurred, and has corrupted the internal state machines (including the WDT). An external brown-out circuit is recommended to hold the device in RESET during the brown-out condition.
4. The $CLRWDT$ instruction is not being used (often enough) when the WDT is enabled.

Q. ***When I power-up the device, it does not operate and it gets hot.***

A. Your design is probably permitting fast high voltage signals (spike) onto one of the device pins. This sudden high voltage (and associated current) is in excess of the protection diode limit. The device must be powered-down (to V_{SS}) to release this condition. This condition may cause a functional failure or affect device reliability. All Microchip devices meet or exceed the Human Body Model (HBM) and Machine Model (MM) for ESD and latch-up.

Q. ***My oscillator is oscillating, but not at the expected frequency. What could be wrong.***

A. For many designers, working with oscillators and their related issues are a “black magic”, since the characteristics can vary widely between manufacturers. I suggest that you read all the application notes that we have available on oscillators. Some quick possibilities are:

1. The cut of the crystal is a series type, as opposed to the specified parallel type.
2. No bypassing capacitors were used on the device. The noise on V_{DD} could affect the oscillator circuitry.
3. The capacitor values used are causing the oscillator to operate in one of the harmonic frequencies.

Note: This is not an all inclusive list. You may need to investigate other design aspects.

Q. ***The device seems to never exit reset, or is continually resetting.***

A. The `CLRWDT` instruction is not being used (often enough) when the WDT is enabled.

Q. ***The device was powered-down and back up again, but it does not reset. It just starts operating immediately.***

A. Possibilities include:

1. If power was applied to an I/O pin when the device was “powered-down”, the device would be powered through the I/O pin. The internal logic is not actually powered-down, and a Power-on Reset (POR) will not occur.
2. When V_{DD} was powered down, V_{DD} was not given enough time to settle to 0V.

Q. ***The oscillator is operating (I check it with a scope), yet when I look at other pins the program is not executing. Why?***

A. One possible reason is that when the oscilloscope probe is placed on the OSC2 pin, the additional capacitance is enough to cause oscillation to start. Removing the capacitive load of the probe causes the oscillation to stop.

AN607

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service, page 6-3.

APPENDIX B: TEST PROGRAMS

PIC16C5X BIT TOGGLE

MPASM 01.02.04 Intermediate C5X_B0T.ASM 12-20-1994 9:25:7

PAGE 1

```
LOC OBJECT CODE      LINE SOURCE TEXT
VALUE

0001          LIST    P = 16C54, F = INHX8M, n = 66
0002 ;
0003 ;*****
0004 ;
0005 ; This program is a minimam program to toggle a single I/O port pin for the
0006 ; 16C5x family of devices. The only initialization is that of the data
0007 ; direction register (TRIS) of the I/O pin and the Toggling of the pin.
0008 ; The waveform will be 1 unit high and 3 units low.
0009 ;
0010 ;      Program:      C5X_B0T.ASM
0011 ;      Revision Date: 12-20-94
0012 ;
0013 ;*****
0014 ;
0015 ;
0016 ; HARDWARE SETUP
0017 ;      None
0018 ;
0019 ;
0020          INCLUDE <p16C5x.inc>
0021 ;
0022          __FUSES ( _CP_OFF & _WDT_OFF & _XT_OSC )
0023 ;
0024 ;*****
0025 ;*****      Start program here.
0026 ;*****
0027 ;
0000          0028 START                                ; POWER_ON Reset (Beginning of program)
0000 0063          0029          CLRWF STATUS                ; Do initialization (Bank 0)
0001 0C00          0030          MOVLW 0x00                ; Specify value for PortB output latch
0002 0026          0031          MOVWF PORTB                ;
0003 0C00          0032          MOVLW 0x00                ; Specify which PortB pins are inputs / outputs
0004 0006          0033          TRIS PORTB                ;
0005 0506          0034 ;
0006 0406          0035 lzz          BSF PORTB, 0          ; B0 is High
0007 0A05          0036          BCF PORTB, 0          ; B0 is Low
0008          0037          GOTO lzz          ; Loop
0009          0038 ;
0010          0039 ;
0011          0040 ;
0012          0041 ;
0013          0042 ; Reset address. Determine type of RESET
0014          0043 ;
0015          0044          IFDEF __16C54
0016          01FF          0045 RESET_V EQU 0x1FF
0017          0046          ENDIF
0018          0047 ;
0019          0048          IFDEF __16C54A
0020          0049 RESET_V EQU 0x1FF
0021          0050          ENDIF
0022          0051 ;
0023          0052          IFDEF __16C55
0024          0053 RESET_V EQU 0x1FF
0025          0054          ENDIF
0026          0055 ;
0027          0056          IFDEF __16C56
0028          0057 RESET_V EQU 0x3FF
0029          0058          ENDIF
```

```

LOC  OBJECT CODE      LINE SOURCE TEXT
VALUE
                                0059 ;
                                0060     IFDEF      __16C57
0061 RESET_V          EQU      0x7FF
0062     ENDIF
0063 ;
0064     IFDEF      __16C58A
0065 RESET_V          EQU      0x7FF
0066     ENDIF
                                0067 ;
01FE                                0068 PROG_MEM_END    EQU      RESET_V - 1
                                0069 ;
                                0070 ;
                                0071     org      PROG_MEM_END      ; End of Program Memory
01FE 0BFE                                0072 ERR_LP_1      GOTO      ERR_LP_1      ; If you get here your program was lost
                                0073 ;
                                0074     org      RESET_V          ; RESET vector location
01FF 0A00                                0075 R_VECTOR     GOTO      START          ;
                                0076 ;
                                0077 ;
                                0078     end
                                0079
                                0080
                                0081

```

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```

0000 : XXXXXXXX-----
0040 : -----
0180 : -----
01C0 : -----XX

```

All other memory blocks unused.

```

Errors   : 0
Warnings : 0
Messages : 0

```

AN607

PIC16CXX BIT TOGGLE

MPASM 01.02.04 Intermediate CXX_B0T.ASM 12-20-1994 10:18:22

PAGE 1

```
LOC OBJECT CODE      LINE SOURCE TEXT
VALUE

0001          LIST    P = 16C74, F = INHX8M, n = 66
0002 ;
0003 ;*****
0004 ;
0005 ; This program is a minimam program to toggle a single I/O port pin for the
0006 ; 16Cxx family of devices. The only initialization is that of the data
0007 ; direction register (TRIS) of the I/O pin and the Toggling of the pin.
0008 ; The waveform will be 1 unit high and 3 units low.
0009 ;
0010 ;      Program:          CXX_B0T.ASM
0011 ;      Revision Date:   12-20-94
0012 ;
0013 ;*****
0014 ;
0015 ;
0016 ; HARDWARE SETUP
0017 ;      None
0018 ;
0019 ;
0020          INCLUDE <p16Cxx.inc>
0002 ; P16CXX.INC Standard Header File, Version 0.2      Microchip Technology, Inc.
0020
0021 ;
3FF9          0022      __FUSES ( _CP_OFF & _WDT_OFF & _XT_OSC & _PWDT_ON )
0023 ;
0024 ;*****
0025 ;*****      Start program here.
0026 ;*****
0027 ;
0000          0028 START                                ; POWER_ON Reset (Beginning of program)
0000 0183          0029          CLRWF STATUS                ; Do initialization (Bank 0)
0001 3000          0030          MOVLW 0x00                ; Specify value for PortB output latch
0002 0086          0031          MOVWF PORTB                ;
0003 1683          0032          BSF STATUS, RP0            ; Bank 1
0004 3000          0033          MOVLW 0x00                ; Specify which PortB pins are inputs / outputs
0005 0086          0034          MOVWF TRISB                ;
0006 1283          0035          BCF STATUS, RP0            ; Bank 0
0036 ;
0007 1406          0037 lzz          BSF PORTB, 0            ; B0 is High
0008 1006          0038          BCF PORTB, 0            ; B0 is Low
0009 2807          0039          GOTO lzz                    ; Loop
0040 ;
0041 ;
0042 ;
0043 ;
0044 ; End of Program Memory
0045 ;
0046          IFDEF __16C71
0047 PROG_MEM_END EQU 0x3FF
0048          ENDIF
0049 ;
0050          IFDEF __16C71A
0051 PROG_MEM_END EQU 0x3FF
0052          ENDIF
0053 ;
0054          IFDEF __16C73
0055 PROG_MEM_END EQU 0xFFF
0056          ENDIF
```


0057 ;
 MPASM 01.02.04 Intermediate CXX_B0T.ASM 12-20-1994 10:18:22 PAGE 2

```

LOC OBJECT CODE      LINE SOURCE TEXT
VALUE
0058      IFDEF      __16C74
0059      PROG_MEM_END EQU      0xFFFF
0060      ENDIF
0061 ;
0062      IFDEF      __16C61
0063      PROG_MEM_END EQU      0x3FF
0064      ENDIF
0065 ;
0066      IFDEF      __16C63
0067      PROG_MEM_END EQU      0x7FF
0068      ENDIF
0069 ;
0070      IFDEF      __16C64
0071      PROG_MEM_END EQU      0x7FF
0072      ENDIF
0073 ;
0074      IFDEF      __16C65
0075      PROG_MEM_END EQU      0xFFFF
0076      ENDIF
0077 ;
0078      IFDEF      __16C84
0079      PROG_MEM_END EQU      0x3FF
0080      ENDIF
0081 ;
0082      IFDEF      __16C84A
0083      PROG_MEM_END EQU      0x3FF
0084      ENDIF
0085 ;
0086 ;
0087      org      PROG_MEM_END      ; End of Program Memory
0088      ERR_LP_1  GOTO      ERR_LP_1 ; If you get here your program was lost
0089 ;
0090 ;
0091      end
0092
0093
0094
0095
  
```

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```

0000 : XXXXXXXXXXXX-----
0040 : -----
0F80 : -----
0FC0 : -----X
  
```

All other memory blocks unused.

```

Errors   : 0
Warnings : 0
Messages : 0
  
```

Note: Special Function Register data memory locations, in Bank 1, are specified by their true address in the file PIC16CXX.INC. The use of the MPASM assembler will generate a warning message, when those labels are used with direct addressing. Warning messages can be turned off with an assembler option.

AN607

PIC17CXX BIT TOGGLE

MPASM 01.02.04 Intermediate P17_B0T.ASM 12-19-1994 17:15:3

PAGE 1

```
LOC OBJECT CODE      LINE SOURCE TEXT
VALUE

0001          LIST      P = 17C42, F = INHX32, n = 66
0002 ;
0003 ;*****
0004 ;
0005 ; This program is a minimam program to toggle a single I/O port pin for the
0006 ; 17Cxx family of devices. The only initialization is that of the data
0007 ; direction register (TRIS) of the I/O pin and the Toggling of the pin.
0008 ; The waveform will be 1 unit high and 1 unit low.
0009 ;
0010 ;      Program:          P17_B0T.ASM
0011 ;      Revision Date:    12-20-94
0012 ;
0013 ;*****
0014 ;
0015 ;
0016 ; HARDWARE SETUP
0017 ;      None
0018 ;
0019 ;
0020          INCLUDE <p17Cxx.inc>
0021 ;
0022          __FUSES ( _MC_MODE & _WDT_NORM & _XT_OSC )
0023 ;
0024 ;*****
0025 ;*****      Start program here.
0026 ;*****
0027 ;
0000          0028 START                                ; POWER_ON Reset (Beginning of program)
0000 2904          0029          CLRF      ALUSTA          ; Do initialization
0001 290F          0030          CLRF      BSR            ; Bank 0
0002 B000          0031          MOVLW    0x00          ; Specify value for PortB output latch
0003 0112          0032          MOVWF    PORTB         ;
0004 B000          0033          MOVLW    0x00          ; Specify which PortB pins are inputs / outputs
0005 0111          0034          MOVWF    DDRB         ;
0006          0035 ;
0006 3812          0036 lzz          BTG      PORTB, 0    ; Toggle level on B0
0007 C006          0037          GOTO     lzz           ; Loop
0008          0038 ;
0009          0039 ;
0010          0040 ;
0011          0041 ;
0012          0042 ; End of Program Memory
0013          0043 ;
0014          0044          IFDEF    __17C42
0015          0045          PROG_MEM_END EQU    0x7FF
0016          0046          ENDIF
0017          0047 ;
0018          0048 ;
0019          0049          org      PROG_MEM_END        ; End of Program Memory
0020 07FF C7FF          0050          ERR_LP_1 GOTO    ERR_LP_1    ; If you get here your program was lost
0021          0051 ;
0022          0052 ;
0023          0053          end
0024          0054
0025          0055
```

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```
0000 : XXXXXXXX-----
0040 : -----
0080 : -----
07C0 : -----X
```

All other memory blocks unused.

```
Errors : 0
Warnings : 0
Messages : 0
```

WDT RESET WITH RAM VERIFY

MPASM 01.20 Released

BO_RAMT.ASM 6-30-1995 16:04:36

PAGE 1

```

LOC OBJECT CODE      LINE SOURCE TEXT
VALUE

00001          LIST    P = 17C44, F = INHX32, n = 66
00002 ;
00003 ;*****
00004 ;
00005 ; This program is a minimum program to recover from a brown-out condition, thru
00006 ; the use of the WDT. The method is to load RAM locations with a known value
00007 ; and compare these locations after any RESET. If the RAM location matches the
00008 ; expected value then program flow can continue. The longer this RAM string
00009 ; is, the greater the probability that the RAM would NOT power up in that state.
00010 ;
00011 ;
00012 ; NOTE: This does not Guarantee device recovery, Due to the random start-up
00013 ; point after brown-out. This point could be a loop with a CLRWDT
00014 ; instruction. The recommended solution is to always use a brown-out
00015 ; circuit.
00016 ;
00017 ; Program:          B0_RAMT.ASM
00018 ; Revision Date:   06-29-95
00019 ;
00020 ;*****
00021 ;
00022 ;
00023 ; HARDWARE SETUP
00024 ; None
00025 ;
00026 ;
0001 00027 TRUE      EQU    1
0000 00028 FALSE     EQU    0
00029 ;
0001 00030 Debug      EQU    TRUE
00031 #define     __CONFIG __FUSES
00032 ;
00033          INCLUDE <DEV_FAM.inc>
00102 list
00034 ;
00035 if ( P16C5X )
00036     INCLUDE <p16C5x.inc>
00037     __CONFIG ( _CP_OFF & _WDT_ON & _XT_OSC )
00038 endif
00039 ;
00040 if ( P16CXX )
00041     INCLUDE <p16Cxx.inc>
00042     __CONFIG ( _CP_OFF & _WDT_ON & _XT_OSC & _PWRT_ON )
00043 endif
00044 ;
00045 if ( P17CXX )
00046     INCLUDE <p17Cxx.inc>
00001 LIST
00002 ; P17CXX.INC Standard Header File, Version 2.01      Microchip Technology, Inc.
00298 LIST
FE00 FFE2 00047     __CONFIG ( _MC_MODE & _WDT_NORM & _XT_OSC )
00048     endif
00049 ;
00050 if ( P16C5X + P16CXX + P17CXX != 1 )
00051 MESSG "WARNING - USER DEFINED: One and only one device family can be selected"
00052     endif
00053 ;
00054          INCLUDE <BO_RAMT.inc>
00029 list
00055          INCLUDE <PMEM_END.inc>
00116 list
00056 ;
00057 ;
00058 ;*****
00059 ;**** Start program here.
00060 ;*****
00061 ;
0000 00062 org Reset_Address
00063 ; ; in the LIST directive
00064 if ( P16C5X )
00065     org 0h ; Override the start of this code.
00066     CLRF STATUS ; Force program memory to Page 0
00067     CLRF FSR ; Force Data Memory to Bank 0
00068     endif

```

AN607

```
00069 ;
00070     if ( P16CXX )
00071         CLRF   PCLATH           ; Force program memory to Page 0
00072         CLRF   STATUS          ; Force Data Memory to Bank 0
00073     endif
00074 ;
00075     if ( P17CXX )
00076         CLRF   PCLATH, F       ; Force program memory to Page 0
0000 2903 00077         CLRF   BSR, F       ; Force Peripheral / GP Data Memory to Bank 0
0001 290F 00078     endif
00079 ;
0002 C100 00080         GOTO   RAM_TEST      ; At any reset,
00081         ; test the RAM
00082 ;
00083 ; In RAM_TEST, program execution is held-off until a valid "warm" reset
00084 ; occurs. That is, the contents of some RAM locations retain the
00085 ; values that were written to them. The probability that the RAM would power-up
00086 ; in that state is dependent on the number of bytes of RAM used. The
00087 ; more RAM, the less the probability (probability = 1 / ( 2 ** 8(N+1) ).
00088 ;
00089 ;
0100      00090     org   MAIN           ; In Program Memory Page 0
0100      00091     RAM_TEST
0100 B0A5 00092         MOVLW  BYTE_0
0101 0520 00093         SUBWF  RAM0, F
0102 9204 00094         BTFSS  STATUS, Z   ; Result = 0?
0103 C110 00095         GOTO   LD_RAM      ; NO, Load Ram
00096 ;
0104 B00F 00097         MOVLW  BYTE_1       ; YES, Check next
0105 0521 00098         SUBWF  RAM1, F       ; location
0106 9204 00099         BTFSS  STATUS, Z   ; Result = 0?
0107 C110 00100         GOTO   LD_RAM      ; NO, Load RAM
00101 ;
00102 ;         :                       ; YES, Do Again
00103 ;         :                       ;
00104 ;
0108 B05A 00105         MOVLW  BYTE_n       ; YES, Check nth
0109 0522 00106         SUBWF  RAMn, F     ; location
010A 9204 00107         BTFSS  STATUS, Z   ; Result = 0?
010B C110 00108         GOTO   LD_RAM      ; NO, Load RAM
00109 ;
00110     if ( P16C5X || P16CXX )
00111         CLRF   RAM0           ; YES, Time-out
00112         CLRF   RAM1           ; occurred, clear
00113 ;         :                       ; RAM locations
00114 ;         :                       ;
00115         CLRF   RAMn          ;
00116     endif
00117 ;
00118 ;
00119     if ( P17CXX )
010C 2920 00120         CLRF   RAM0, F       ; YES, Time-out
010D 2921 00121         CLRF   RAM1, F       ; occurred, clear
00122 ;         :                       ; RAM locations
00123 ;         :                       ;
00124         CLRF   RAMn, F       ;
010E 2922 00125     endif
00126 ;
010F C117 00127         GOTO   Time_Out     ; Initialize Device
00128
0110      00129     LD_RAM
0110 B0A5 00130         MOVLW  BYTE_0       ; Load RAM
0111 0120 00131         MOVWF  RAM0         ; locations to
0112 B00F 00132         MOVLW  BYTE_1       ; compare against
0113 0121 00133         MOVWF  RAM1         ;
00134 ;         :                       ;
0114 B05A 00135         MOVLW  BYTE_n       ;
0115 0122 00136         MOVWF  RAMn        ;
00137 ;
0116 C116 00138     HERE   GOTO   HERE       ; Wait for WDT TO
0117      00139     Time_Out
0117 0004 00140         CLRWDT      ; Start here
00141 ;         :                       ; Initialize Device
00142 ;         :                       ; Application Code
00143 ;
00144     if ( Debug )           ;
00145         if ( P16C5X )
00146             CLRF   PORTB       ; PORTB output latch is cleared
00147             MOVLW  0x00        ;
00148             TRIS  PORTB       ; Port B is output
00149             BCF   PORTB, 0     ;
00150             BSF   PORTB, 0     ; Toggle pin B0
00151         endif
endif
```

```

00152 ;
00153     if ( P16CXX )
00154         CLRF    PORTB        ; PORTB output latch is cleared
00155         BSF    STATUS, RP0  ; Bank 1
00156         CLRF    TRISB       ; Port B is output
00157         BCF    STATUS, RP0  ; Bank 0
00158         BCF    PORTB, 0     ;
00159         BSF    PORTB, 0     ; Toggle pin B0
00160     endif
00161 ;
00162     if ( P17CXX )
0118 2912 00163         CLRF    PORTB, F    ; PORTB output latch is cleared
0119 2911 00164         CLRF    DDRB, F    ; Port B is output
011A 8812 00165         BCF    PORTB, 0     ;
011B 8012 00166         BSF    PORTB, 0     ; Toggle pin B0
00167     endif
00168     endif
00169 ;
011C C117 00170         GOTO    Time_Out      ; Return to start of Program
00171 ;
1FFF     00172         org      PROG_MEM_END ; End of Program Memory
1FFF     00173 ERR_LP_1
1FFF DFFF 00174         GOTO    ERR_LP_1      ; If you get here your program was lost
00175 ;
00176     if ( P16C5X )
00177         NOP
00178     ; This will cause the Program memory rollover
00179     ; for PIC16C5x devices
00179     endif
00180 ;
00181 ;
00182     end

```

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```

0000 : XXX-----
0040 : -----
0100 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX-----
0140 : -----
1F80 : -----
1FC0 : -----X
FE00 : X-----
FE40 : -----

```

All other memory blocks unused.

```

Errors   : 0
Warnings : 0
Messages : 0

```

AN607

NOTES:

Resistance and Capacitance Meter Using a PIC16C622

*Author: Rodger Richey
Logic Products Division*

INTRODUCTION

The PIC16C62X devices create a new branch in Microchip's PIC16CXX 8-bit microcontroller family by incorporating two analog comparators and a variable voltage reference on-chip. The comparators feature programmable input multiplexing from device inputs and an internal voltage reference. The internal voltage reference has two ranges, each capable of 16 distinct voltage levels. Typical applications such as appliance controllers or low-power remote sensors can now be implemented using fewer external components thus reducing cost and power consumption. The 18-pin SOIC or 20-pin SSOP packages are ideal for designs having size constraints.

The PIC16C62X family includes some familiar PIC16CXX features such as:

- 8-bit timer/counter with 8-bit prescaler
- PORTB interrupt on change
- 13 I/O pins
- Program and Data Memory

Device	Program Memory	Data Memory
PIC16C620	512 x 14	80 x 8
PIC16C621	1K x 14	80 x 8
PIC16C622	2K x 14	128 x 8

This family of devices also introduce on-chip brown-out detect circuitry and a filter on the reset input ($\overline{\text{MCLR}}$) to the PIC16CXX mid-range microcontrollers. Brown-out Detect holds the device in reset while V_{DD} is below the Brown-out Detect voltage of $4.0V, \pm 0.2V$. The reset filter is used to filter out glitches on the $\overline{\text{MCLR}}$ pin.

This application note will describe:

- Comparator module
 - operation
 - initialization
 - outputs
- Voltage Reference module
 - operation
 - initialization
 - outputs
- Linear slope integrating Analog to Digital conversion techniques
 - advantages
 - disadvantages
- Overview of the application circuit
- Detailed description of the measurement techniques used in the application circuit

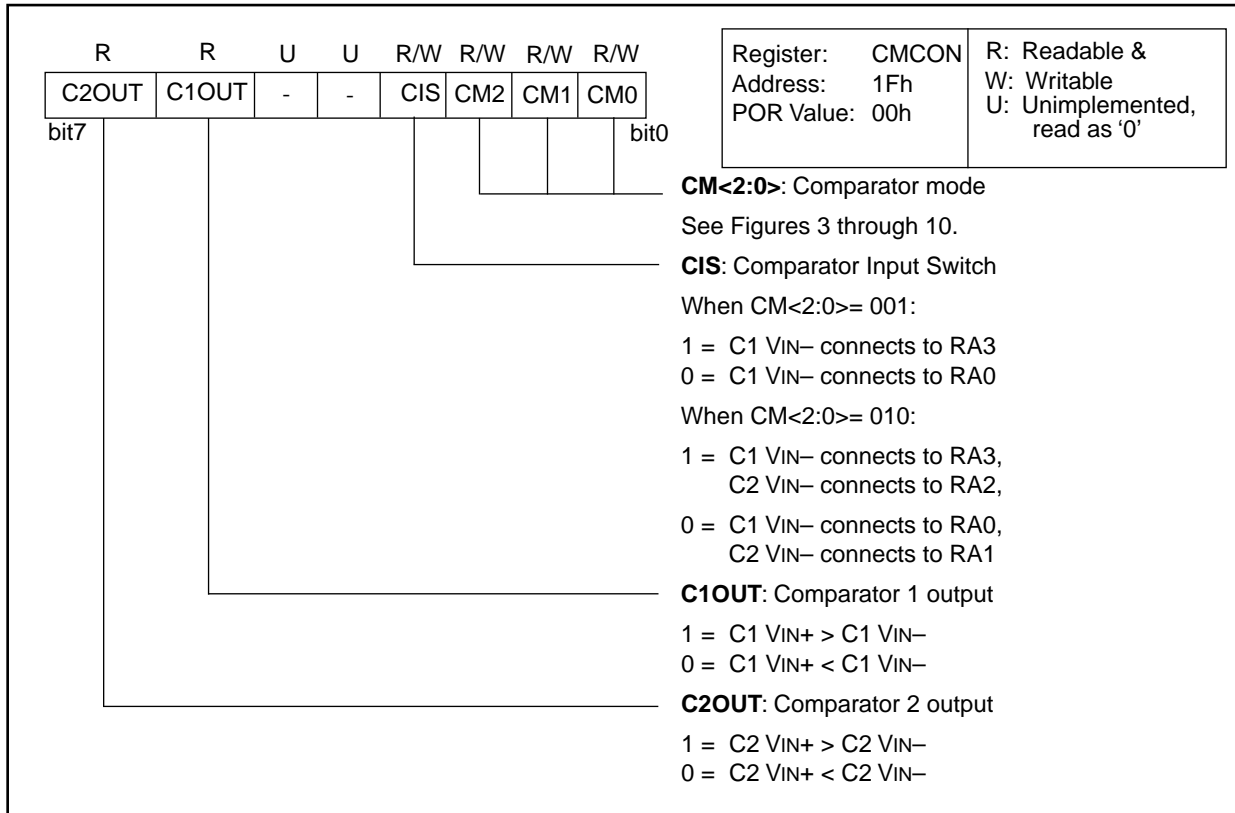
AN611

COMPARATOR MODULE

The comparator module contains two analog comparators with eight modes of operation. The inputs to the comparators are multiplexed with the RA0 through RA3 pins. The on-chip voltage reference can

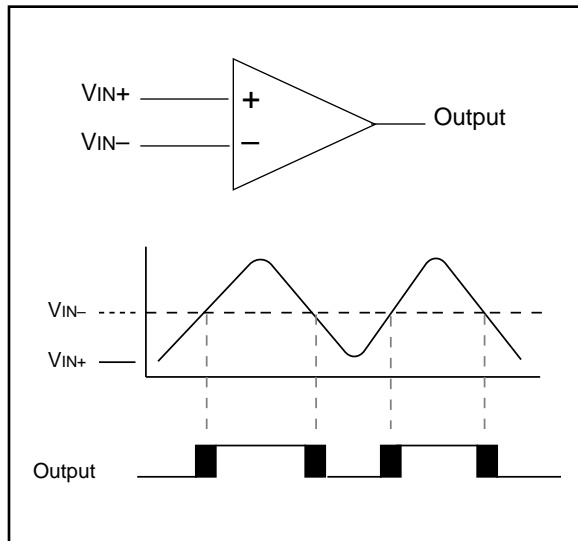
also be selected as an input to the comparators. The Comparator Control Register (CMCON) controls the operation of the comparator and contains the comparator output bits. Figure 1 shows the CMCON register.

FIGURE 1: CMCON REGISTER



A single comparator is shown in Figure 2. The relationship between the inputs and the output is also shown. When the voltage at VIN+ is less than the voltage at VIN-, the output of the comparator is at a digital low level. When the voltage at VIN+ is greater than the voltage at VIN-, the output of the comparator is at a digital high level. The shaded areas of the comparator output waveform represent the uncertainty due to input offsets and response time.

FIGURE 2: SINGLE COMPARATOR

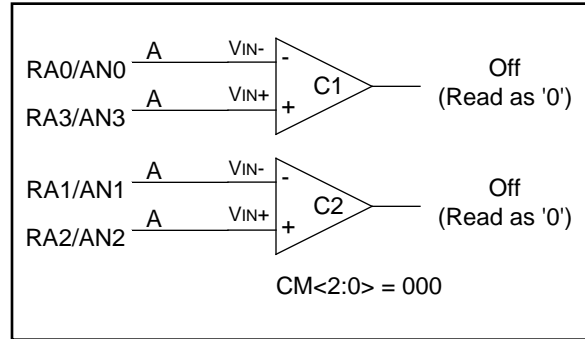


The TRISA register controls the I/O direction of the PORTA pins regardless of the comparator mode. If the comparator mode configures a pin as an analog input and the TRISA register configures that pin as an output, the contents of the PORTA data latch are placed on the pin. The value at the pin, which can be a digital high or low voltage, then becomes the input signal to the comparators. This technique is useful to check the functionality of the application circuit and the comparator module.

Comparator Operating Modes

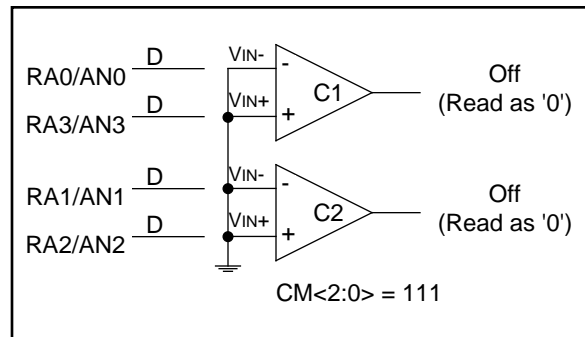
The analog inputs to the comparator module must be between VSS and VDD and one input must be in the Common Mode Range (CMR). The CMR is defined as VDD-1.5 volt to VSS. The output of a comparator will default to a high level if both inputs are outside of the CMR. If the input voltage deviates above VDD or below VSS by more than 0.6 volt, the microcontroller may draw excessive current. A maximum source impedance to the comparators of 10 kΩ is recommended. Figure 3 through Figure 10 show the eight modes of operation.

FIGURE 3: COMPARATORS RESET



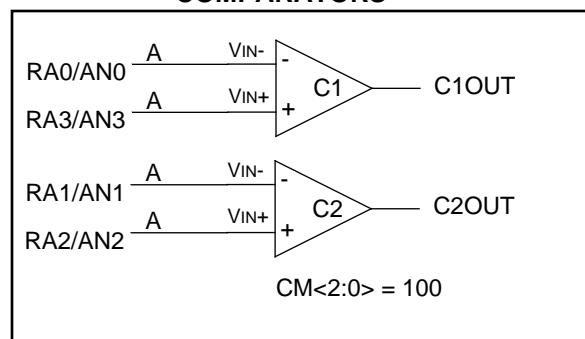
The Comparators Reset Mode (Figure 3) is considered the lowest power mode because the comparators are turned off and RA0 through RA3 are analog inputs. The comparator module defaults to this mode on Power-on Reset.

FIGURE 4: COMPARATORS OFF



The Comparators Off Mode (Figure 4) is the same as the Comparators Reset Mode except that RA0 through RA3 are digital I/O. This mode may consume more current if RA0 through RA3 are configured as inputs and the pins are left floating.

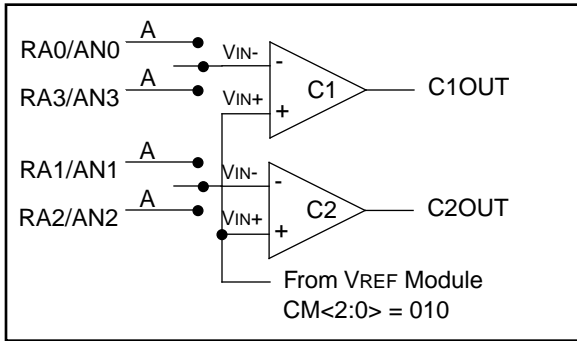
FIGURE 5: TWO INDEPENDENT COMPARATORS



The Two Independent Comparators Mode (Figure 5) enables both comparators to operate independently.

AN611

FIGURE 6: FOUR INPUTS MULTIPLEXED TO TWO COMPARATORS

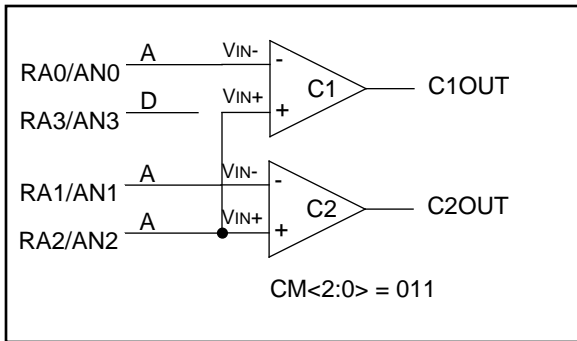


The Four Inputs Multiplexed to Two Comparators Mode (Figure 6) allows two inputs into the VIN- pin of each comparator. The internal voltage reference is connected to the VIN+ pin input of each comparator. The CIS bit, CMCON<3>, controls the input multiplexing to the VIN- pin of each comparator. Table 1 shows this relationship.

TABLE 1: COMPARATOR INPUT MULTIPLEXING

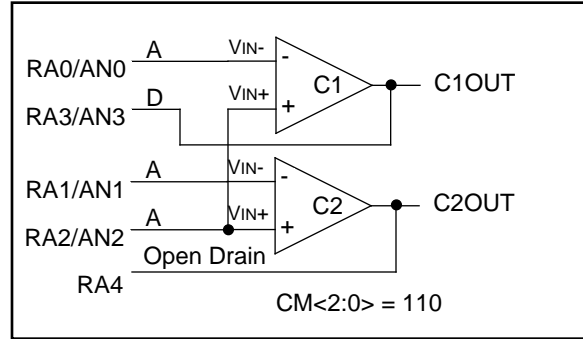
CIS	C1 VIN-	C2 VIN-
0	RA0	RA1
1	RA3	RA2

FIGURE 7: TWO COMMON REFERENCE COMPARATORS



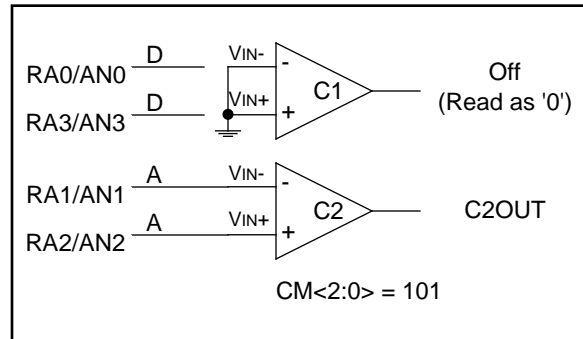
The Two Common Reference Comparators Mode (Figure 7) configures the comparators such that the signal present on RA2 is connected to the VIN+ pin of each comparator. RA3 is configured as a digital I/O pin.

FIGURE 8: TWO COMMON REFERENCE COMPARATORS WITH OUTPUTS



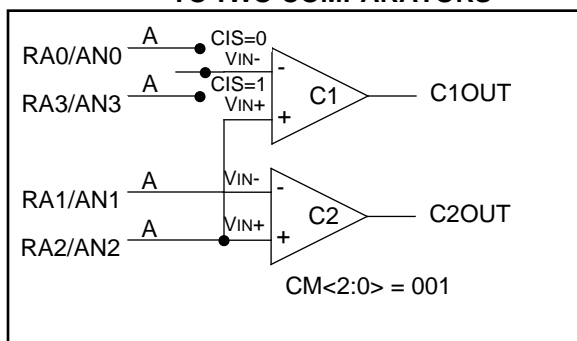
The Two Common Reference Comparators with Outputs Mode (Figure 8) connects the outputs of the comparators to an I/O pin. These outputs are digital outputs only with RA3 defined as a CMOS output and RA4 defined as an open drain output. RA4 requires a pull-up resistor to function properly. The value of resistance used for the pull-up will affect the response time of comparator C2. The signal present on RA2 is connected to the VIN+ pin of both comparators.

FIGURE 9: ONE INDEPENDENT COMPARATOR



The One Independent Comparator Mode (Figure 9) turns comparator C1 off making both RA0 and RA3 digital I/O. Comparator C2 is operational with analog inputs from RA1 and RA2.

FIGURE 10: THREE INPUTS MULTIPLEXED TO TWO COMPARATORS



The Three Inputs Multiplexed to Two Comparators Mode (Figure 10) connects the VIN+ pin of each comparator to RA2. The VIN- pin of comparator 2 is connected to RA1. The CIS bit, CMCON<3>, controls the input to the VIN- pin of comparator 1. If CIS = 0, then RA0 is connected to the VIN- pin. Otherwise RA3 is connected to the VIN- pin of comparator 1.

Note: Each comparator that is active will consume less power when the output is at a high level.

Clearing the Comparator Interrupt Flag

The comparator interrupt flag, CMIF, is located in the PIR1 register. This flag must be cleared after changing comparator modes. Whenever the comparator mode or the CIS bit is changed, the CMIF may be set due to the internal circuitry switching between modes. Therefore, comparator interrupts should be disabled before changing modes. Then, a delay of 10 μ s should be used after changing modes to allow the comparator circuitry to stabilize.

The steps to clear the CMIF flag when changing modes are as follows:

- Change the comparator mode or CIS bit
- 10 μ s delay
- Read the CMCON register to end the “mismatch” condition
- Clear the CMIF bit of the PIR1 register

The value of C1OUT and C2OUT are internally latched on every read of the CMCON register. The current values of C1OUT and C2OUT are compared with the latched values, and when these values are different a “mismatch” condition occurs. The CMIF interrupt flag will not be cleared if the CMCON register has not been read.

Using the Comparator Module

The CMCON register contains the comparator output bits C1OUT and C2OUT, CMCON<7:6>. These bits are read only. C1OUT and C2OUT follow the output of the comparators and are not synchronized to any internal clock edges. Therefore, the firmware will need to maintain the status of these output bits to determine the actual change that has occurred. The PIR1 register contains the comparator interrupt flag CMIF, PIR1<6>. The CMIF bit is set whenever there is a change in the output value of either comparator relative to the last time the CMCON register was read.

Note: If a change in C1OUT or C2OUT should occur when a read operation on the CMCON register is being executed (start of the Q2 pcycle), the CMIF interrupt flag may not be set.

When reading the PORTA register, all pins configured as analog inputs will read as a '0'. Analog levels on any pin that is defined as a digital input may cause the input buffer to consume more current than is specified.

The code in Example 1 shows the steps required to configure the comparator module. RA3 and RA4 are configured as digital outputs. RA0 and RA1 are configured as the VIN- inputs to the comparators and RA2 is the VIN+ input to both comparators.

EXAMPLE 1: INITIALIZING THE COMPARATOR MODULE

```

CLRF   PORTA           ;init PORTA
MOVLW  0X03           ;Two Common
MOVWF  CMCON          ;Reference
                          ;Comparators
                          ;mode selected
BSF    STATUS,RP0     ;go to Bank 1
MOVLW  0X07           ;Set RA<2:0> as
MOVWF  TRISA          ;inputs,RA<4:3>
                          ;as outputs
BCF    STATUS,RP0     ;go to Bank 0
CALL   DELAY10        ;10 $\mu$ s delay
MOVF   CMCON,F        ;read the CMCON
BCF    PIR1,CMIF      ;clear the CMIF
BSF    STATUS,RP0     ;go to Bank 1
BSF    PIE1,CMIE      ;enable compar-
                          ;ator interrupt
BCF    STATUS,RP0     ;go to Bank 0
BSF    INTCON,PEIE    ;enable global
BSF    INTCON,GIE     ;and peripheral
                          ;interrupts

```

The comparators will remain active if the device is placed in sleep mode, except for the Comparators Off Mode (CM<2:0>=111) and Comparators Reset Mode (CM<2:0>=000). In these modes the comparators are turned off and are in a low power state. A comparator interrupt, if enabled, will wake-up the device from sleep in all modes except **Off** and **Reset**.

AN611

Comparator Timings

The comparator module has a response time and a mode change to output valid timing associated with it. The response time is defined as the time from when an input to the comparator changes until the output of that comparator becomes valid. The response time is faster when the output of the comparator transitions from a high level to a low level. The mode change to output valid time refers to the amount of time it takes for the output of the comparators to become valid after the mode has changed. The internal voltage reference may contribute some delay if used in conjunction with the comparators (see Voltage Reference Settling Time).

VOLTAGE REFERENCE MODULE

The voltage reference is a 16-tap resistor ladder network that is segmented to provide two ranges of VREF values. Each range has 16 distinct voltage levels. The voltage reference has a power-down function to conserve power when the reference is not being used. The voltage reference also has the capability to be connected to RA2 as an output. Figure 11 shows the Voltage Reference Control Register (VRCON) register which controls the voltage reference. Figure 12 shows the block diagram for the voltage reference module.

FIGURE 11: VRCON REGISTER

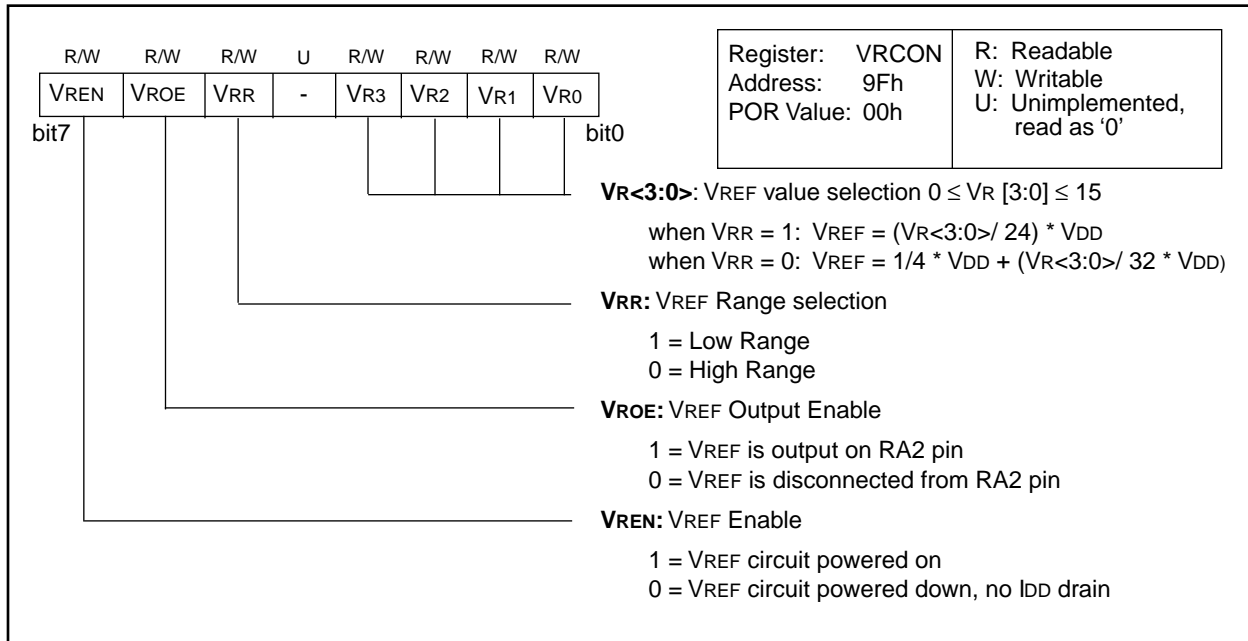
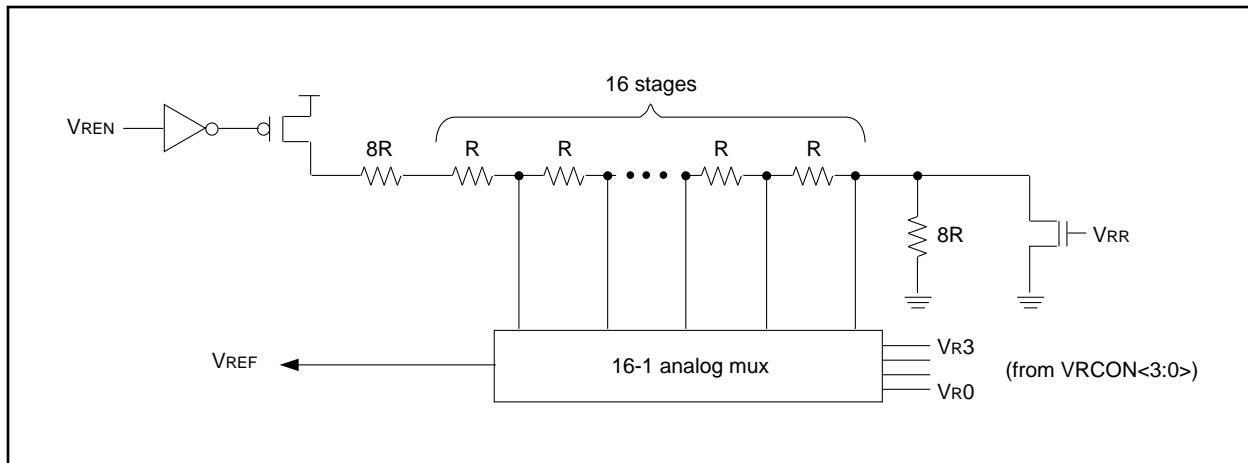


FIGURE 12: VOLTAGE REFERENCE BLOCK DIAGRAM



Note: The voltage reference is VDD derived and therefore, the VREF output changes with fluctuations in VDD.

Using the Voltage Reference

The voltage reference module operates independently of the comparator module. The output of the voltage reference may be connected to the RA2 pin at any time by setting the TRISA<2> bit and the VRCON<6> bit (VROE). It should be noted that enabling the voltage reference with an input signal present will increase current consumption. Configuring the RA2 pin as a digital output with the VREF output enabled will also increase current consumption. The increases in current are caused by the voltage reference output conflicting with an input signal or the digital output. The amount of increased current consumption is dependent on the setting of VREF and the value of the input signal or the digital output.

The full range of VSS to VDD cannot be realized due to the construction of the module (Figure 12). The transistors on the top and bottom of the resistor ladder network keep VREF from approaching VSS or VDD. Equation 1 and Equation 2 are used to calculate the output of the voltage reference.

EQUATION 1: VOLTAGE REFERENCE EQUATION, VRR=1

$$V_{REF} = (VR<3:0>/24) \times V_{DD}$$

EQUATION 2: VOLTAGE REFERENCE EQUATION, VRR=0

$$V_{REF} = (V_{DD}/4) + (VR<3:0>/32) \times V_{DD}$$

An example of how to configure the voltage reference is given in Equation 2. The reference is set for an output voltage of 1.25V at a VDD of 5.0V.

EXAMPLE 2: VOLTAGE REFERENCE CONFIGURATION

```

MOVLW 0X02      ;4 Inputs Muxed
MOVWF CMCON     ;to 2 comps.
BSF STATUS,RP0 ;go to Bank 1
MOVLW 0x07      ;RA3-RA0 are
MOVWF TRISA     ;outputs
MOVLW 0XA6      ;enable VREF,
MOVWF VRCON     ;low range
                ;set VR<3:0>=6
BCF STATUS,RP0 ;go to Bank 0
CALL DELAY10    ;10µs delay

```

If the voltage reference is used with the comparator module, the following steps should be followed when making changes to the voltage reference.

1. Disable the comparator interrupts
2. Make changes to the voltage reference
3. Delay 10 µs to allow VREF to stabilize
4. Delay 10 µs to allow comparators to settle
5. Clear the comparator interrupt flag
 - Read the CMCON register
 - Clear the CMIF bit
6. Enable comparator interrupts

The output of the voltage reference may be used as a simple DAC. However, the VREF output has limited drive capability when connected to the RA2 pin. In fact the amount of drive the voltage reference can provide is dependent on the setting of the tap on the resistor ladder. If VREF is used as an output, an external buffer must be utilized.

Voltage Reference Settling Time

Settling time of the voltage reference is defined as the time it takes the output voltage to settle within 1/4 LSB after making a change to the reference. The changes include adjusting the tap position on the resistor ladder, enabling the output, and enabling the reference itself. If the voltage reference is used with the comparator module, the settling time must be considered.

MAKING SIMPLE A/D CONVERSIONS

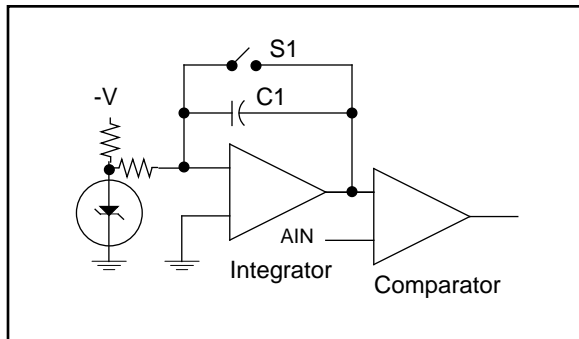
Linear slope integrating A/D converters are very simple to implement and can achieve high linearity and resolution for low conversion rates. The three types of converters that will be discussed are the single-slope, dual-slope, and modified single-slope converters. The following material was referenced from application note AN260, "A 20-Bit (1ppm) Linear Slope-Integrating A/D Converter", found in the Linear Applications Handbook from National Semiconductor®.

Single-Slope Integrating Converter

A single-slope integrating converter is shown in Figure 13. In a single-slope converter, a linear ramp is compared against an unknown input AIN. When the switch S1 is opened the ramp begins. The time interval between the opening of the switch and the comparator changing state is proportional to the value of AIN.

The basic assumptions are that the integrating capacitor C1 and the clock used to measure the time interval remain constant over time and temperature. This type of converter is heavily dependent on the stability of the integrating capacitor.

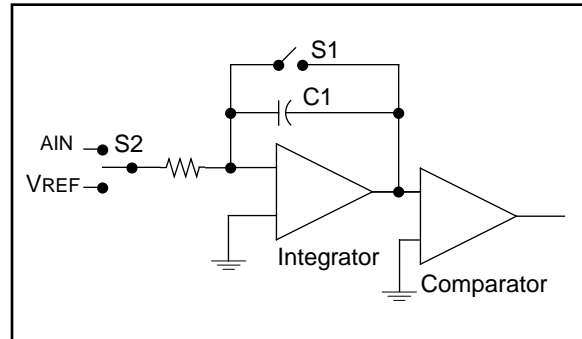
FIGURE 13: SINGLE-SLOPE INTEGRATING CONVERTER



Dual-Slope Integrating Converter

Figure 14 shows a dual-slope integrating converter. The dual-slope converter integrates the AIN input for a predetermined length of time. The voltage reference is then switched into the integrator input, using S2, which integrates in a negative direction from the AIN slope. The length of time the reference slope requires to return to zero is proportional to the value of AIN. Both slopes are made with the same integrating capacitor C1 and measured with the same clock, so they need only to be stable over one conversion cycle.

FIGURE 14: DUAL-SLOPE INTEGRATING CONVERTER



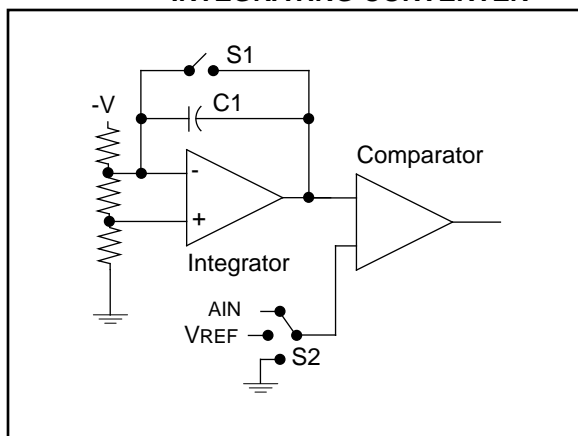
The dual-slope converter essentially removes the stability factor of the integrating capacitor from a conversion, however, the dielectric absorption of C1 has a direct effect. Dielectric absorption not only creates residual non-linearity in the dual-slope converter, but causes the converter to output different values for a fixed input as the conversion rate is varied. Dielectric absorption is defined as the capacitor dielectric's unwillingness to accept or give up charge instantaneously. This effect is modeled as a parasitic RC network across the main capacitor. A charged capacitor will require some time to discharge, even through a dead short, due to the parasitic RC network and some amount of charge will be absorbed by the parasitic C after charging of the main capacitor has stopped. Typically, Teflon, polystyrene and polypropylene dielectrics offer better performance than paper, mylar, or glass. Electrolytics have the worst dielectric absorption characteristics and should be avoided for use in slope integrating converters.

Modified Single-Slope Converter

The modified single-slope converter has been designed to compensate for the effects present in the previous converters. Resolutions of up to 16-bits can be achieved using high precision components and voltage reference source. Figure 15 shows the modified single-slope converter. Some features of this converter are:

- Continuously corrects for zero and full-scale drifts in all components of the circuit.
- The integrating capacitor C1 is charged periodically and always in the same direction. The error induced from dielectric absorption will be small and can be compensated by using an offset term in the calibration procedure.
- The ramp voltage always approaches the comparator trip point from the same direction and slew rate.
- There is no noise rejection capability because the input signal is directly coupled to the comparator input. A filter at the comparator input would cause a delay due to the settling time of the filter.

FIGURE 15: MODIFIED SINGLE-SLOPE INTEGRATING CONVERTER



The microcontroller sends a periodic signal to the switch S1 regardless of the operating mode of the system. The output of the integrator is a fixed frequency, period and height signal which is fed into the input of the comparator. The time between ramps is long enough to allow the integrating capacitor C1 to discharge completely. The other input is multiplexed with ground, reference, and the AIN through switch S2. When the microcontroller starts a conversion, the ground signal is switched into the comparator and the time for the ramp to cross zero is measured and stored. The same measurements are repeated for the reference and AIN signals. Assuming that the integrator ramps are highly linear, Equation 3 is used to determine the value of AIN.

EQUATION 3: OUTPUT EQUATION FOR THE MODIFIED-SLOPE CONVERTER

$$AIN = \frac{\tau_{AIN} - \tau_{GND}}{\tau_{VREF} - \tau_{GND}} \times K \mu V$$

where τ_{AIN} is the measured time for the AIN signal, τ_{VREF} is the measured time for the voltage reference signal, τ_{GND} is the measured time for the ground signal, and K is a constant (typically 10^7).

APPLICATION CIRCUIT

The application circuit, called PICMETER, uses a PIC16C622 as a resistance and capacitance meter. The PICMETER uses a variation of the single-slope integrating convertor. The linear slope and integrator of Figure 13 are replaced with the exponential charge waveform of a RC Network. The charge time of a known component is compared against the charge time of an unknown component to determine the value of the unknown component.

A schematic of the PICMETER is shown in Figure 16. All reference designators cited in this section refer to this schematic. Results are transmitted to a PC which displays the value measured. The PICMETER can measure resistance in the range 1 K Ω to 999 K Ω and capacitance from 1 nF to 999 nF.

The following sections describe, in detail, the hardware, firmware, and PC software used in the application circuit. Appendix A shows the PICMETER firmware and Appendix B has the PC software. Appendix C contains the PCB layout.

Power

The RS-232 serial port provides power to the PICMETER. The RTS and DTR lines from the serial port output 3V to 11V to the PICMETER. The diodes D2 and D3 prevent any damage to the PC's serial port. Resistor R10 is used to current limit the Zener diode, D4. D4 is used to regulate the RTS and DTR voltage to 5.6V. Capacitors C3 and C4 provide power supply filtering to the Zener diode and the PIC16C622. This method of supplying power to devices using a serial port, such as a trackball or mouse, is very simple considering that the PICMETER requires approximately 7 mA to function.

Switches

Switch S1 is used to select either a resistor or capacitor measurement. RB5 of the PIC16C622 is used to detect what type of component is being measured. This switch also swaps the unknown component into the RC network.

If a resistor is the unknown component and a capacitor measurement is requested, the circuit reduces to a resistor divider on the VIN- pin of the comparator. This would result in a measured value of 0 pF if the voltage on the resistor divider network is greater than the voltage reference setting. Otherwise an error is detected. If a capacitor is the unknown component and a resistor measurement is selected, the circuit reduces to a capacitor divider network on the VIN- pin of the comparator. This case will also produce an error message.

Resistor measurements that are started without any component connected to the measuring terminals will cause an error. Capacitor measurements without a component connected to the measuring terminals will give a result of 0 pF.

Switch S2 is used to initiate a measurement. The switch is connected to RB6 of the PIC16C622 and the PORTB wake-up on change interrupt is used to detect a key press. A modified version of the firmware in AN552, "Implementing Wake-up on Key Stroke" was used to control the interrupt.

Measuring the Charge Time

The procedures for measuring a resistor or capacitor are the same except for the I/O pins used to control the RC networks. This also applies when measuring a known or unknown component.

Measurement Overview

The charge time of the unknown RC network is measured using Timer0. This value is multiplied by the known value of resistance or capacitance and stored in an accumulator. Then the charge time of the known RC network is measured. The accumulator is divided by the known RC network charge time to give the value of resistance or capacitance of the unknown component. Equation 4 shows the equation used to calculate resistance and Equation 5 shows the capacitance equation.

EQUATION 4: RESISTANCE EQUATION

$$R_{UNK} = \frac{\tau_{UNK} \times R_{KN}}{\tau_{KN}}$$

EQUATION 5: CAPACITANCE EQUATION

$$C_{UNK} = \frac{\tau_{UNK} \times C_{KN}}{\tau_{KN}}$$

R_{UNK} and C_{UNK} are the unknown resistor or capacitor values. R_{KN} and C_{KN} are the known resistor and capacitor values. τ_{UNK} and τ_{KN} are the charge times for the unknown and known components.

Detailed Measurement Description

The first step in measuring the charge time of either the known or the unknown RC networks is to reconfigure the I/O pins. The default state of the PORTA and PORTB pins connected to the RC network are all grounded outputs. This discharges all capacitors in the RC networks. The unknown component is measured first, so the known component, R4 or C1, is removed from the RC network. This is accomplished by making RB0 or RB2 on the PIC16C622 an input. Connections to the other RC network are kept grounded.

The analog modules are now initialized. The mode of the comparators is set to Four Inputs Multiplexed to Two Comparators (Figure 6). The CIS bit, CMCON<3> is cleared to select RA0 as the VIN- input to comparator 1 and RA1 as the VIN- input to comparator 2. The voltage reference is enabled, the output is disabled, and the high range is selected. The tap on the resistor ladder is set to 12. The value of 12 was selected because it is the lowest value of VREF that will trip the comparators, yet gives a time constant long enough to achieve good resolution for the measurement. After a 20 msec delay, which allows the analog modules to stabilize, the comparator flag is cleared. Comparator interrupts are enabled and Timer0 is cleared. Finally, the PEIE bit is set to enable comparator interrupts and the GIE bit is set to enable interrupts.

Now that the analog systems are ready, Timer0 is cleared again and power is applied to the unknown RC network by setting RB1 or RB3 high. Timer0 begins to increment a set of three registers which are cascaded together. These registers contain the charge time of the component. While waiting for the DONE flag, the ERROR flag is checked. See the Error Message section for an explanation of error detection. When the capacitor voltage trips the comparator, Timer0 is prevented from further incrementing the time registers and the DONE flag is set. The value in the time registers is τ_{UNK} .

The analog modules are now disabled. The comparator interrupts are disabled and the comparators are turned off (CM<2:0>=111). RA0 through RA3 and RB0 through RB4 are set up as grounded outputs to discharge the capacitors in the RC networks. This prevents a false reading during the next measurement. The voltage reference is disabled to conserve power and all interrupt flags are cleared. Extra delay loops are added at this time to ensure that the capacitors are discharged.

The charge time, τ_{UNK} , is then multiplied by the value of known resistance or capacitance. These values, in pF or Ω , were obtained by measuring the known RC networks with a Fluke meter. Each of these values is a 24-bit number. The result of multiplication is a 56-bit number which is stored in accumulators ACCb (most significant 24-bits) and ACCc (least significant 24-bits).

The process now repeats itself, except this time the charge time of the known RC network is measured. Now the unknown component is removed from the RC network by making the connections from the PIC16C622 inputs. The analog modules are initialized and the same procedure explained above is followed to measure the charge time of the known RC network. The 56-bit result previously stored in accumulators ACCb and ACCc is now divided by the charge time of the known component, τ_{KN} . This result is a 24-bit number which has the units of pF or Ω . This value is then transmitted to the PC.

RS-232 Transmission

PICMETER uses a transmit only, software implemented serial port adapted from AN593, "Serial Port Routines Without Using the RTCC". Hardware hand-shaking is not used. Since the serial port is realized in software, all interrupts must be disabled during transmission otherwise the baud rate can get corrupted.

On power-up, PICMETER sends a boot message to the PC which is "PICMETER Booted!". Otherwise, a four byte packet structure with a command byte and 3 data bytes is used. The command byte contains one of four possible commands:

- ASCII 'S' signifies that a measurement has been initiated
- ASCII 'E' tells the PC that an error has been detected
- ASCII 'R' tells the PC that resistance data is contained in the three data bytes
- ASCII 'C' tells the PC that capacitance data is contained in the three data bytes

The first data byte for the 'R' and 'C' commands contain the MSB of the measured value. The last data byte contains the LSB of the measured value. The three data bytes for the commands 'S' and 'E' do not contain any useful information at this time.

An 'S' command is issued every time the start switch, S2, is pressed. PICMETER then sends an 'R' or 'C' command for a valid measurement or an 'E' command when an error is detected.

Since the PICMETER operates from a single supply voltage, a discrete transistor is used as a level shifter. This insures that a low output on the RS-232 TXD line is between -3V and -11V. When the TXD line, RB7, from the PIC16C622 is at a logic high level, the transistor Q1 is off. The RXD line of the computer will then be at approximately the same voltage as the TXD line, -11V to -3V. A logic low level from RB7 of the PIC16C622 will turn on transistor Q1. This will bring the RXD line of the computer to about the same voltage of the DTR or RTS line, +3V to +11V.

The pins of interest on the DB9 connector CON1 are:

- pin 2 - RXD
- pin 3 - TXD
- pin 4 - DTR
- pin 5 - GND
- pin 7 - RTS

RTS, DTR, and GND provide power and ground to the PICMETER. RXD is connected to the collector of transistor Q1. TXD is connected to RXD through resistor R14. Since hardware hand-shaking is not implemented on the PICMETER, DSR (pin 6) and CTS (pin 8) are left disconnected.

The demo board developed by Microchip was intended to connect directly to a 9-pin serial port. A 9-pin male-to-female cable may also be used. These boards were manufactured by Southwest Circuits located in Tucson, Arizona (Appendix C). The PCB layout for this demo board is shown in Appendix C.

Error Message

The error message is sent only when the PICMETER is making a measurement and detects an error. The range of resistance that the PICMETER measures is 1 k Ω to 999 k Ω . Using the value of C2, 1 μ F, the range of charging times for resistance measurements is 1msec to 999 ms. The range of capacitor charging times is also 1 ms to 999 ms using the resistance value of R3, 1M Ω , and a capacitor measuring range of 1 nF to 999 nF. A ceramic resonator of 4 MHz gives Timer0 a resolution of 1 μ sec. Therefore, the highest count that the time registers should reach is 999,000. This is a 20-bit number. If the 21st bit should ever be set, it is assumed that the PICMETER is trying to measure the air gap between the measuring terminals, a component that is out of range, or switch S1 is not set correctly for the component in the measuring terminals.

24-Bit Math Routines

The 24-bit math routines were developed using simple algorithms found in any computer math book. These math routines include addition, subtraction, multiplication, division, and 2's complement. Four 24-bit accumulators located in the general purpose RAM area of the PIC16C622 are used by the math routines: ACCa, ACCb, ACCc, and ACCd. Table 2 shows the relationship between the math routines and the accumulators.

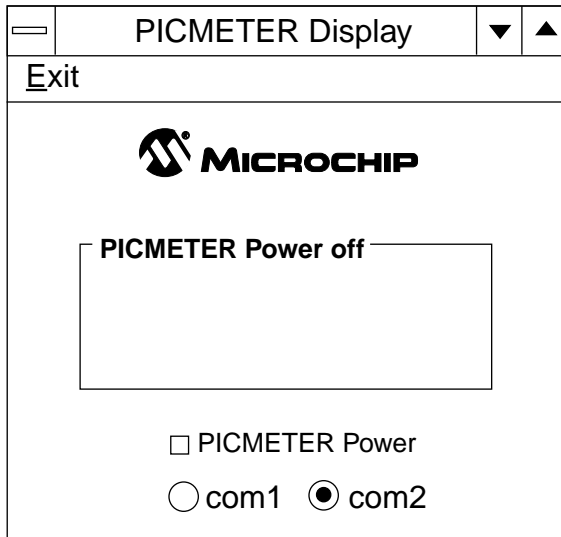
TABLE 2: MATH ROUTINE ACCUMULATORS

Name	Operation	Result	Temp. Storage
Add	ACCa + ACCb	ACCb	N/A
Subtract	2's Comp ACCa then	ACCa	N/A
	ACCa + ACCb	ACCa	
Multiply	ACCa x ACCb	ACCb (MSB's) ACCc (LSB's)	ACCd
Divide	ACCb:ACCc ACCa	quotient in ACCc remainder in ACCb	ACCd
2's Comp	NOT(ACCa) + 1	ACCa	N/A

Computer Program

The program that receives data from the PICMETER was written in Visual Basic® from Microsoft® for the Windows® environment. Figure 17 show the display of the Windows based PICMETER program.

FIGURE 17: PICMETER PC PROGRAM



The operation of this program is simple. A functional description is given below:

- a) Select the appropriate COM port by clicking on the COM1 or COM2 buttons.
- b) Turn power on to the PICMETER by clicking on the PICMETER Power button.
- c) The frame message should read "PICMETER Booted!", the frame contents will be cleared, and the LED on the PICMETER should be on.
- d) The switch S1 selects the type of component that is in the measuring terminals.
- e) Pressing the START button, S2, on the PICMETER will initiate a measurement. The frame message should read "Measuring Component" and the contents of the frame will be cleared.
- f) When the measurement is complete, the frame message will read "Resistance" or "Capacitance" depending on the position of switch S1. The value of the component will be displayed in the frame as well as the units.
- g) If an error is detected, the frame message will read "Error Detected". This is only a measurement error. Check the component on the measuring terminals and the position of switch S1.
- h) Turn off the PICMETER by clicking on the PICMETER Power button. The frame message will change to "PICMETER Power OFF", the frame contents will be cleared, and the LED on the PICMETER will turn off.

Appendix B contains a complete listing of the Visual Basic program.

PICMETER ACCURACY

The PICMETER measures capacitance in the range of 1 nF to 999 nF. Table 3 shows a comparison of various capacitors. All capacitors have a tolerance of 10% and have various dielectrics. The average error percentage is 3%.

TABLE 3: CAPACITANCE MEASUREMENTS

Capacitance Accuracy			
Marked Value	Fluke Value	PICMETER Value	Error %
2.2 nF	2.3 nF	2.2 nF	4.3
2.5 nF	2.63 nF	2.5 nF	4.9
20 nF	16.5 nF	16.3 nF	1.2
33 nF	35.2 nF	35.8 nF	1.7
47 nF	45 nF	44.5 nF	1.1
50 nF	52 nF	52.9 nF	1.7
100 nF	99.7 nF	93 nF	6.7
0.1 μ F	95 nF	96.1 nF	1.2
0.1 μ F	99.4 nF	102.8 nF	3.4
0.22 μ F	215 nF	215.2 nF	0.1
470 nF	508 nF	518.9 nF	2.1
940 nF	922 nF	983.1 nF	6.6

The 2.5 nF, 100 nF and 940 nF capacitors all have polyester dielectric material. The Equivalent Series Resistance (ESR) of polyester capacitors is typically high which would cause the PICMETER to have a larger error than other dielectrics. If the error percentages for these capacitors is ignored, the average error decreases to 1.9%.

The resistance range of the PICMETER is 1 k Ω to 999 k Ω . Table 4, Resistance Measurements, shows a comparison of various resistors in this range. All resistors have a tolerance of 5%. The average error percentage is 1%.

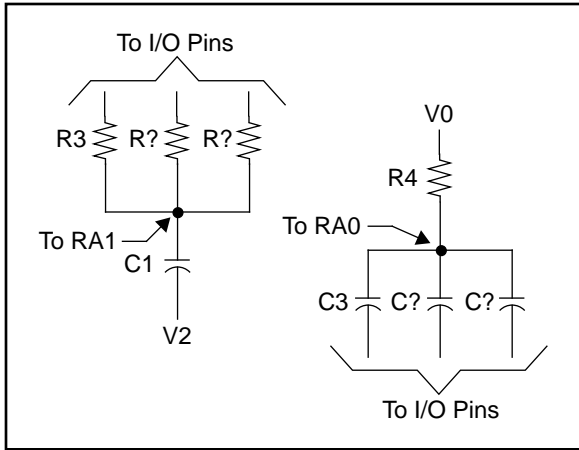
TABLE 4: RESISTANCE MEASUREMENTS

Resistance Accuracy			
Marked Value	Fluke Value	PICMETER Value	Error %
1.2K	1.215K	1.2K	1.3
5.1K	5.05K	5.0K	1.0
8.2K	8.47K	8.3K	2.0
10K	10.2K	10K	2.0
15K	15.36K	15.1K	1.7
20K	20.8K	20.5K	1.5
30K	30.4K	30K	1.4
51K	50.3K	49.8K	1.0
75K	75.5K	74.8K	1.0
91K	96.4K	95.9K	0.6
150K	146.3K	145.6K	0.5
200K	195.5K	195K	0.3
300K	309K	309.5K	0.2
430K	433K	434.5K	0.4
560K	596K	599.6K	0.6
680K	705K	709.8K	0.7
820K	901K	907.3K	0.7
910K	970K	977.8K	0.8

AN611

The accuracy of the PICMETER is dependent on the range of components being measured. If auto-ranging could be implemented, the accuracy of the PICMETER could be improved. The addition of capacitors in parallel with C2 of Figure 16 would allow auto-ranging for resistor measurements. Additional resistors in parallel with R3 would give auto-ranging capability to capacitor measurements. Figure 18 shows a simple implementation of auto-ranging given that the I/O pins are available. The R? and C? are the extra components that are added to the PICMETER circuit. These components should be optimized for a particular range of devices.

FIGURE 18: AUTO-RANGING TECHNIQUE



Another addition to the PICMETER that would increase the accuracy of components being measured is a constant current source. The source would feed into the resistor of the RC networks. This provides the same charging current to all RC networks being measured. Figure 19 shows a bilateral current source and Figure 20 shows a precision current source.

FIGURE 19: BILATERAL CURRENT SOURCE

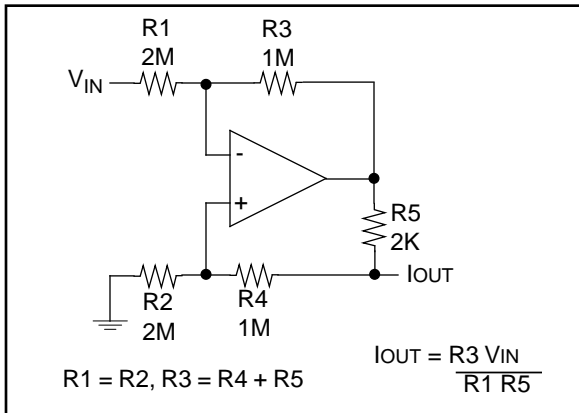
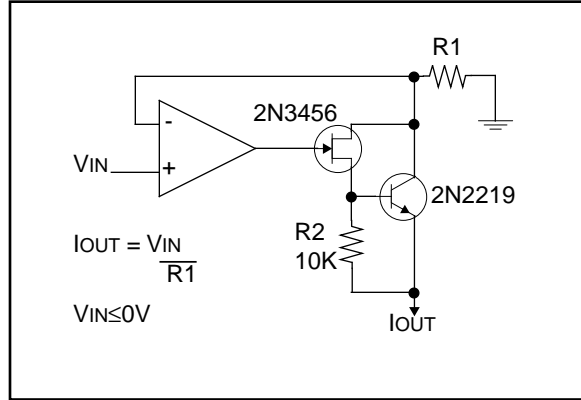
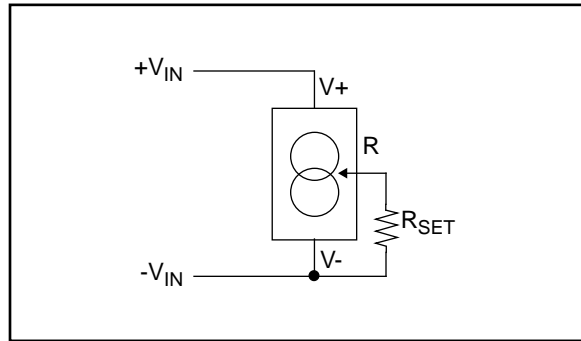


FIGURE 20: PRECISION CURRENT SOURCE



The alternative to the previous current sources is a single chip solution. A 3-terminal adjustable current source, such as a LM134/LM234/LM334 from National Semiconductor, is an ideal choice. This output current is programmable from 1 μ A to 10 mA and requires a single external resistor to set the value of current. Figure 21 shows a block diagram of the LM334Z.

FIGURE 21: LM334Z BLOCK DIAGRAM



CONCLUSION

The PIC16C62X devices add two significant analog features to the PIC16CXX mid-range family: comparators and a voltage reference. The flexibility of eight operating modes for the comparator module allows the designer to tailor the PIC16C62X device to the application. The addition of an on-chip voltage reference simplifies the design by removing at least one external component and power consumption. These analog modules coupled with the PIC16CXX mid-range family core create a new path to achieve high resolution results.

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service, page 6-3.

APPENDIX A: PICMETER FIRMWARE

MPASM 01.02.05 Intermediate PICMETER.ASM 5-1-1995 11:29:17
PICMETER Firmware for PIC16C622

PAGE 1

```

LOC OBJECT CODE    LINE SOURCE TEXT
VALUE

                                0001      TITLE "PICMETER Firmware for PIC16C622"
                                0002      LIST P = 16C622, F = INHX8M
                                0003
                                0004      INCLUDE "C:\PICMASTR\P16CXX.INC"
                                0002 ; P16CXX.INC Standard Header File, Version 0.2 Microchip Technology, Inc.
                                0004
                                0005
3FB9                          0006      FUSES _BODEN_OFF&_CP_OFF&_PWDT_ON&_WDT_OFF&_XT_OSC
                                0007
                                0008 ;*****
                                0009 ;-----*
                                0010 ;*-
                                0011 ;*-   PICMETER - Resistance and Capacitance Meter
                                0012 ;*-
                                0013 ;-----*
                                0014 ;*-
                                0015 ;*-   Author:      Rodger Richey
                                0016 ;*-
                                0017 ;*-   Filename:    picmtr.asm
                                0018 ;*-   Revision:    1 May 1995
                                0019 ;*-
                                0020 ;-----*
                                0021 ;*-
                                0022 ;*-   PICMETER is based on a PIC16C622 which has two comparators and
                                0023 ;*-   a variable voltage reference. Resistance and capacitance is
                                0024 ;*-   calculated by measuring the time constant of a RC network. The
                                0025 ;*-   toggle switch selects either resistor or capacitor input. The
                                0026 ;*-   pushbutton switch starts a measurement. The time constant of the
                                0027 ;*-   unknown component is compared to that of known component to
                                0028 ;*-   calculate the value of the unknown component. The following
                                0029 ;*-   formulas are used:
                                0030 ;*-
                                0031 ;*-   Resistance:   Ru = ( Rk * Tu ) / Tk
                                0032 ;*-   Capacitance:   Cu = ( Ck * Tu ) / Tk
                                0033 ;*-
                                0034 ;-----*
                                0035 ;*****
                                0036
                                0037
                                0038 ;*****
                                0039 ;-----*
                                0040 ;*-   RS232 code borrowed from Application Note AN593
                                0041 ;*-   "Serial Port Routines Without Using the RTCC"
                                0042 ;*-   Author: Stan D'Souza
                                0043 ;-----*
                                0044 ;*****
003D 0900                      0045 xtal    equ    .4000000
2580                          0046 baud    equ    .9600
000F 4240                      0047 fclk    equ    xtal/4
                                0048 ;*****
                                0049 ;The value baudconst must be a 8-bit value only
0020                          0050 baudconst equ    ((fclk/baud)/3-2)
                                0051 ;*****
                                0052
                                0053

```

AN611

```
0054 ;*****
0055 ;      Bit Equates
0056 ;*****
0000 0057 BEGIN   equ    0           ;begin a measurement flag
0007 0058 DONE    equ    7           ;done measuring flag
0005 0059 WHICH    equ    5           ;R or C measurement flag
0003 0060 F_ERROR equ    3           ;error detection flag
0005 0061 EMPTY    equ    5           ;flag if component is connected
0000 0062 V0      equ    0           ;power for R reference ckt
0001 0063 V1      equ    1           ;power for C reference ckt
0002 0064 V2      equ    2           ;ground for C reference ckt
0003 0065 V3      equ    3           ;power for unknown R ckt
0004 0066 V4      equ    4           ;ground for unknown C ckt
0007 0067 msb_bit equ    7           ;define for bit 7
0000 0068 lsb_bit equ    0           ;define for bit 0
0007 0069 RkHI    equ    0x07        ;value of the known resistance, R4, in ohms
009D 0070 RkMID   equ    0x9D        ;measured by a Fluke meter
0038 0071 RkLO    equ    0x38
0007 0072 CkHI    equ    0x07        ;value of the known capacitance, C1, in pF
00C8 0073 CkMID   equ    0xC8        ;measured by a Fluke meter
0030 0074 CkLO    equ    0x30
0075
0076 ;*****
0077 ;      User Registers
0078 ;*****
0079 ;      Bank 0
0020 0080 W_TEMP   equ    0x20        ;Bank 0 temporary storage for W reg
0021 0081 STATUS_TEMP equ 0x21        ;temporary storage for STATUS reg
0023 0082 Ttemp   equ    0x23        ;temporary Time register
0024 0083 flags   equ    0x24        ;flags register
0025 0084 count   equ    0x25        ;RS232 register
0026 0085 txreg   equ    0x26        ;RS232 data register
0027 0086 delay   equ    0x27        ;RS232 delay register
0028 0087 offset  equ    0x28        ;table position register
0029 0088 msb     equ    0x29        ;general delay register
002A 0089 lsb     equ    0x2A        ;general delay register
0040 0090 TimeLO  equ    0x40        ;Time registers
0041 0091 TimeMID equ    0x41
0042 0092 TimeHI  equ    0x42
0093
0094 ;      Math related registers
0050 0095 ACCaHI   equ    0x50        ;24-Bit accumulator a
0051 0096 ACCaMID equ    0x51
0052 0097 ACCaLO  equ    0x52
0053 0098 ACCbHI   equ    0x53        ;24-Bit accumulator b
0054 0099 ACCbMID equ    0x54
0055 0100 ACCbLO  equ    0x55
0056 0101 ACCcHI   equ    0x56        ;24-Bit accumulator c
0057 0102 ACCcMID equ    0x57
0058 0103 ACCcLO  equ    0x58
0059 0104 ACCdHI   equ    0x59        ;24-Bit accumulator d
005A 0105 ACCdMID equ    0x5A
005B 0106 ACCdLO  equ    0x5B
005C 0107 temp    equ    0x5C        ;temporary storage
0108
0109 ;      User Registers Bank 1
0110 ;W_TEMP   equ    0xA0        ;Bank 1 temporary storage for W reg
0111
0112 ;      User defines
0113 #define tx    PORTB,7        ;define for RS232 TXD output pin
0114
0115 ;*****
0116
0117         org    0x0
0000 2810 0118         goto   init
0119
```



```

0004 28B9      0120      org      0x4
               0121      goto     ServiceInterrupts
               0122
               0123      org      0x10
0010          0124      init
0010 1283      0125      bcf      STATUS,RP0      ;select bank 0
0011 0185      0126      clrf     PORTA          ;clear PORTA and PORTB
0012 0186      0127      clrf     PORTB
0013 1786      0128      bsf      tx              ;set TXD output pin
0014 01A4      0129      clrf     flags          ;clear flags register
0015 3010      0130      movlw   0x10           ;load table offset register
0016 00A8      0131      movwf   offset
0017 018B      0132      clrf     INTCON         ;clear interrupt flags and disable interrupts
0018 3007      0133      movlw   0x07           ;turn off comparators, mode 111
0019 009F      0134      movwf   CMCON
001A 2140      0135      call    delay20        ;wait for comarators to settle
001B 089F      0136      movf    CMCON,F
001C 130C      0137      bcf      PIR1,CMIF
001D 1683      0138      bsf      STATUS,RP0    ;select bank 1
001E 3088      0139      movlw   0x88           ;WDT prescalar,internal TMR0 increment
001F 0081      0140      movwf   OPTION_REG
0020 0185      0141      clrf     TRISA          ;PORTA all outputs, discharges RC ckts
0021 3060      0142      movlw   0x60           ;PORTA<7,4:0> outputs, PORTA<6:5> inputs
0022 0086      0143      movwf   TRISB
0023 300C      0144      movlw   0x0C           ;setup Voltage Reference
0024 009F      0145      movwf   VRCON
0025 1283      0146      bcf      STATUS,RP0    ;select bank 0
0026 3008      0147      movlw   0x08           ;enable RBIE interrupt
0027 008B      0148      movwf   INTCON
0028 213D      0149      call    vlong          ;delay before transmitting boot message
0029 213D      0150      call    vlong          ;to allow computer program to setup
002A 213D      0151      call    vlong
002B 2131      0152      call    BootMSG        ;transmit boot message
002C 178B      0153      bsf      INTCON,GIE     ;enable global interrupt bit
               0154
002D          0155      start
002D 1C24      0156      btfss   flags,BEGIN    ;wait for a start measurement key press
002E 282D      0157      goto    start
002F 1024      0158      bcf      flags,BEGIN    ;clear start measurement flag
               0159
0030 138B      0160      bcf      INTCON,GIE     ;transmit a start measurement message
0031 3053      0161      movlw   'S'            ;to the PC
0032 20AD      0162      call    Send
0033 178B      0163      bsf      INTCON,GIE
               0164
0034 01C2      0165      clrf     TimeHI         ;reset Time registers
0035 01C1      0166      clrf     TimeMID
0036 01C0      0167      clrf     TimeLO
0037 1E86      0168      btfss   PORTB,WHICH    ;detect if resistor or capacitor measure
0038 2862      0169      goto    Capacitor
               0170
0039          0171      Resistor
0039 1683      0172      bsf      STATUS,RP0    ;set V0 to input
003A 1406      0173      bsf      TRISB,V0
003B 1283      0174      bcf      STATUS,RP0
003C 20FB      0175      call    AnalogOn       ;turn analog on
003D 0181      0176      clrf     TMR0
003E 0000      0177      nop
003F 1586      0178      bsf      PORTB,V3       ;turn power on to unknown RC ckt
0040 19A4      0179      RwaitU  btfsc   flags,F_ERROR  ;detect if an error occurs
0041 288B      0180      goto    ErrorDetect
0042 1FA4      0181      btfss   flags,DONE     ;measurement completed flag
0043 2840      0182      goto    RwaitU
0044 13A4      0183      bcf      flags,DONE     ;clear measurement completed flag
0045 2111      0184      call    AnalogOff      ;turn analog off
               0185

```

AN611

```
0046 2126          0186      call    SwapTtoA      ;swap Time to accumulator a
0047 3007          0187      movlw   RkHI          ;swap known resistance value
0048 00D3          0188      movwf  ACCbHI        ;to accumulator b
0049 309D          0189      movlw   RkMID
004A 00D4          0190      movwf  ACCbMID
004B 3038          0191      movlw   RkLO
004C 00D5          0192      movwf  ACCbLO
004D 2230          0193      call    Mpy24         ;multiply accumulator a and b
                   0194
004E 1683          0195      bsf    STATUS,RP0     ;set V3 to input
004F 1586          0196      bsf    TRISB,V3
0050 1283          0197      bcf    STATUS,RP0
0051 20FB          0198      call    AnalogOn     ;turn analog on
0052 0181          0199      clrf   TMR0
0053 0000          0200      nop
0054 1406          0201      bsf    PORTB,V0      ;turn power on to known RC ckt
0055 19A4          0202 RwaitK  btfsc  flags,F_ERROR  ;detect if an error occurs
0056 288B          0203      goto   ErrorDetect
0057 1FA4          0204      btfss  flags,DONE    ;measurement completed flag

0058 2855          0205      goto   RwaitK
0059 13A4          0206      bcf    flags,DONE    ;clear measurement completed flag
005A 2111          0207      call   AnalogOff     ;turn analog off
                   0208
005B 2126          0209      call   SwapTtoA     ;swap Time to accumulator a
005C 224B          0210      call   Div24         ;divide multiply by known time
                   0211
005D 138B          0212      bcf    INTCON,GIE    ;disable all interrupts
005E 3052          0213      movlw  'R'           ;transmit, for R measurement
005F 20AD          0214      call   Send
0060 178B          0215      bsf    INTCON,GIE    ;enable global interrupt bit
0061 282D          0216      goto   start         ;restart
                   0217
0062              0218 Capacitor
0062 1683          0219      bsf    STATUS,RP0     ;set V2 to input
0063 1506          0220      bsf    TRISB,V2
0064 1283          0221      bcf    STATUS,RP0
0065 20FB          0222      call   AnalogOn     ;turn analog on
0066 0181          0223      clrf   TMR0
0067 0000          0224      nop
0068 1486          0225      bsf    PORTB,V1      ;turn power on to unknown RC ckt
0069 19A4          0226 CwaitU  btfsc  flags,F_ERROR  ;detect if an error occurs
006A 288B          0227      goto   ErrorDetect
006B 1FA4          0228      btfss  flags,DONE    ;measurement completed flag
006C 2869          0229      goto   CwaitU
006D 13A4          0230      bcf    flags,DONE    ;clear measurement completed flag
006E 2111          0231      call   AnalogOff     ;turn analog off
                   0232
006F 2126          0233      call   SwapTtoA     ;swap Time to accumulator a
0070 3007          0234      movlw  CkHI          ;swap known resistance value
0071 00D3          0235      movwf  ACCbHI        ;to accumulator b
0072 30C8          0236      movlw  CkMID
0073 00D4          0237      movwf  ACCbMID
0074 3030          0238      movlw  CkLO
0075 00D5          0239      movwf  ACCbLO
0076 2230          0240      call   Mpy24         ;multiply accumulator a and b
                   0241
0077 1683          0242      bsf    STATUS,RP0     ;set V3 to input
0078 1606          0243      bsf    TRISB,V4
0079 1283          0244      bcf    STATUS,RP0
007A 20FB          0245      call   AnalogOn     ;turn analog on
007B 0181          0246      clrf   TMR0
007C 0000          0247      nop
007D 1486          0248      bsf    PORTB,V1      ;turn power on to known RC ckt
007E 19A4          0249 CwaitK  btfsc  flags,F_ERROR  ;detect if an error occurs
007F 288B          0250      goto   ErrorDetect
```

```

0080 1FA4      0251      btfss   flags,DONE      ;measurement completed flag
0081 287E      0252      goto    CwaitK
0082 13A4      0253      bcf     flags,DONE      ;clear measurement completed flag
0083 2111      0254      call    AnalogOff       ;turn analog off
                                0255
0084 2126      0256      call    SwapTtoA        ;swap Time to accumulator a
0085 224B      0257      call    Div24           ;divide multiply by known time
                                0258
0086 138B      0259      bcf     INTCON,GIE      ;disable all interrupts
0087 3043      0260      movlw   'C'            ;transmit, for C measurement
0088 20AD      0261      call    Send
0089 178B      0262      bsf     INTCON,GIE      ;enable global interrupt bit
008A 282D      0263      goto    start           ;restart
                                0264
008B          0265      ErrorDetect
008B 1283      0266      bcf     STATUS,RP0      ;disable TMR0
008C 128B      0267      bcf     INTCON,T0IE
008D 110B      0268      bcf     INTCON,T0IF
008E 2111      0269      call    AnalogOff       ;turn analog off
008F 11A4      0270      bcf     flags,F_ERROR   ;clear error flag
                                0271
0090 138B      0272      bcf     INTCON,GIE      ;disable all interrupts
0091 3045      0273      movlw   'E'            ;transmit, for C measurement
0092 20AD      0274      call    Send
0093 178B      0275      bsf     INTCON,GIE      ;enable global interrupt bit
0094 282D      0276      goto    start           ;restart
                                0277
0278 ;*****
0279 ;-----*
0280 ;*-      RS232 Transmit Routine
0281 ;*-      Borrowed from AN593, "Serial Port Routines Without Using the R1CC"
0282 ;*-      Author: Stan D'Souza
0283 ;*-      This is the routine that interfaces directly to the hardware
0284 ;-----*
0285 ;*****
0095          0286      Transmit
0095 1283      0287      bcf     STATUS,RP0
0096 00A6      0288      movwf   txreg
0097 1386      0289      bcf     tx              ;send start bit
0098 3020      0290      movlw   baudconst
0099 00A7      0291      movwf   delay
009A 3009      0292      movlw   0x9
009B 00A5      0293      movwf   count
009C          0294      txbaudwait
009C 0BA7      0295      decfsz delay
009D 289C      0296      goto    txbaudwait
009E 3020      0297      movlw   baudconst
009F 00A7      0298      movwf   delay
00A0 0BA5      0299      decfsz count
00A1 28A6      0300      goto    SendNextBit
00A2 3009      0301      movlw   0x9
00A3 00A5      0302      movwf   count
00A4 1786      0303      bsf     tx              ;send stop bit
00A5 0008      0304      return
00A6          0305      SendNextBit
00A6 0CA6      0306      rrf     txreg
00A7 1C03      0307      btfss   STATUS,C
00A8 28AB      0308      goto    Setlo
00A9 1786      0309      bsf     tx
00AA 289C      0310      goto    txbaudwait
00AB 1386      0311      Setlo   bcf     tx
00AC 289C      0312      goto    txbaudwait
                                0313 ;-----
0314
0315 ;*****
0316 ;-----*

```

AN611

```
0317 ;*-      Generic Transmit Routine
0318 ;*-      Sends what is currently in the W register and accumulator ACCc
0319 ;-----*
0320 ;*****
00AD      0321 Send
00AD 2095      0322      call    Transmit
00AE 2146      0323      call    delay1      ;delay between bytes
00AF 0856      0324      movf   ACCcHI,W      ;transmit high resistance byte
00B0 2095      0325      call    Transmit
00B1 2146      0326      call    delay1      ;delay between bytes
00B2 0857      0327      movf   ACCcMID,W      ;transmit mid resistance byte
00B3 2095      0328      call    Transmit
00B4 2146      0329      call    delay1      ;delay between bytes
00B5 0858      0330      movf   ACCcLO,W      ;transmit low resistance byte
00B6 2095      0331      call    Transmit
00B7 2146      0332      call    delay1      ;delay between bytes
00B8 0008      0333      return
0334 ;-----*
0335 ;*****
0336 ;-----*
0337 ;*-      Interrupt Service Routines
0338 ;-----*
0339 ;*****
00B9      0341 ServiceInterrupts
00B9 00A0      0342      movwf  W_TEMP      ;Pseudo push instructions
00BA 0E03      0343      swapf STATUS,W
00BB 1283      0344      bcf    STATUS,RP0
00BC 00A1      0345      movwf  STATUS_TEMP
0346
00BD 0801      0347      movf   TMR0,W
00BE 00A3      0348      movwf  Ttemp
00BF 190B      0349      btfs   INTCON,T0IF      ;Service Timer 0 overflow
00C0 20E5      0350      call  ServiceTimer
00C1 1B0C      0351      btfs   PIR1,CMIF      ;Stops Timer0, Records Value
00C2 20EC      0352      call  ServiceComparator
00C3 180B      0353      btfs   INTCON,RBIF      ;Service pushbutton switch
00C4 20CB      0354      call  ServiceKeystroke ;Starts a measurement
0355
00C5 1283      0356      bcf    STATUS,RP0
00C6 0E21      0357      swapf STATUS_TEMP,W      ;Pseudo pop instructions
00C7 0083      0358      movwf  STATUS
00C8 0EA0      0359      swapf  W_TEMP,F
00C9 0E20      0360      swapf  W_TEMP,W
0361
00CA 0009      0362      retfie
0363 ;-----*
0364 ;*****
0365 ;-----*
0366 ;*-      Borrowed from AN552, "Implementing Wake-up on Key Stroke"
0367 ;*-      Author: Stan D'Souza
0368 ;-----*
0369 ;*****
00CB      0371 ServiceKeystroke
00CB 118B      0372      bcf    INTCON,RBIE      ;disable interrupt
00CC 0906      0373      comf  PORTB,W      ;read PORTB
00CD 100B      0374      bcf    INTCON,RBIF      ;clear interrupt flag
00CE 3940      0375      andlw B'01000000'
00CF 1903      0376      btfs   STATUS,Z
00D0 28D6      0377      goto  NotSwitch
00D1 2143      0378      call  delay16      ;de-bounce switch for 16msec
00D2 0906      0379      comf  PORTB,W      ;read PORTB again
00D3 20D9      0380      call  KeyRelease      ;check for key release
00D4 1424      0381      bsf   flags,BEGIN
00D5 0008      0382      return
```

```

0383
00D6      0384 NotSwitch      ;detected other PORTB pin change
00D6 100B      0385      bcf      INTCON,RBIF      ;reset RBI interrupt
00D7 158B      0386      bsf      INTCON,RBIE
00D8 0008      0387      return
0388
00D9      0389 KeyRelease
00D9 2143      0390      call     delay16      ;debounce switch
00DA 0906      0391      comf    PORTB,W      ;read PORTB
00DB 100B      0392      bcf      INTCON,RBIF      ;clear flag
00DC 158B      0393      bsf      INTCON,RBIE      ;enable interrupt
00DD 3940      0394      andlw   B'01000000'
00DE 1903      0395      btfsc   STATUS,Z      ;key still pressed?
00DF 0008      0396      return      ;if no, then return
00E0 0063      0397      sleep      ;else, save power
00E1 118B      0398      bcf      INTCON,RBIE      ;disable interrupts
00E2 0906      0399      comf    PORTB,W      ;read PORTB
00E3 100B      0400      bcf      INTCON,RBIF      ;clear flag
00E4 28D9      0401      goto    KeyRelease      ;try again
0402 ;
0403
0404 ;*****
0405 ;-----*
0406 ;*-      ISR to service a Timer0 overflow
0407 ;-----*
0408 ;*****
00E5      0409 ServiceTimer
00E5 0AC1      0410      incf    TimeMID,F      ;increment middle Time byte
00E6 1903      0411      btfsc   STATUS,Z      ;if middle overflows,
00E7 0AC2      0412      incf    TimeHI,F      ;increment high Time byte
00E8 1AC2      0413      btfsc   TimeHI,EMPTY      ;check if component is connected
00E9 15A4      0414      bsf     flags,F_ERROR      ;set error flag
00EA 110B      0415      bcf     INTCON,T0IF      ;clear TMR0 interrupt flag
00EB 0008      0416      return
0417 ;
0418
0419 ;*****
0420 ;-----*
0421 ;*-      ISR to service a Comparator interrupt
0422 ;-----*
0423 ;*****
00EC      0424 ServiceComparator
00EC 1283      0425      bcf     STATUS,RP0      ;select bank 0
00ED 1E86      0426      btfss   PORTB,WHICH      ;detect which measurement, R or C?
00EE 28F2      0427      goto    capcomp
00EF 1F1F      0428      btfss   CMCON,C1OUT      ;detect if R ckt has interrupted
00F0 28F4      0429      goto    scstop
00F1 28F9      0430      goto    scend
00F2      0431 capcomp
00F2 1B9F      0432      btfsc   CMCON,C2OUT      ;detect if C ckt has interrupted
00F3 28F9      0433      goto    scend
00F4      0434 scstop
00F4 128B      0435      bcf     INTCON,T0IE      ;disable TMR0 interrupts
00F5 110B      0436      bcf     INTCON,T0IF
00F6 0823      0437      movf    Ttemp,W
00F7 00C0      0438      movwf   TimeLO
00F8 17A4      0439      bsf     flags,DONE      ;set DONE flag
00F9      0440 scend
00F9 130C      0441      bcf     PIR1,CMIF      ;clear comparator interrupt flag
00FA 0008      0442      return
0443 ;
0444
0445 ;*****
0446 ;-----*
0447 ;*-      Turn Comparators and Vref On
0448 ;-----*

```

AN611

```
0449 ;*****
00FB      0450 AnalogOn
00FB 1283 0451      bcf      STATUS,RP0      ;select bank 0
00FC 3002 0452      movlw   0x02      ;turn comparators on, mode 010
00FD 009F 0453      movwf  CMCON      ;4 inputs multiplexed to 2 comparators
00FE 1683 0454      bsf      STATUS,RP0      ;select bank 1
00FF 300F 0455      movlw   0x0F      ;make PORTA<3:0> all inputs
0100 0085 0456      movwf  TRISA
0101 179F 0457      bsf      VRCON,VREN
0102 1283 0458      bcf      STATUS,RP0      ;select bank 0
0103 2140 0459      call   delay20      ;20msec delay
0104 089F 0460      movf   CMCON,F      ;clear comparator mismatch condition
0105 130C 0461      bcf      PIR1,CMIF      ;clear comparator interrupt flag
0106 1683 0462      bsf      STATUS,RP0
0107 170C 0463      bsf      PIE1,CMIE      ;enable comparator interrupts
0108 1283 0464      bcf      STATUS,RP0
0109 170B 0465      bsf      INTCON,PEIE      ;enable peripheral interrupts
010A 11A4 0466      bcf      flags,F_ERROR
010B 0181 0467      clrf   TMR0      ;clear TMR0 counter
010C 0000 0468      nop
010D 0000 0469      nop
010E 110B 0470      bcf      INTCON,T0IF      ;clear TMR0 interrupt flag
010F 168B 0471      bsf      INTCON,T0IE      ;enable TMR0 interrupts
0110 0008 0472      return
0473 ;
0474
0475 ;*****
0476 ;*-----*
0477 ;*-      Turn Comparators and Vref Off
0478 ;*-----*
0479 ;*****
0111      0480 AnalogOff
0111 1283 0481      bcf      STATUS,RP0
0112 130B 0482      bcf      INTCON,PEIE
0113 3080 0483      movlw   0x80      ;reset PORTB value
0114 0086 0484      movwf  PORTB
0115 1683 0485      bsf      STATUS,RP0      ;select bank 1
0116 130C 0486      bcf      PIE1,CMIE      ;disable comparator interrupts
0117 0185 0487      clrf   TRISA      ;set PORTA pins to outputs, discharge RC ckt
0118 3060 0488      movlw   0x60      ;set PORTB 7,4-0 as outputs, 6,5 as inputs
0119 0086 0489      movwf  TRISB
011A 139F 0490      bcf      VRCON,VREN      ;disable Vref
011B 1283 0491      bcf      STATUS,RP0      ;select bank 0
011C 3007 0492      movlw   0x07
011D 009F 0493      movwf  CMCON      ;disable comparators
011E 2140 0494      call   delay20      ;20msec delay
011F 089F 0495      movf   CMCON,F      ;clear comparator mismatch condition
0120 130C 0496      bcf      PIR1,CMIF      ;clear comparator interrupt flag
0121 110B 0497      bcf      INTCON,T0IF      ;clear Timer0 interrupt flag
0122 213D 0498      call   vlong      ;long delay to allow capacitors to discharge
0123 213D 0499      call   vlong
0124 213D 0500      call   vlong
0125 0008 0501      return
0502 ;
0503
0504 ;*****
0505 ;*-----*
0506 ;*-      Swap Time to Accumulator a
0507 ;*-----*
0508 ;*****
0126      0509 SwapTtoA
0126 1283 0510      bcf      STATUS,RP0
0127 0842 0511      movf   TimeHI,W
0128 00D0 0512      movwf  ACCaHI
0129 0841 0513      movf   TimeMID,W
012A 00D1 0514      movwf  ACCaMID
```

```

012B 0840          0515      movf    TimeLO,W
012C 00D2          0516      movwf   ACCaLO
012D 01C2          0517      clrf   TimeHI
012E 01C1          0518      clrf   TimeMID
012F 01C0          0519      clrf   TimeLO
0130 0008          0520      return
0521 ; _____
0522
0523 ;*****
0524 ;*-----*
0525 ;*-      Transmit the Boot Message
0526 ;*-----*
0527 ;*****

0131              0528 BootMSG
0131 1283          0529      bcf    STATUS,RP0      ;select bank 0
0132 3002          0530 msg   movlw  HIGH Table   ;init the PCH for a table call
0133 008A          0531      movwf  PCLATH
0134 0828          0532      movf   offset,W       ;move table offset into W
0135 2200          0533      call   Table          ;get table value
0136 2095          0534      call   Transmit       ;transmit table value
0137 2146          0535      call   delay1         ;delay between bytes
0138 0BA8          0536      decfsz offset,F      ;check for end of table
0139 2932          0537      goto   msg
013A 3010          0538      movlw  0x10          ;reset table offset
013B 00A8          0539      movwf  offset
013C 0008          0540      return
0541 ; _____
0542
0543 ;*****
0544 ;*-----*
0545 ;*-      Delay Routines
0546 ;*-----*
0547 ;*****

013D 30FF          0548 vlong  movlw  0xff           ;very long delay, approx 200msec
013E 00A9          0549      movwf  msb
013F 2948          0550      goto   d1
0140              0551 delay20                ;20 msec delay
0140 301A          0552      movlw  .26
0141 00A9          0553      movwf  msb
0142 2948          0554      goto   d1
0143              0555 delay16                ;16 msec delay
0143 3015          0556      movlw  .21
0144 00A9          0557      movwf  msb
0145 2948          0558      goto   d1
0146              0559 delay1                ;approx 750nsec delay
0146 3001          0560      movlw  .1
0147 00A9          0561      movwf  msb
0148 30FF          0562 d1    movlw  0xff
0149 00AA          0563      movwf  lsb
014A 0BAA          0564 d2    decfsz lsb,F
014B 294A          0565      goto   d2
014C 0BA9          0566      decfsz msb,F
014D 2948          0567      goto   d1
014E 0008          0568      return
0569 ; _____
0570
0571
0572      org    0x200
0573
0574
0575 ;*****
0576 ;*-----*
0577 ;*-      Table for Boot Message
0578 ;*-----*
0579 ;*****

0200              0580 Table                ;boot message "PICMETER Booted!"

```

AN611

```
0200 0782          0581      addwf   PCL           ;add W to PCL
0201 3400          0582      retlw   0
0202 3421          0583      retlw   '!'
0203 3464          0584      retlw   'd'
0204 3465          0585      retlw   'e'
0205 3474          0586      retlw   't'
0206 346F          0587      retlw   'o'
0207 346F          0588      retlw   'o'
0208 3442          0589      retlw   'B'
0209 3420          0590      retlw   ' '
020A 3472          0591      retlw   'r'
020B 3465          0592      retlw   'e'
020C 3474          0593      retlw   't'
020D 3465          0594      retlw   'e'
020E 346D          0595      retlw   'm'
020F 3443          0596      retlw   'C'
0210 3449          0597      retlw   'I'
0211 3450          0598      retlw   'P'
0599 ; _____
0600
0601 ;*****
0602 ;*-----*
0603 ;*-      24-bit Addition
0604 ;*-
0605 ;*-      Uses ACCa and ACCb
0606 ;*-
0607 ;*-      ACCa + ACCb -> ACCb
0608 ;*-----*
0609 ;*****
0212          0610 Add24
0212 0852          0611      movf   ACCaLO,W
0213 07D5          0612      addwf  ACCbLO           ;add low bytes
0214 1803          0613      btfsc STATUS,C         ;add in carry if necessary
0215 2A1D          0614      goto  A2
0216 0851          0615 A1      movf   ACCaMID,W
0217 07D4          0616      addwf  ACCbMID         ;add mid bytes
0218 1803          0617      btfsc STATUS,C         ;add in carry if necessary
0219 0AD3          0618      incf  ACCbHI
021A 0850          0619      movf  ACCaHI,W
021B 07D3          0620      addwf  ACCbHI         ;add high bytes
021C 3400          0621      retlw  0
021D 0AD4          0622 A2      incf  ACCbMID
021E 1903          0623      btfsc STATUS,Z
021F 0AD3          0624      incf  ACCbHI
0220 2A16          0625      goto  A1
0626 ; _____
0627
0628 ;*****
0629 ;*-----*
0630 ;*-      Subtraction ( 24 - 24 -> 24 )
0631 ;*-
0632 ;*-      Uses ACCa, ACCb, ACCd
0633 ;*-
0634 ;*-      ACCa -> ACCd,
0635 ;*-      2's complement ACCa,
0636 ;*-      call Add24 ( ACCa + ACCb -> ACCb ),
0637 ;*-      ACCd -> ACCa
0638 ;*-----*
0639 ;*****
0221          0640 Sub24
0221 0850          0641      movf  ACCaHI,W         ;Transfer ACCa to ACCd
0222 00D9          0642      movwf ACCdHI
0223 0851          0643      movf  ACCaMID,W
0224 00DA          0644      movwf ACCdMID
0225 0852          0645      movf  ACCaLO,W
0226 00DB          0646      movwf ACCdLO
```



```

0227 2275      0647      call    compA          ;2's complement ACCa
0228 2212      0648      call    Add24          ;Add ACCa to ACCb
0229 0859      0649      movf   ACCdHI,W       ;Transfer ACCd to ACCa
022A 00D0      0650      movwf  ACCaHI
022B 085A      0651      movf   ACCdMID,W
022C 00D1      0652      movwf  ACCaMID
022D 085B      0653      movf   ACCdLO,W
022E 00D2      0654      movwf  ACCaLO
022F 3400      0655      retlw  0
0656 ; _____
0657
0658 ;*****
0659 ;-----*
0660 ;*-      Multiply ( 24 X 24 -> 56 )
0661 ;*-
0662 ;*-      Uses ACCa, ACCb, ACCc, ACCd
0663 ;*-
0664 ;*-      ACCa * ACCb -> ACCb,ACCc 56-bit output
0665 ;*-      with ACCb (ACCbHI,ACCbMID,ACCbLO) with 24 msb's and
0666 ;*-      ACCc (ACCcHI,ACCcMID,ACCcLO) with 24 lsb's
0667 ;-----*
0668 ;*****
0230      0669 Mpy24
0230 223F      0670      call    Msetup
0231 0CD9      0671 mloop  rrf    ACCdHI          ;rotate d right
0232 0CDA      0672      rrf    ACCdMID
0233 0CDB      0673      rrf    ACCdLO
0234 1803      0674      btfsc  STATUS,C       ;need to add?
0235 2212      0675      call    Add24
0236 0CD3      0676      rrf    ACCbHI
0237 0CD4      0677      rrf    ACCbMID
0238 0CD5      0678      rrf    ACCbLO
0239 0CD6      0679      rrf    ACCcHI
023A 0CD7      0680      rrf    ACCcMID
023B 0CD8      0681      rrf    ACCcLO
023C 0BDC      0682      decfsz temp          ;loop until all bits checked
023D 2A31      0683      goto   mloop
023E 3400      0684      retlw  0
0685
023F      0686 Msetup
023F 3018      0687      movlw  0x18          ;for 24 bit shifts
0240 00DC      0688      movwf  temp
0241 0853      0689      movf   ACCbHI,W       ;move ACCb to ACCd
0242 00D9      0690      movwf  ACCdHI
0243 0854      0691      movf   ACCbMID,W
0244 00DA      0692      movwf  ACCdMID
0245 0855      0693      movf   ACCbLO,W
0246 00DB      0694      movwf  ACCdLO
0247 01D3      0695      clrf   ACCbHI
0248 01D4      0696      clrf   ACCbMID
0249 01D5      0697      clrf   ACCbLO
024A 3400      0698      retlw  0
0699 ; _____
0700
0701 ;*****
0702 ;-----*
0703 ;*-      Division ( 56 / 24 -> 24 )
0704 ;*-
0705 ;*-      Uses ACCa, ACCb, ACCc, ACCd
0706 ;*-
0707 ;*-      56-bit dividend in ACCb,ACCc ( ACCb has msb's and ACCc has lsb's)
0708 ;*-      24-bit divisor in ACCa
0709 ;*-      quotient is stored in ACCc
0710 ;*-      remainder is stored in ACCb
0711 ;-----*
0712 ;*****

```

AN611

```
024B          0713 Div24
024B 2272     0714          call   Dsetup
              0715
024C 1003     0716 dloop   bcf    STATUS,C
024D ODD8     0717          rlf    ACCcLO          ;Rotate dividend left 1 bit position
024E ODD7     0718          rlf    ACCcMID
024F ODD6     0719          rlf    ACCcHI
0250 ODD5     0720          rlf    ACCbLO
0251 ODD4     0721          rlf    ACCbMID
0252 ODD3     0722          rlf    ACCbHI
              0723
0253 1803     0724          btfsc  STATUS,C          ;invert carry and exclusive or with the
0254 2A58     0725          goto   clear          ;msb of the divisor then move this bit
0255 1FD0     0726          btfss  ACCaHI,msb_bit ;into the lsb of the dividend
0256 0AD8     0727          incf  ACCcLO
0257 2A5A     0728          goto   cont
0258 1BD0     0729 clear   btfsc  ACCaHI,msb_bit
0259 0AD8     0730          incf  ACCcLO
              0731
025A 1858     0732 cont   btfsc  ACCcLO,lsb_bit ;check the lsb of the dividend
025B 2A5E     0733          goto   minus
025C 2212     0734          call   Add24          ;if = 0, then add divisor to upper 24 bits
025D 2A5F     0735          goto   check          ;of dividend
025E 2221     0736 minus  call   Sub24          ;if = 1, then subtract divisor from upper
              0737          ;24 bits of dividend
              0738
025F 0BDC     0739 check  decfsz temp,f          ;do 24 times
0260 2A4C     0740          goto   dloop
              0741
0261 1003     0742          bcf    STATUS,C
0262 ODD8     0743          rlf    ACCcLO          ;shift lower 24 bits of dividend 1 bit
0263 ODD7     0744          rlf    ACCcMID          ;position left
0264 ODD6     0745          rlf    ACCcHI
0265 1BD3     0746          btfsc  ACCbHI,msb_bit ;exclusive or the inverse of the msb of the
0266 2A6A     0747          goto   w1             ;dividend with the msb of the divisor
0267 1FD0     0748          btfss  ACCaHI,msb_bit ;store in the lsb of the dividend
0268 0AD8     0749          incf  ACCcLO
0269 2A6C     0750          goto   wzd
026A 1BD0     0751 w1     btfsc  ACCaHI,msb_bit
026B 0AD8     0752          incf  ACCcLO
026C 1FD3     0753 wzd   btfss  ACCbHI,msb_bit ;if the msb of the remainder is set and
026D 2A71     0754          goto   wend
026E 1BD0     0755          btfsc  ACCaHI,msb_bit ;the msb of the divisor is not
026F 2A71     0756          goto   wend
0270 2212     0757          call   Add24          ;add the divisor to the remainder to correct
              0758          ;for zero partial remainder
              0759
0271 3400     0760 wend   retlw  0          ;quotient in 24 lsb's of dividend
              0761          ;remainder in 24 msb's of dividend
              0762
0272          0763 Dsetup
0272 3018     0764          movlw  0x18          ;loop 24 times
0273 00DC     0765          movwf  temp
              0766
0274 3400     0767          retlw  0
              0768 ;
              0769
0277          0770 ;*****
0277          0771 ;*-----*
0277          0772 ;*-      2's Complement
0277          0773 ;*-
0277          0774 ;*-      Uses ACCa
0277          0775 ;*-
0277          0776 ;*-      Performs 2's complement conversion on ACCa
0277          0777 ;*-----*
0277          0778 ;*****
```

```

0275          0779 compA
0275 09D2     0780      comf  ACCaLO      ;invert all bits in accumulator a
0276 09D1     0781      comf  ACCaMID
0277 09D0     0782      comf  ACCaHI
0278 0AD2     0783      incf  ACCaLO      ;add one to accumulator a
0279 1903     0784      btfsc STATUS,Z
027A 0AD1     0785      incf  ACCaMID
027B 1903     0786      btfsc STATUS,Z
027C 0AD0     0787      incf  ACCaHI
027D 3400     0788      retlw 0
0789 ; _____
0790
0791          END
0792

```

```

0000 : X---X----- XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX

0100 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXXXXXX- -----

0200 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX--

```

All other memory blocks unused.

```

Errors   :    0
Warnings :    0
Messages :    0

```

AN611

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service, page 6-3.

APPENDIX B: VISUAL BASIC PROGRAM

PICMTR.FRM

```
Sub Form_Load ()
    'Initialize the program
    Image1.Height = 600
    Image1.Width = 2700
    Frame1.Caption = "PICMETER Power Off"
    Label1.Caption = ""
    Label2.Caption = ""

    'Initialize Comm Port 1
    Comm1.RThreshold = 1
    Comm1.Handshaking = 0
    Comm1.Settings = "9600,n,8,1"
    Comm1.CommPort = 2
    Comm1.PortOpen = True

    'Initialize the global variable First%
    First% = 0
End Sub

Sub Form_Unload (Cancel As Integer)
    'Unload PICMETER
    Comm1.RTSEnable = False
    Comm1.DTREnable = False
    Comm1.PortOpen = False
    Unload PICMETER
End Sub

Sub Comm1_OnComm ()
    Dim Value As Double
    Dim High As Double
    Dim Medium As Double
    Dim Low As Double

    'Received a character
    If Comm1.CommEvent = 2 Then
        If First% = 0 Then
            If Comm1.InBufferCount = 16 Then
                Label1.FontSize = 10
                InString$ = Comm1.Input
                If InString$ = "PICMETER Booted!" Then
                    Frame1.Caption = "PICMETER Booted!"
                End If
                First% = 1
                Comm1.InputLen = 4
            End If
        Else
            If Comm1.InBufferCount >= 4 Then
                InString$ = Comm1.Input
                If Left$(InString$, 1) = "R" Then
                    Frame1.Caption = "Resistance"
                    Label2.FontName = "Symbol"
                    Label2.Caption = "KW"
                    Label1.FontSize = 24
                ElseIf Left$(InString$, 1) = "C" Then
                    Frame1.Caption = "Capacitance"
                    Label2.FontName = "MS Sans Serif"
                    Label2.Caption = "nF"
                    Label1.FontSize = 24
                ElseIf Left$(InString$, 1) = "E" Then
                    Frame1.Caption = "Error Detected"
                    Label2.Caption = ""
                ElseIf Left$(InString$, 1) = "S" Then
                    Frame1.Caption = "Measuring Component"
                End If
            End If
        End If
    End If
End Sub
```

```

        Label2.Caption = ""
    Else
        Frame1.Caption = "Error Detected"
        Label2.Caption = ""
    End If

    If Frame1.Caption = "Error Detected" Then
        Label1.Caption = ""
    ElseIf Frame1.Caption = "Measuring Component" Then
        Label1.Caption = ""
    Else
        High = 65536# * Asc(Mid$(InString$, 2, 1))
        Medium = 256# * Asc(Mid$(InString$, 3, 1))
        Low = Asc(Mid$(InString$, 4, 1))
        Label1.Caption = Format$(High + Medium + Low) / 1000, "###0.0")
    End If
End If
End If
End If
End Sub

Sub Check3D1_Click (Value As Integer)
    'Control Power to the PICMETER
    If Check3D1.Value = False Then
        Comml.InputLen = 0
        Label1.Caption = ""
        Label2.Caption = ""
        Comml.RTSEnable = False
        Comml.DTREnable = False
        Frame1.Caption = "PICMETER Power Off"
        InString$ = Comml.Input
    Else
        Frame1.Caption = ""
        First% = 0
        Comml.InputLen = 0
        InString$ = Comml.Input
        Comml.RTSEnable = True
        Comml.DTREnable = True
    End If
End Sub

Sub menExitTop_Click ()
    'Unload PICMETER
    Unload PICMETER
End Sub

Sub Option1_Click ()
    'Open COM1 for communications
    If Option1.Value = True Then
        If Comml.CommPort = 2 Then
            Comml.PortOpen = False
            Comml.CommPort = 1
            Comml.PortOpen = True
        End If
    End If
End Sub

Sub Option2_Click ()
    'Open COM2 for communications
    If Option2.Value = True Then
        If Comml.CommPort = 1 Then
            Comml.PortOpen = False
            Comml.CommPort = 2
            Comml.PortOpen = True
        End If
    End If
End Sub

```

PICMETER.BAS

```

Global I%
Global First%

```

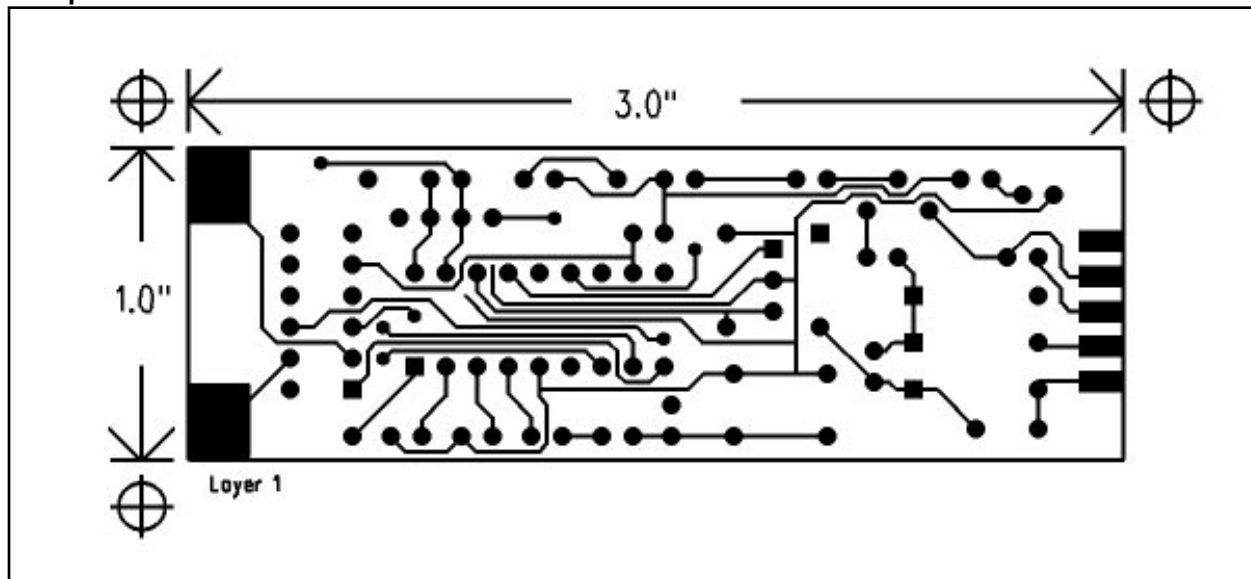
AN611

APPENDIX C: PICMETER PCB LAYOUT

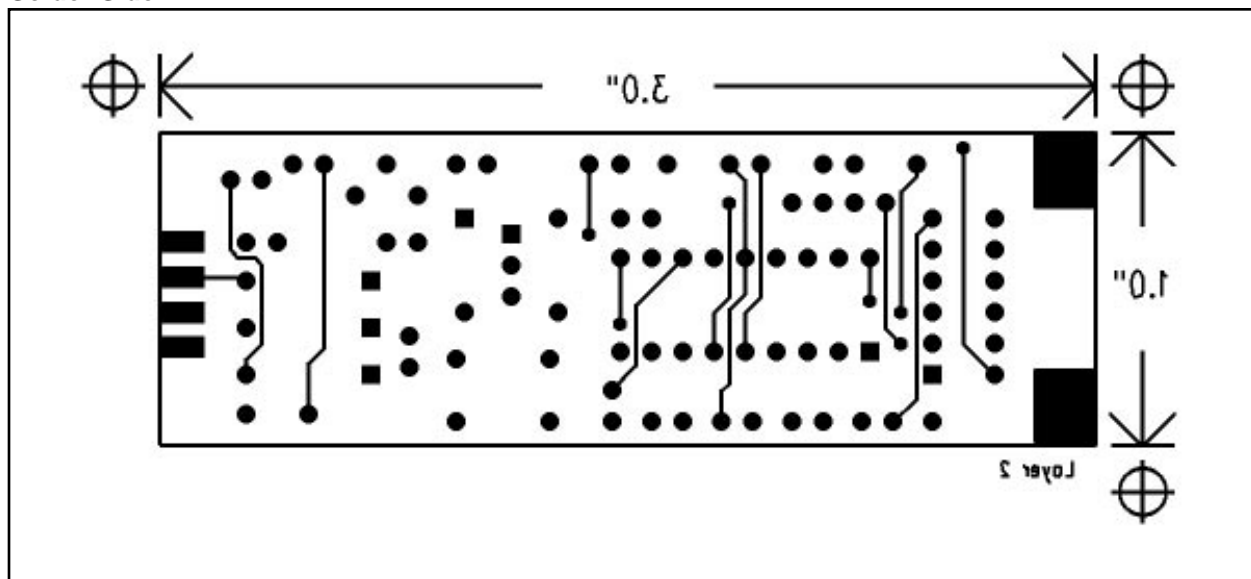
Boards Manufactured by: Southwest Circuits
Contact: Perry Groves
3760 E. 43rd Place
Tucson, AZ 85713
1-520-745-8515

The following artwork is not printed to scale:

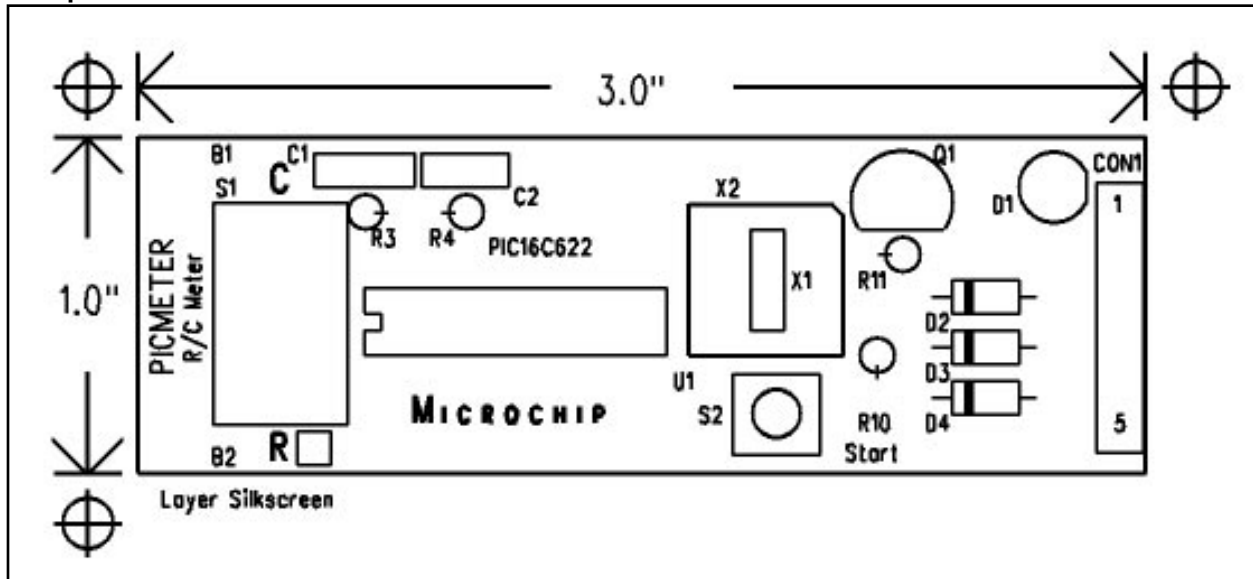
Component Side



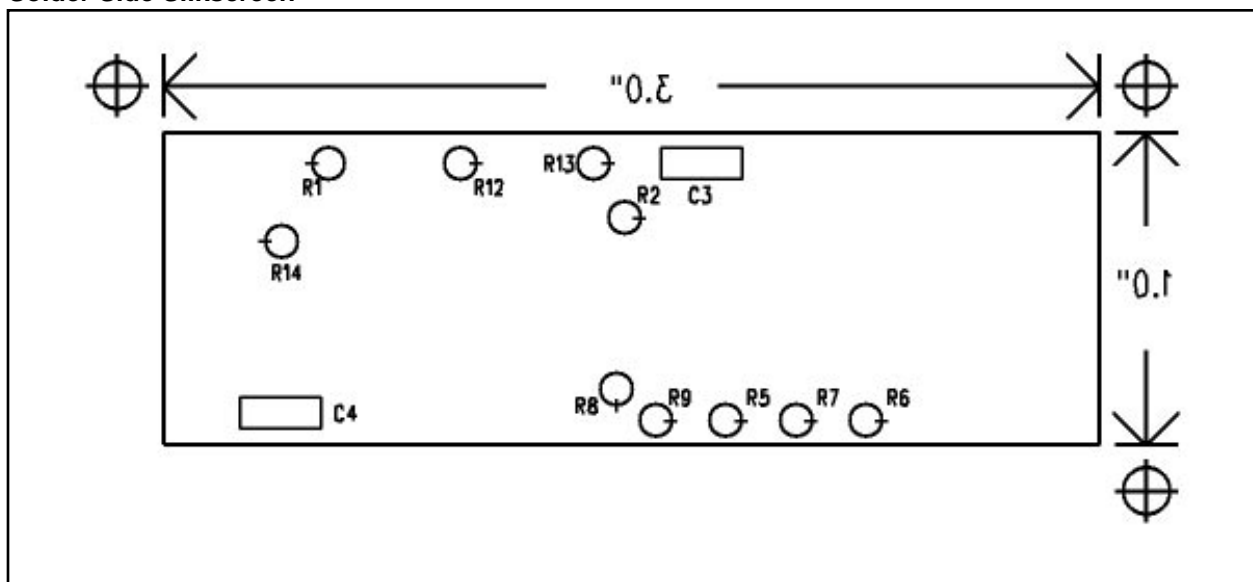
Solder Side



Component Side Silkscreen

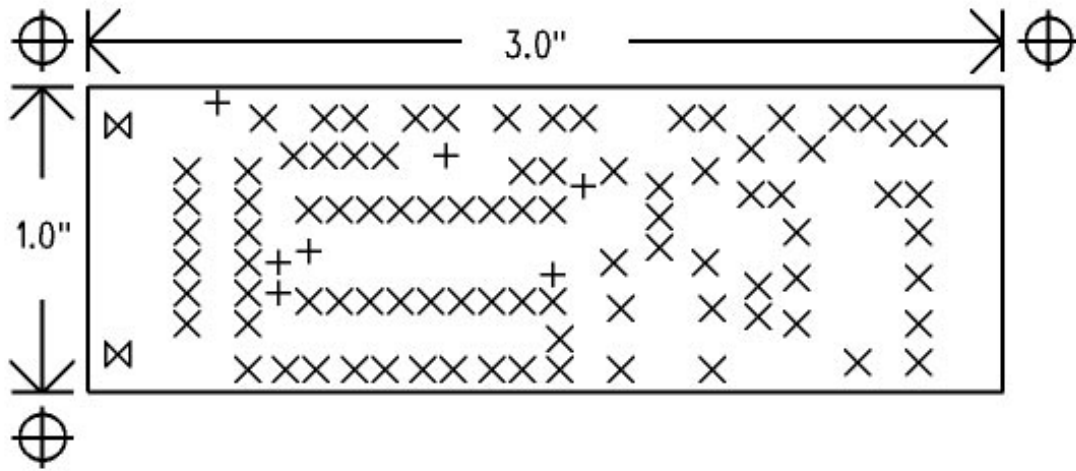


Solder Side Silkscreen



AN611

Manufacturing Drawing



SIZE	QTY	SYM
18	7	+
37	89	X
95	2	⊠

Clock Design Using Low Power/Cost Techniques

*Author: John Day
Sr., Field Application Engineer, Boston*

INTRODUCTION

Typical embedded control applications place demands such as low power consumption, small size, low cost and reduced component count onto the microcontroller. This application implements a 24-hour digital clock, alarm and 99 minute 59 second count down timer, yet operates on two "AA" batteries. The PIC16C54A is perfect for this application, due to its small size, high current I/Os with direct LED drive, low cost, fast instruction throughput and low frequency/current operation.

System cost

The objective of this design was to implement the maximum number of features with the least expensive and smallest device. The PIC16C54A is Microchip's lowest cost microcontroller and it has 12 I/O lines, each capable of sinking 25 mA and sourcing 20 mA. High efficiency common cathode LED displays were chosen for their 3.5 mA current requirement, eliminating the need for any external transistors for display drive. A low impedance direct drive piezo buzzer was chosen and its tone is generated by the software of the PIC16C54A to further reduce system cost.

Operating power

In battery powered applications, the operating current determines the lifetime of the batteries. There are many ways to reduce the operating current of any application, including low frequency operation and the use of sleep modes. Since the clock has to keep track of time, SLEEP mode could not be used and the processor must be kept running all of the time. The PIC16C54A supports the 32.768 kHz "watch" crystal and typically consumes less than 15 μ A of current in this configuration. Since the PIC16C54A executes instructions in one cycle and its instruction set is very efficient, this application was able to be implemented using a low frequency crystal. Another solution to this problem comes with the PIC16C74/73/65/63 in its Timer1 module. This timer will run when the device is asleep, so it could have been used to keep track of time, simplifying the software.

Clock system

A 32.768 kHz crystal was chosen for the clock due to the low power and cost requirements of this design. The four internal phases of this input clock create an internal instruction cycle. Therefore, the instruction time is calculated as follows:

$$\text{Instruction rate} = \frac{1}{(\text{CLKIN}/4)} = \frac{1}{32,768/4} = \frac{1}{8,192}$$

$$\text{Instruction cycle} = 122.07 \mu\text{s}$$

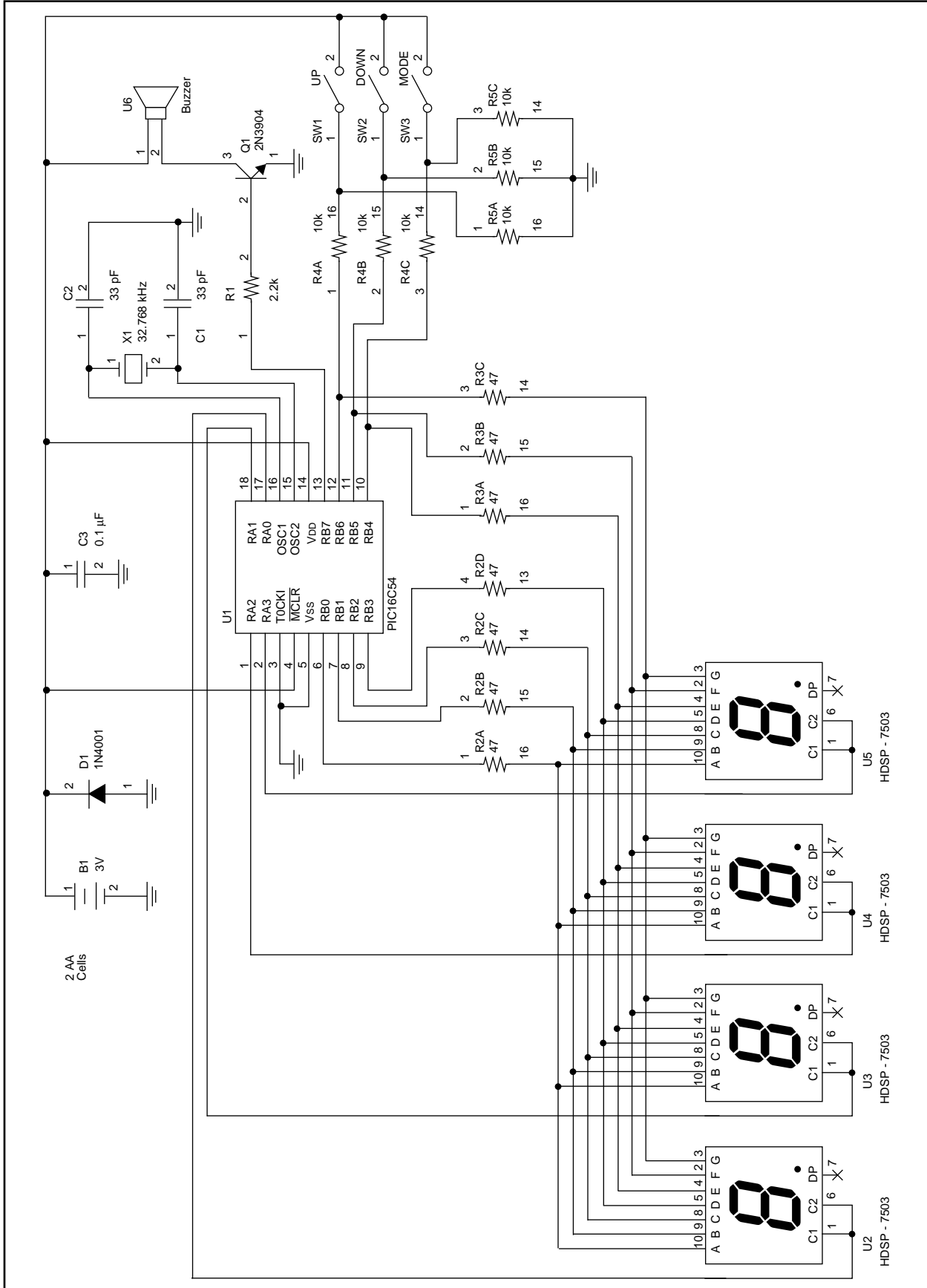
This means that every instruction executes in 122.07 μ s or we execute exactly 8,192 instructions per second.

Display and keypad multiplexing

The display contains four digits with seven segments each; therefore a multiplexing scheme was used to reduce the number of I/O lines needed to drive the displays. There are 4 common cathode display connections (one for each display digit and connected to PORTA for convenience so that rotates and moves can be used) and 7 segments (connected to PORTB for convenience so that moves can be used) for a total of 11 I/O lines needed for the display. Common Cathode displays were chosen, since the PIC16C54A can sink 5 mA more current than it can source. The last I/O line (RB7) was used to drive the buzzer. The three keys for setting the time are multiplexed onto the LED display segments to eliminate the need for additional I/O lines.

AN615

FIGURE 1: CLOCK DESIGN SCHEMATIC



SOFTWARE IMPLEMENTATION

The main loop of the software must perform the following tasks to implement the clock's functionality:

1. Determine when one second has passed (when bit7 on the TMR0 register changes state 4 times), increment the current time and (if enabled) decrement the countdown timer.
2. Determine if any of the alarms (countdown timer or the alarm time) are currently alarming or should be alarming. If so, the buzzer is buzzed.
3. Check for any keys that are pressed. If MODE is pressed, the current mode is incremented and if UP or DOWN is pressed, the time displayed is modified.
4. Automatically turn the displays on/off for power management.
5. Multiplex the displays every 3.9 ms (32 instruction cycles!).

General purpose registers are defined and used for the following purposes:

- **DISPSEGS_A** through **DISPSEGS_D** store the bit pattern that is to be displayed on each of the four 7-segment displays.
- **CLK_SEC** stores the second counter for the current clock time (values from 0-59 decimal are stored).
- **CLK_MIN_LD**, **CLK_MIN_HD** store the upper and lower minute digit of the current time.
- **CLK_HOUR_LD**, **CLK_HOUR_HD** store the upper and lower hour digit of the current time.
- **ALM_MIN_LD**, **ALM_MIN_HD** store the upper and lower minute digit of the alarm time.
- **ALM_HOUR_LD**, **ALM_HOUR_HD** store the upper and lower hour digit of the alarm time.
- **TMR_SEC_LD**, **TMR_SEC_HD** store the upper and lower second digit of the countdown timer.
- **TMR_MIN_LD**, **TMR_MIN_HD** store the upper and lower minute digit of the countdown timer.
- **KEYPAT** stores a pattern showing the currently pressed keys:
 - UP = bit6
 - DOWN = bit5
 - MODE = bit4
- **FLAGS** stores key flag bits such as the current mode, display on, alarm on, etc.
- **PREVTMR0** stores previous TMR0 values so that the differences can be detected the next time the TMR0 is polled.
- **TEMP** is a temporary register used for various routines.
- **DISPNCNT** stores the remaining number of seconds the displays should be on.

- **MODE_COUNT** stores the number of 1/2 seconds the MODE and UP or DOWN buttons are pressed. Used to switch from setting minutes to hours.
- **ALARMCNT** stores the number of beeps remaining to be driven into the buzzer.

FLAGS Register

Most designs require flag or state bits to indicate current modes or the state of a software routines. In this design, the FLAGS register is defined as follows:

Bits 0,1 -indicates the current operating mode (changed by pressing and releasing the MODE button):

00 - Display OFF

01 - Display/Set countdown timer

10 - Display/Set alarm time

11 - Display/Set clock (current time)

Bit 3 - indicates if (alarm time) = (current clock time)

Bit 4 - indicates if (count down timer) = 0

Bits 5,6,7 -Used as a divide by four counter to keep track of seconds

The software is broken up into the following routines for modularity:

buzz_now routine - *Output buzzing tone during alarm for 156 ms.*

Buzzers are available in self-oscillating and direct drive models. To save cost, a low impedance, direct drive model was selected. The buzz_now routine is called by the main_loop and it chirps the buzzer for 156 ms at a 1638 Hz frequency. This routine first turns off the LEDs (by clearing PORTB) and then uses TEMP to count for 256 pulses. The pulse is sent to the buzzer by the `BSF BUZZEROUT` and `BCF BUZZEROUT` instructions. This routine returns once 256 pulses are sent to the buzzer. This is necessary, since the controller cannot buzz the buzzer and keep track of time at the same time (running at such a low frequency), so these two functions are multiplexed.

task_scan routine - *multiplex LEDs to display the next digit, only one digit is lit at a time.*

The PIC16C5X family is designed for polled I/O applications and does not contain a hardware interrupt structure. To achieve the lowest cost design, the PIC16C54A was selected and all modules are written to CALL this task_scan routine within the multiplexing time frame of 3.9 ms or 32 instruction cycles. This routine first synchronizes itself with the TMR0 register, bit0 to ensure that the scanning occurs at the same point in time, regardless of when the routine is called. Next, PREVSCAN is rotated, setting up the CARRY bit correctly. The bit pattern for the next digit to be displayed is then moved into the W register. The display is blanked, PORTA is rotated (to select the next digit) and the next display bit pattern is moved to PORTB to display it. For ESD integrity, PORTA is later restored from the PREVSCAN register. This routine takes a total of 21 cycles (including the CALL and RETLW instructions) to execute and the displays are scanned every 3.9 ms (32 instruction cycles); therefore, this routine needs to be called after every 11 instruction cycles from every routine to maintain proper display multiplexing.

disp_value routine - *Update the display registers with the bitmap of what digits are to be displayed next.*

Indirect addressing is used here to reduce the amount of code needed and to simplify the routine. Since the clock, alarm or countdown time could each be displayed, the W register contains the base address (in the register file) of the four digits that are to currently be displayed. The W register is first moved to the FSR register so that the indirect address register contains the first digit to be displayed. The first digit is first converted into the segment bit pattern by calling the led_lookup table and then the bit pattern is moved to DISPSEGS_A. The FSR register is incremented (moving to the next digit) and the process is repeated for the remaining 3 digits. To maintain proper multiplexing, task_scan is called throughout this routine.

turnon_scan routine - *Turns on the LEDs and restores a legal scan position.*

To save battery power, the displays are automatically shut off after 8 seconds when no buttons are pressed. The DISPON bit is used to preset the remaining display on time to 8 seconds. This routine sets this flag (to later turn on the displays) and then checks to see if the PREVSCAN register contains a legal value (an illegal value of "FFh" is used to turn off all of the displays) and it restores a legal value if the displays were off.

scan_keys routine - *Turns off LEDs for a moment and scans the push-button inputs.*

To reduce the number of I/Os needed by this application, the three user input keys are multiplexed onto the LED display segments through PORTB. First, the PORTB is cleared and PORTA is set to '0Fh', turning off the LED displays. Next, PORTB is set up with bits 4,5 and 6 as inputs to read the keys. TEMP is then loaded with the keys that have changed state (to detect

the falling edge of a key press) and KEYPAT is loaded with a pattern ('0' = not pressed, '1' = pressed) for the keys that are pressed. Lastly, PORTB is restored to all outputs and the current multiplex scan is restored to PORTA.

check_time routine - *Checks for alarm or countdown timer expiration.*

Each second, alarm conditions must be detected and the buzzer sounded if an alarm condition is true. ALARMNOW and EGGNOW are flag bits that are used by the main program to sound the buzzer if they are set. This routine starts by setting both ALARMNOW and EGGNOW. Next, the current time hours and minutes are compared (through a subtraction and a test of the STATUS register Z bit) with the alarm time. If there is any miscompare, the ALARMNOW bit is cleared. To finish, the countdown timer time minutes and seconds digits are each compared with zero. If there is any miscompare, the EGGNOW bit is cleared. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.

inc_time routine - *Adds one second, minute or hour to the clock, alarm or timer.*

Every second, inc_time is called by main_loop to increment the seconds count for the clock. This routine is also called when the "UP" key is pressed and "MODE" key is held down to adjust the current time, alarm time or set the countdown timer. This routine uses indirect addressing to reduce the amount of code and simplify it's operation. Before this routine is called, the W register is loaded with the address of the clock second register and the routine is called. The FSR register is loaded with this value and the indirect address register is incremented (effectively incrementing the seconds counter).

Once the second counter is incremented, this register is checked for overflow (greater than 59 seconds) and if no overflow occurred, the routine returns. If an overflow happened, the second counter is cleared and the minute low digit is incremented. This register is then checked for an overflow (greater than 9 minutes) and so on until the all digits are updated.

This routine can also be called from multiple points. If called with the label inc_min_Id, only the minutes (and hours if an overflow occurs) will be incremented. Additionally, calling inc_hour_Id will increment only the hour digits. These features are used when setting the clock or alarm function. The FLAGS register (bits 0 and 1) is used to determine the current mode (clock, alarm or countdown timer) and ensure proper overflow calculations. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.

dec_time routine - *Subtracts one second, minute or hour from the clock, alarm or timer.*

If the countdown timer is enabled, dec_time is called by the main loop to decrement the seconds count for the countdown timer. This routine is also called by the main loop when the "DOWN" key is pressed and "MODE" key is held down to adjust the current time, alarm time or set the countdown timer. This routine uses indirect addressing to reduce the amount of code and simplify it's operation. Before this routine is called, the W register is loaded with the address of the countdown timer's second register and the routine is called. The FSR register is loaded with this value and the indirect address register is incremented (effectively incrementing the seconds counter).

Once the second counter LSD is decremented, this register is checked for underflow (less than 0 seconds) and if no underflow occurred, the routine returns. If an underflow happened, the second counter LSD is set to 9 and the second MSD is decremented. This register is then checked for an underflow (less than 0 seconds) and so on until all digits are updated.

This routine also can be called from multiple points. If called with the label dec_hour_id_vec, only the hour digits (or minutes if it is the countdown timer) will be decremented. This feature is used when setting the clock or alarm function. The FLAGS register (bits 0 and 1) is used to determine the current mode (clock, alarm or countdown timer) and ensure proper underflow calculations. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.

main_loop routine - *Calls the above routines as needed and keeps track of when to increment the clock or decrement the countdown timer.*

The main_loop calls all of the previous routines as necessary to maintain time, LED multiplexing, alarming and setting each function. The OPTION register is loaded with a 03h value to set up a Divide by 16 prescaler for the TMR0 register and internal instruction cycle increment. The instruction cycle is 122.07 μ s; therefore, bit0 changes every $(122 \mu\text{s} \cdot 16) = 1.953 \text{ ms}$ and bit7 changes every $(122.07 \mu\text{s} \cdot 16 \cdot 128) = 250 \text{ ms}$. Bits 5 and 6 of the FLAGS register are used to divide this 250 ms event by 4 to call inc_time every second.

The check_time routine is called after calling inc_time (every second), setting the EGGNOW or ALARMNOW flag bits. If the alarm is enabled, the buzzer is buzzed by calling buzz_now; however, the main timer updates need to occur in between buzzer beeps to keep track of time.

Every 500 ms, the keys are scanned and the edges on the MODE key are detected. Pressing the UP or DOWN key will shut off the buzzer (clearing the enable bits) and pressing the MODE key will advance the current mode. The mode is a 4-state state machine, revolving between the following states:

1. Display OFF - saves battery power - defaults to this mode if no keys are pressed for 8 seconds.
2. Display or Set countdown timer (holding MODE key allows setting).
3. Display or Set Alarm time (holding MODE key allows setting).
4. Display or Set Clock time (holding MODE key allows setting).

Next, the UP and DOWN keyscan values are tested and if the MODE and UP are both pressed, the currently displayed mode time is incremented or decremented. If MODE is not pressed and UP or DOWN is pressed, the displays are turned on, but the displayed time is not altered.

DISPONCNT is used to keep track of how long the displays have been on once all buttons are released. After 8 seconds, the displays are automatically turned off to save power. MODE_COUNT is used to switch from setting the right hand displays (minutes or seconds) to the left hand displays (hours or minutes). When the UP or DOWN button is held with mode for more than 4 seconds consecutively, MODE_COUNT reaches zero, switching from the right to left hand displays.

Finally, the main_loop finishes by updating the display registers by calling disp_value and if DISPONCNT has decreased to zero, the displays are turned off.

Lookup Tables - *Convert a number into a bit pattern or RAM address.*

There are three lookup tables used in this design for BCD to 7-Segment decoding, manufacturing diagnostics and looking up the address of the currently displayed mode.

- **mode_timer** - look-up the address of the clock, Alarm or Timer data storage RAM.
- **led_lookup** - look-up table contains the bitmap display pattern for displaying digits 0-9.
- **mfg_led_lookup** - look-up table contains the bitmap display pattern used for manufacturing mode. Only one segment is lit at a time.

Miscellaneous routines used for initialization and manufacturing test:

- **init** - Initializes all of RAM to zero, sets up the I/O ports and sets default time values.
- **mfg_selftest** - Used in manufacturing mode only - tests each LED segment, push-button, buzzer and display separately to expose bad keys, connections, buzzer or displays.

AN615

CONCLUSION

The implementation of this application highlights the PIC16C54's highly efficient instruction set, low power and frequency operation, high current direct LED drive capability and high performance instruction execution. Many of the routines used in this application note apply to a variety embedded control applications.

Ram Used: 25 Bytes

Code Space

Used: 444 Words (without manufacturing diagnostics)

510 Words (including manufacturing diagnostics)

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: CODE

MPASM 01.21.03 Intermediate CLK8.ASM 8-21-1995 9:17:56 PAGE 1

LOC OBJECT LINE SOURCE TEXT
VALUE CODE

```

00001 ; *****
00002 ; *      PIC Egg Timer Give-Away      *
00003 ; *
00004 ; * Author:   John Day                *
00005 ; *          Sr. Field Applications Engineer *
00006 ; *          Northeast Region          *
00007 ; *
00008 ; * Revision: 1.2                    *
00009 ; * Date     September 22, 1994      *
00010 ; * Part:   PIC16C54-LP/P or PIC16LC54A/P *
00011 ; * Fuses:  OSC:  LP                  *
00012 ; *          WDT:  OFF                *
00013 ; *          PuT:  OFF                *
00014 ; *          CP:   OFF                *
00015 ; *****
00016 ;
00017 ; This program is intended to run on a 32 Khz watch crystal and
00018 ; connects to four multiplexed seven segment displays. It displays the
00019 ; current time, alarm time and egg count down timers. There are
00020 ; switches that allow the user to set the alarm, timer and clock functions.
00021
00022 LIST F=INHX8M,P=16C54
00023 INCLUDE "p16C5X.inc"
00001      LIST
00002 ; P16C5X.INC Standard Header File, Version 2.02 Microchip Technology, Inc.
00143      LIST
OFFF  OFF8 00024      __FUSES __CP_OFF&_WDT_OFF&_LP_OSC
00025
0007  00026      ORG 07h
00027 ; *****
00028 ; * Static RAM Register File Definitions *
00029 ; *****
00000000 00030 INDADDR      EQU    0      ; Indirect address register
00000007 00031 DISPSEGS_A     EQU    07h     ; Current Display A segment bit pattern
00000008 00032 DISPSEGS_B     EQU    08h     ; Current Display B segment bit pattern
00000009 00033 DISPSEGS_C     EQU    09h     ; Current Display C segment bit pattern
0000000A 00034 DISPSEGS_D     EQU    0Ah     ; Current Display D segment bit pattern
0000000B 00035 CLK_SEC      EQU    0Bh     ; Clock second counter (0-59)
0000000C 00036 CLK_MIN_LD    EQU    0Ch     ; Clock minute low digit counter (0-9)
0000000D 00037 CLK_MIN_HD    EQU    0Dh     ; Clock minute high digit counter (0-5)
0000000E 00038 CLK_HOUR_LD   EQU    0Eh     ; Clock hour low digit counter (0-9)
0000000F 00039 CLK_HOUR_HD   EQU    0Fh     ; Clock hour high digit counter (0-2)
00000010 00040 ALM_MIN_LD     EQU    10h     ; Alarm minute low digit counter (0-9)
00000011 00041 ALM_MIN_HD   EQU    11h     ; Alarm minute high digit counter (0-5)
00000012 00042 ALM_HOUR_LD  EQU    12h     ; Alarm hour low digit counter (0-9)
00000013 00043 ALM_HOUR_HD  EQU    13h     ; Alarm hour high digit counter (0-2)
00000014 00044 TMR_SEC_LD  EQU    14h     ; Timer second low digit counter (0-9)
00000015 00045 TMR_SEC_HD  EQU    15h     ; Timer second high digit counter (0-5)
00000016 00046 TMR_MIN_LD  EQU    16h     ; Timer hour low digit counter (0-9)
00000017 00047 TMR_MIN_HD  EQU    17h     ; Timer hour high digit counter (0-2)
00000018 00048 KEYPAT     EQU    18h     ; Currently pressed key bits
00000019 00049 FLAGS      EQU    19h     ; Status of alarms, display on, etc.
0000001A 00050 PREVTMR0     EQU    1Ah     ; Used to determine which TMR0 bits changed
0000001B 00051 PREVSCAN    EQU    1Bh     ; Store Common Cathode display scan state
0000001C 00052 TEMP       EQU    1Ch     ; Temporary storage
0000001D 00053 DISPONCNT  EQU    1Dh     ; Time the displays have been on

```

AN615

```
0000001E      00054 MODE_COUNT      EQU      1Eh          ; Current mode state
0000001F      00055 ALARMCNT      EQU      1Fh          ; Time the alarm has been sounding
00056 ; *****
00057 ; * Flag and state bit definitions *
00058 ; *****
00059 #define          SECBIT      TEMP,7          ; Bit to spawn 1/4 second count
00060 #define          SCANBIT     TMR0,0          ; Bit to spawn display MUX
00061 #define          MODEKEY     KEYPAT,4        ; Bit for MODEKEY pressed
00062 #define          UPKEY       KEYPAT,6        ; Bit for UPKEY pressed
00063 #define          DOWNKEY     KEYPAT,5        ; Bit for DOWNKEY pressed
00064 #define          MODEKEYCHG  TEMP,4         ; Bit for delta MODEKEY
00065 #define          TIMENOW     FLAGS,7         ; Flag to indicate 1 second passed
00066 #define          ALARMCNT     FLAGS,3        ; Flag to indicate wakeup alarm
00067 #define          EGGNOW      FLAGS,4         ; Flag to indicate egg timer alarm
00068 #define          ALARMOK     STATUS,PA0      ; Flag to enable wakeup alarm
00069 #define          EGGOK       STATUS,PA1      ; Flag to enable timer alarm
00070 #define          BUZZEROUT    PORTB,7        ; Pin for pulsing the buzzer
00071 #define          DISPON      DISPONCNT,4    ; Bit to turn on LED displays
00072
00073 ; *****
00074 ; * Various Constants used throughout the program *
00075 ; *****
0000003C      00076 SEC_MAX      EQU      .60          ; Maximum value for second counter
0000000A      00077 MIN_LD_MAX   EQU      .10          ; Maximum value for low digit of minute
00000006      00078 MIN_HD_MAX   EQU      .6           ; Maximum value for high digit of minute
00000004      00079 HOUR_LD_MAX  EQU      .4           ; Maximum value for low digit of hour
00000002      00080 HOUR_HD_MAX  EQU      .2           ; Maximum value for high digit of hour
00000003      00081 OPTION_SETUP EQU      b'00000011' ; TMR0 - internal, /16 prescale
00000007      00082 BUZINITVAL  EQU      7           ;
00000008      00083 INIT_MODE_COUNT EQU      8           ; Digit counts to move to hour digits
00000028      00084 ALARMCYCNT  EQU      .40          ; Alarm for 10 seconds (ALARMCYCNT/4)
00085
01FF          00086          ORG      01FFh          ; The PIC5X reset vector is at end of memory
01FF          00087 reset_vector
01FF 0BA8      00088          GOTO     init          ; Jump to the initialization code
00089
0000          00090          ORG      0
00091 ; *****
00092 ; * Current mode look-up table *
00093 ; *****
0000          00094 mode_timer
0000 0E03      00095          ANDLW   3           ; Mask off upper bits just in case
0001 01E2      00096          ADDWF   PCL,F          ; Jump to one of 4 look-up entries
0002 0814      00097          RETLW   TMR_SEC_LD      ; Return the address of the 99 min timer RAM
0003 0810      00098          RETLW   ALM_MIN_LD      ; Return the address of the alarm RAM
0004 080C      00099          RETLW   CLK_MIN_LD      ; Return the address of the clock RAM
0005 080C      00100          RETLW   CLK_MIN_LD      ; Return the address of the clock RAM
00101
00102 ; *****
00103 ; * Buzz the buzzer for 1/8 second *
00104 ; *****
0006          00105 buzz_now
0006 0066      00106          CLRF    PORTB          ; Shut off the segments
0007          00107 buzz_now_dispon
0007 007C      00108          CLRF    TEMP           ; Buzz for 256 pulses
0008          00109 loop_buz
0008 05E6      00110          BSF     BUZZEROUT      ; Send out pulse
0009 04E6      00111          BCF     BUZZEROUT      ; Clear out the pulse
000A 02FC      00112          DECFSZ  TEMP,F          ; Decrement counter and skip when done
000B 0A08      00113          GOTO   loop_buz          ; Go back and send another pulse
000C 0800      00114          RETLW   0           ; We are done so come back!
00115
00116 ; *****
00117 ; * Mux drive the next LED display digit *
00118 ; *****
000D          00119 task_scan ; (19 (next_scan) + 2 = 21 cycles - must be called every 11 cy)
```



```

000D 0601      00120      BTFSC  SCANBIT      ; Synch up with 3.9 mS timer bit
000E 0A0D      00121      GOTO   task_scan   ; Jump back until bit is clear
00122
000F          00123 next_scan   ; (15 + 2 call + 2 return = 19 cycles)
000F 035B      00124      RLF   PREVSCAN,W   ; Move to the next digit select into C
0010 073B      00125      BTFSS PREVSCAN,1   ; 0 Check if display A was on before
0011 0209      00126      MOVF  DISPSEGS_C,W ; Place display B value into W
0012 071B      00127      BTFSS PREVSCAN,0   ; 1 Check if display B was on before
0013 0208      00128      MOVF  DISPSEGS_B,W ; Place display C value into W
0014 077B      00129      BTFSS PREVSCAN,3   ; 2 Check if display C was on before
0015 0207      00130      MOVF  DISPSEGS_A,W ; Place display D value into W
0016 075B      00131      BTFSS PREVSCAN,2   ; 3 Check if display D was on before
0017 020A      00132      MOVF  DISPSEGS_D,W ; Place display A value into W
0018 0066      00133      CLRF  PORTB        ; Turn off all segments
0019 037B      00134      RLF   PREVSCAN,F   ; Move to the next digit
001A 0365      00135      RLF   PORTA,F      ; Move port to the next digit
001B 0026      00136      MOVWF PORTB        ; Place next segment value on PORTB
001C 021B      00137      MOVF  PREVSCAN,W   ; Restore the port in case it is wrong
001D 0025      00138      MOVWF PORTA        ; Restore the port
001E 0800      00139      RETLW 0            ; Display is updated - now return
00140
00141
00142 ; *****
00143 ; * Move new digit display info out to display *
00144 ; *****
001F          00145 disp_value
001F 0024      00146      MOVWF FSR          ; Place W into FSR for indirect addressing
0020 090D      00147      CALL  task_scan    ; Scan the next LED digit.
0021 0200      00148      MOVF  INDADDR,W   ; Place display value into W
0022 0937      00149      CALL  led_lookup   ; Look up seven segment value
0023 0027      00150      MOVWF DISPSEGS_A  ; Move value out to display register A
0024 02A4      00151      INCF  FSR,F        ; Go to next display value
0025 090D      00152      CALL  task_scan    ; Scan the next LED digit.
0026 0200      00153      MOVF  INDADDR,W   ; Place display value into W
0027 0937      00154      CALL  led_lookup   ; Look up seven segment value
0028 0028      00155      MOVWF DISPSEGS_B  ; Move value out to display register B
0029 02A4      00156      INCF  FSR,F        ; Go to next display value
002A 090D      00157      CALL  task_scan    ; Scan the next LED digit.
002B 0200      00158      MOVF  INDADDR,W   ; Place display value into W
002C 0937      00159      CALL  led_lookup   ; Look up seven segment value
002D 0029      00160      MOVWF DISPSEGS_C  ; Move value out to display register C
002E 02A4      00161      INCF  FSR,F        ; Go to next display value
002F 090D      00162      CALL  task_scan    ; Scan the next LED digit.
0030 0200      00163      MOVF  INDADDR,W   ; Place display value into W
0031 0643      00164      BTFSC STATUS,Z    ; ZBLANK - Check for a zero
0032 0240      00165      COMF  INDADDR,W   ; ZBLANK - Clear digit with FF if leading 0
0033 0937      00166      CALL  led_lookup   ; Look up seven segment value
0034 002A      00167      MOVWF DISPSEGS_D  ; Move value out to display register D
0035 090D      00168      CALL  task_scan    ; Scan the next LED digit.
0036 0800      00169      RETLW 0
00170
00171 ; *****
00172 ; * Convert display value into segments *
00173 ; *****
0037          00174 led_lookup
0037 0E0F      00175      ANDLW 0Fh         ; Strip off upper digits
0038 01E2      00176      ADDWF PCL,F       ; Jump into the correct location
0039 083F      00177      RETLW b'00111111' ; Bit pattern for a Zero
003A 0806      00178      RETLW b'00000110' ; Bit pattern for a One
003B 085B      00179      RETLW b'01011011' ; Bit pattern for a Two
003C 084F      00180      RETLW b'01001111' ; Bit pattern for a Three
003D 0866      00181      RETLW b'01100110' ; Bit pattern for a Four
003E 086D      00182      RETLW b'01101101' ; Bit pattern for a Five
003F 087D      00183      RETLW b'01111101' ; Bit pattern for a Six
0040 0807      00184      RETLW b'00000111' ; Bit pattern for a Seven
0041 087F      00185      RETLW b'01111111' ; Bit pattern for a Eight

```

AN615

```
0042 086F      00186      RETLW    b'01101111'      ; Bit pattern for a Nine
0043 0800      00187      RETLW    0                ; Turn display off - ILLEGAL VALUE
0044 0800      00188      RETLW    0                ; Turn display off - ILLEGAL VALUE
0045 0800      00189      RETLW    0                ; Turn display off - ILLEGAL VALUE
0046 0800      00190      RETLW    0                ; Turn display off - ILLEGAL VALUE
0047 0800      00191      RETLW    0                ; Turn display off - ILLEGAL VALUE
0048 0800      00192      RETLW    0                ; Turn display off - ILLEGAL VALUE
00193
00194 ; *****
00195 ; * Convert display value into single segment ON for manufacturing diags *
00196 ; *****
0049          00197 mfg_led_lookup
0049 0E07      00198      ANDLW    07h             ; Strip off upper digits
004A 01E2      00199      ADDWF    PCL,F           ; Jump into the correct location
004B 0801      00200      RETLW    b'00000001'     ; Bit pattern for segment A on only
004C 0802      00201      RETLW    b'00000010'     ; Bit pattern for segment B on only
004D 0804      00202      RETLW    b'00000100'     ; Bit pattern for segment C on only
004E 0808      00203      RETLW    b'00001000'     ; Bit pattern for segment D on only
004F 0810      00204      RETLW    b'00010000'     ; Bit pattern for segment E on only
0050 0820      00205      RETLW    b'00100000'     ; Bit pattern for segment F on only
0051 0840      00206      RETLW    b'01000000'     ; Bit pattern for segment G on only
0052 087F      00207      RETLW    b'01111111'     ; Bit pattern for all segments on
00208
00209 ; *****
00210 ; * Wake-up and turn on the displays *
00211 ; *****
0053          00212 turnon_scan
0053 059D      00213      BSF      DISPON         ; Set display ON bit
0054 0CEE      00214      MOVLW    b'11101110'     ; Place digit 0 scan pattern in W
0055 019B      00215      XORWF    PREVSCAN,W      ; See if this is the current scan
0056 0643      00216      BTFSC    STATUS,Z        ; Skip if this is not the current scan
0057 0800      00217      RETLW    0                ; Legal scan value - we are done!
0058 0CDD      00218      MOVLW    b'11011101'     ; Place digit 1 scan pattern in W
0059 019B      00219      XORWF    PREVSCAN,W      ; See if this is the current scan
005A 0643      00220      BTFSC    STATUS,Z        ; Skip if this is not the current scan
005B 0800      00221      RETLW    0                ; Legal scan value - we are done!
005C 0CBB      00222      MOVLW    b'10111011'     ; Place digit 2 scan pattern in W
005D 019B      00223      XORWF    PREVSCAN,W      ; See if this is the current scan
005E 0643      00224      BTFSC    STATUS,Z        ; Skip if this is not the current scan
005F 0800      00225      RETLW    0                ; Legal scan value - we are done!
0060 0C77      00226      MOVLW    b'01110111'     ; Place digit 3 scan pattern in W
0061 019B      00227      XORWF    PREVSCAN,W      ; See if this is the current scan
0062 0643      00228      BTFSC    STATUS,Z        ; Skip if this is not the current scan
0063 0800      00229      RETLW    0                ; Legal scan value - we are done!
0064 0CEE      00230      MOVLW    0EEh            ; Move digit 0 scan value into W
0065 003B      00231      MOVWF    PREVSCAN        ; Move it into scan pattern register
00232
00233 ; *****
00234 ; * Scan for pressed keys *
00235 ; *****
0066          00236 scan_keys
0066 0066      00237      CLRF    PORTB            ; Turn off all of the segments
0067 0CFF      00238      MOVLW    0FFh           ; Place FF into W
0068 0025      00239      MOVWF    PORTA           ; Make PORT A all ones
0069 0C70      00240      MOVLW    b'01110000'     ; Place 70 into W
006A 0006      00241      TRIS    PORTB            ; Make RB4,5,6 inputs others outputs
006B 0206      00242      MOVF    PORTB,W          ; Place keyscan value into W
006C 0198      00243      XORWF    KEYPAT,W        ; Place Delta key press into W
006D 003C      00244      MOVWF    TEMP            ; Place Delta key press into TEMP
006E 01B8      00245      XORWF    KEYPAT,F        ; Update KEYPAT reg to buttons pressed
006F 0040      00246      CLRW                    ; Place 0 into W
0070 0006      00247      TRIS    PORTB            ; Make PORT B outputs
0071 021B      00248      MOVF    PREVSCAN,W       ; Place previous scan value into W
0072 0025      00249      MOVWF    PORTA           ; Turn on the scan
0073 0800      00250      RETLW    0
00251 ; *****
```

```

00252 ; * Check if alarm or timer is expired *
00253 ; *****
00254 check_time
0074 090D 00255 CALL task_scan ; Scan the next LED digit.
0075 0579 00256 BSF ALARMINOW ; Set the alarm bit
0076 0599 00257 BSF EGGNOW ; Set the Egg timer alarm bit
0077 0210 00258 MOVF ALM_MIN_LD,W ; Place alarm minute counter into W
0078 008C 00259 SUBWF CLK_MIN_LD,W ; CLK_MIN_LD - W -> W
0079 0743 00260 BTFSS STATUS,Z ; Skip if they are equal
007A 0479 00261 BCF ALARMINOW ; They are not equal so clear alarm bit
007B 0211 00262 MOVF ALM_MIN_HD,W ; Place alarm minute counter into W
007C 008D 00263 SUBWF CLK_MIN_HD,W ; CLK_MIN_HD - W -> W
007D 0743 00264 BTFSS STATUS,Z ; Skip if they are equal
007E 0479 00265 BCF ALARMINOW ; They are not equal so clear alarm bit
007F 090D 00266 CALL task_scan ; Scan the next LED digit.
0080 0212 00267 MOVF ALM_HOUR_LD,W ; Place alarm hour counter into W
0081 008E 00268 SUBWF CLK_HOUR_LD,W ; CLK_HOUR_LD - W -> W
0082 0743 00269 BTFSS STATUS,Z ; Skip if they are equal
0083 0479 00270 BCF ALARMINOW ; They are not equal so clear alarm bit
0084 0213 00271 MOVF ALM_HOUR_HD,W ; Place alarm hour counter into W
0085 008F 00272 SUBWF CLK_HOUR_HD,W ; CLK_HOUR_LD - W -> W
0086 0743 00273 BTFSS STATUS,Z ; Skip if they are equal
0087 0479 00274 BCF ALARMINOW ; They are not equal so clear alarm bit
0088 090D 00275 CALL task_scan ; Scan the next LED digit.
0089 0214 00276 MOVF TMR_SEC_LD,W ; Set the Z bit to check for zero
008A 0743 00277 BTFSS STATUS,Z ; Skip if this digit is zero
008B 0499 00278 BCF EGGNOW ; Timer is not zero so clear egg alarm bit
008C 0215 00279 MOVF TMR_SEC_HD,W ; Set the Z bit to check for zero
008D 0743 00280 BTFSS STATUS,Z ; Skip if this digit is zero
008E 0499 00281 BCF EGGNOW ; Timer is not zero so clear egg alarm bit
008F 0216 00282 MOVF TMR_MIN_LD,W ; Set the Z bit to check for zero
0090 0743 00283 BTFSS STATUS,Z ; Skip if this digit is zero
0091 0499 00284 BCF EGGNOW ; Timer is not zero so clear egg alarm bit
0092 090D 00285 CALL task_scan ; Scan the next LED digit.
0093 0217 00286 MOVF TMR_MIN_HD,W ; Set the Z bit to check for zero
0094 0743 00287 BTFSS STATUS,Z ; Skip if this digit is zero
0095 0499 00288 BCF EGGNOW ; Timer is not zero so clear egg alarm bit
0096 0799 00289 BTFSS EGGNOW ; Skip if we are still at EGG Time
0097 05C3 00290 BSF EGGOK ; If we are not at EGG time, re-set egg alarm
0098 0779 00291 BTFSS ALARMINOW ; Skip if we are still at Alarm time
0099 05A3 00292 BSF ALARMOK ; If we are not at Alarm time, re-set alarm
009A 090D 00293 CALL task_scan ; Scan the next LED digit.
009B 0800 00294 RETLW 0
00295
00296 ; *****
00297 ; * Increment the clock, timer or alarm *
00298 ; *****
009C 002A 00300 MOVWF FSR ; Add one to clock second counter
009D 090D 00301 CALL task_scan ; Scan the next LED digit.
009E 02A0 00302 INCF INDADDR,f ; Add one to minute lower digit
009F 0C3C 00303 MOVLW SEC_MAX ; Place second max value into w
00A0 0080 00304 SUBWF INDADDR,W ; CLOCK_SEC - SEC_MAX -> W
00A1 0703 00305 BTFSS STATUS,C ; Skip if there is an overflow
00A2 0800 00306 RETLW 0 ; We are done so let's get out of here!
00A3 006B 00307 CLRWF CLK_SEC ; Clear CLK_second counter
00A4 02A4 00308 INCF FSR,F ; Move to the next digit
00A5 02A0 00309 INCF INDADDR,F ; Add 1 to minute LOW digit
00A6 0AA9 00310 GOTO skip_min_fsr ; Jump to the next digit
00A7 00311 inc_min_ld
00A7 0024 00312 MOVWF FSR
00A8 02A0 00313 INCF INDADDR,F ; Add 1 to minute LOW digit
00A9 00314 skip_min_fsr
00A9 090D 00315 CALL task_scan ; Scan the next LED digit.
00AA 0C0A 00316 MOVLW MIN_LD_MAX ; Place minute lower digit max value into W
00AB 0080 00317 SUBWF INDADDR,W ; CLK_MIN_LD - MIN_LD_MAX -> W

```

AN615

```
00AC 0703      00318      BTFSS      STATUS,C      ; Skip if there is an overflow
00AD 0800      00319      RETLW      0            ; We are done so let's get out of here!
00AE 0060      00320      CLRF      INDADDR      ; Clear CLK minute low digit
00AF 02A4      00321      INCF      FSR,F        ; Move to the minute high digit
00B0 02A0      00322      INCF      INDADDR,F    ; Add one to minute high digit
00B1          00323      inc_min_hd
00B1 090D      00324      CALL      task_scan    ; Scan the next LED digit.
00B2 0C06      00325      MOVLW     MIN_HD_MAX   ; Place minute high digit max value into W
00B3 0080      00326      SUBWF     INDADDR,W    ; CLK_MIN_HD - MIN_HD_MAX -> W
00B4 0703      00327      BTFSS     STATUS,C    ; Skip if there is an overflow
00B5 0800      00328      RETLW     0            ; We are done so let's get out of here!
00B6 0060      00329      CLRF      INDADDR      ; Clear CLK minute high digit
00B7 02A4      00330      INCF      FSR,F        ; Move to the hour low digit
00B8 02A0      00331      INCF      INDADDR,F    ; Add one to hour low digit
00B9 0ABE      00332      GOTO      skip_hour_fsr ; Jump to the next digit
00BA          00333      inc_hour_ld
00BA 0024      00334      MOVWF     FSR          ;
00BB 02A4      00335      INCF      FSR,F        ;
00BC 02A4      00336      INCF      FSR,F        ;
00BD 02A0      00337      INCF      INDADDR,F    ; Add 1 to minute LOW digit
00BE          00338      skip_hour_fsr
00BE 090D      00339      CALL      task_scan    ; Scan the next LED digit.
00BF 0C0A      00340      MOVLW     MIN_LD_MAX   ; Place hour lower digit max value into W
00C0 0080      00341      SUBWF     INDADDR,W    ; CLK_HOUR_LD - HOUR_LD_MAX -> W
00C1 0703      00342      BTFSS     STATUS,C    ; Skip if there is an overflow
00C2 0AC7      00343      GOTO      check_inc    ; We need to check for overflow
00C3 0060      00344      CLRF      INDADDR      ; Clear CLK hour low digit
00C4 02A4      00345      INCF      FSR,F        ; Move to the hour high digit
00C5 02A0      00346      INCF      INDADDR,F    ; Add one to hour high digit
00C6 0AC8      00347      GOTO      inc_hour_hd  ;
00C7          00348      check_inc
00C7 02A4      00349      INCF      FSR,F        ; Move to hour high digit
00C8          00350      inc_hour_hd
00C8 090D      00351      CALL      task_scan    ; Scan the next LED digit.
00C9 0C02      00352      MOVLW     HOUR_HD_MAX  ; Place hour high digit max value into W
00CA 0639      00353      BTFSC     FLAGS,1     ;
00CB 0ACE      00354      GOTO      off_model1  ;
00CC 0619      00355      BTFSC     FLAGS,0     ;
00CD 0C09      00356      MOVLW     MIN_LD_MAX-1 ;
00CE          00357      off_model1
00CE 0080      00358      SUBWF     INDADDR,W    ; CLK_HOUR_HD - HOUR_HD_MAX -> W
00CF 0703      00359      BTFSS     STATUS,C    ; Skip if there is an overflow
00D0 0800      00360      RETLW     0            ; We are done so let's get out of here!
00D1 00E4      00361      DECF      FSR,F        ; Move to the hour low digit
00D2 090D      00362      CALL      task_scan    ; Scan the next LED digit.
00D3 0C04      00363      MOVLW     HOUR_LD_MAX  ; Place hour high digit max value into W
00D4 0639      00364      BTFSC     FLAGS,1     ;
00D5 0AD8      00365      GOTO      off_mode2   ;
00D6 0619      00366      BTFSC     FLAGS,0     ;
00D7 0C00      00367      MOVLW     0            ; Clear W
00D8          00368      off_mode2
00D8 0080      00369      SUBWF     INDADDR,W    ; CLK_HOUR_HD - HOUR_HD_MAX -> W
00D9 0703      00370      BTFSS     STATUS,C    ; Skip if there is an overflow
00DA 0800      00371      RETLW     0            ; We are done so let's get out of here!
00DB 090D      00372      CALL      task_scan    ; Scan the next LED digit.
00DC 0060      00373      CLRF      INDADDR      ; Clear hour high digit
00DD 0639      00374      BTFSC     FLAGS,1     ;
00DE 0AE0      00375      GOTO      off_mode3   ;
00DF 0719      00376      BTFSS     FLAGS,0     ;
00E0          00377      off_mode3
00E0 0000      00378      NOP
00E1 02A4      00379      INCF      FSR,F        ; Move to the hour high digit
00E2 0060      00380      CLRF      INDADDR      ; Clear one hour low digit
00E3 090D      00381      CALL      task_scan    ;
00E4 0800      00382      RETLW     0            ; We are done so let's get out of here!
00E4          00383
```

```

00E5          00384 dec_hour_ld
00E5 0AF9     00385          GOTO    dec_hour_ld_vect ; ran out of CALL space...
00386
00387 ; *****
00388 ; * Decriment the clock, alarm or timer *
00389 ; *****
00E6          00390 dec_time
00E6          00391 dec_min_ld
00E6 0024     00392          MOVWF  FSR          ; Set up pointer for indirect address
00E7 090D     00393          CALL   task_scan      ; Scan the next LED digit.
00E8 00E0     00394          DECF  INDADDR,F      ; Subtract one from CLK_MIN_LD
00E9 0240     00395          COMF  INDADDR,W      ; Set the Z bit to check for zero
00EA 0743     00396          BTFSS STATUS,Z      ; Skip if CLK_MIN_LD is zero
00EB 0800     00397          RETLW 0              ; We are done... Let's get out of here
00EC 0C09     00398          MOVLW MIN_LD_MAX - 1 ; Place minute lower digit max value into W
00ED 0020     00399          MOVWF INDADDR        ; MIN_LD_MAX -> CLK_MIN_LD
00EE          00400 dec_min_hd
00EE 090D     00401          CALL   task_scan      ; Scan the next LED digit.
00EF 02A4     00402          INCF  FSR,F          ; Move the pointer to Min HIGH DIGIT
00F0 00E0     00403          DECF  INDADDR,F      ; Subtract one from CLK_MIN_LD
00F1 0240     00404          COMF  INDADDR,W      ; Set the Z bit to check for zero
00F2 0743     00405          BTFSS STATUS,Z      ; Skip if CLK_MIN_LD is zero
00F3 0800     00406          RETLW 0              ; We are done... Let's get out of here
00F4 0C05     00407          MOVLW MIN_HD_MAX - 1 ; Place minute lower digit max value into W
00F5 0020     00408          MOVWF INDADDR        ; MIN_HD_MAX -> CLK_MIN_LD
00F6 090D     00409          CALL   task_scan      ; Scan the next LED digit.
00F7 02A4     00410          INCF  FSR,F          ; Move the pointer to Hour LOW DIGIT
00F8 0AFD     00411          GOTO   skip_dhour_fsr ; Jump to the next digit
00F9          00412 dec_hour_ld_vect
00F9 0024     00413          MOVWF  FSR
00FA 02A4     00414          INCF  FSR,F
00FB 02A4     00415          INCF  FSR,F
00FC 090D     00416          CALL   task_scan      ; Scan the next LED digit.
00FD          00417 skip_dhour_fsr
00FD 00E0     00418          DECF  INDADDR,F      ; Subtract one from CLK_HOUR_LD
00FE 0240     00419          COMF  INDADDR,W      ; Set the Z bit to check for zero
00FF 0743     00420          BTFSS STATUS,Z      ; Skip if CLK_MIN_LD is zero
0100 0B06     00421          GOTO   check_hour
0101 0C09     00422          MOVLW MIN_LD_MAX - 1 ; Place minute lower digit max value into W
0102 0020     00423          MOVWF INDADDR        ; MIN_LD_MAX -> CLK_HOUR_LD
0103 02A4     00424          INCF  FSR,F          ; Move the pointer to Hour HIGH DIGIT
0104 00E0     00425          DECF  INDADDR,F      ; Subtract one from CLK_HOUR_LD
0105 0B07     00426          GOTO   dec_hour_hd
0106          00427 check_hour
0106 02A4     00428          INCF  FSR,F          ; Point to hour high digit
0107          00429 dec_hour_hd
0107 090D     00430          CALL   task_scan      ; Scan the next LED digit.
0108 0240     00431          COMF  INDADDR,W      ;
0109 0743     00432          BTFSS STATUS,Z      ;
010A 0800     00433          RETLW 0              ;
010B 090D     00434          CALL   task_scan      ; Scan the next LED digit.
010C 00E4     00435          DECF  FSR,F          ;
010D 0C09     00436          MOVLW .9              ; Reset digit to 9
010E 0080     00437          SUBWF INDADDR,W      ;
010F 0743     00438          BTFSS STATUS,Z      ; Skip if CLK_MIN_LD is zero
0110 0800     00439          RETLW 0              ; We are done... Let's get out of here
0111 090D     00440          CALL   task_scan      ; Scan the next LED digit.
0112 02A4     00441          INCF  FSR,F          ;
0113 0C02     00442          MOVLW HOUR_HD_MAX    ; Place minute lower digit max value into W
0114 0739     00443          BTFSS FLAGS,1        ; Skip if CLOCK or ALARM mode
0115 0C09     00444          MOVLW .9              ; Reset digit to 9
0116 0020     00445          MOVWF INDADDR        ; HOUR_HD_MAX -> CLK_HOUR_LD
0117 0C03     00446          MOVLW HOUR_LD_MAX - 1 ; Place minute lower digit max value into W
0118 0739     00447          BTFSS FLAGS,1        ; Skip if CLOCK or ALARM mode
0119 0C09     00448          MOVLW .9              ; Reset digit to 9
011A 00E4     00449          DECF  FSR,F          ; Move the pointer to Min LOW DIGIT

```

AN615

```
011B 0020      00450      MOVWF  INDADDR      ; HOUR_LD_MAX -> CLK_HOUR_LD
011C 090D      00451      CALL   task_scan    ; Scan the next LED digit.
011D 0800      00452      RETLW  0            ; We are done... Let's get out of here
00453
00454 ; *****
00455 ; * Main loop calls all tasks as needed *
00456 ; *****
011E          00457 main_loop
011E 090D      00458      CALL   task_scan    ; Scan the next LED digit.
011F 0201      00459      MOVF   TMR0,W       ; Place current TMR0 value into W
0120 019A      00460      XORWF  PREV_TMR0,W  ; Lets see which bits have changed...
0121 003C      00461      MOVWF  TEMP         ; All changed bits are placed in temp for test
0122 01BA      00462      XORWF  PREV_TMR0,F  ; Update Previous TMR0 value.
0123 07FC      00463      BTFSS  SECBIT       ; Skip if it is not time to increment second
0124 0B1E      00464      GOTO   main_loop    ; Go back to main loop if 250 mS not passed
0125 0C20      00465      MOVLW  b'00100000' ; Bits 6 and 5 of FLAGS used as divide by 4
0126 01F9      00466      ADDWF  FLAGS,F       ; Add one to bit 5
0127 07F9      00467      BTFSS  TIMENOW      ; Check bit 7 - if four adds occur, skip
0128 0B38      00468      GOTO   skip_timer   ; One second has not passed - skip timers
0129 090D      00469      CALL   task_scan    ; Scan the next LED digit.
012A 04F9      00470      BCF    TIMENOW      ; Clear out second passed flag
012B 0C0B      00471      MOVLW  CLK_SEC      ; Place pointer to increment clock
012C 099C      00472      CALL   inc_time     ; Increment the clock
012D 0974      00473      CALL   check_time   ; Check for alarm or timer conditions
012E 0699      00474      BTFSC  EGGNOW       ; Do NOT decrease timer if zero
012F 0B38      00475      GOTO   skip_timer   ; Jump out if egg timer is zero
0130 06D8      00476      BTFSC  UPKEY        ; Skip if UP key is NOT pressed
0131 0B38      00477      GOTO   skip_timer   ; Jump out if UP key is pressed
0132 06B8      00478      BTFSC  DOWNKEY      ; Skip if DOWN key is NOT pressed
0133 0B38      00479      GOTO   skip_timer   ; Jump out if DOWN key is pressed
0134 0C14      00480      MOVLW  TMR_SEC_LD   ; Place pointer to decrement timer
0135 09E6      00481      CALL   dec_time     ; Decrement countdown timer
0136 0C28      00482      MOVLW  ALARMCYCNT   ; Place the number of alarm beeps into W
0137 003F      00483      MOVWF  ALARMCNT     ; Move beep count to ALARMCNT
0138          00484 skip_timer
0138 07A3      00485      BTFSS  ALARMOK      ; Skip if this is the first pass into alarm
0139 0B3F      00486      GOTO   skip_wakeup  ; Second pass - do not re-init ALARMCNT
013A 0779      00487      BTFSS  ALARMNOW     ; Skip if this is alarm pass
013B 0B3F      00488      GOTO   skip_wakeup  ; Countdown timer - do not re-init ALARMCNT
013C 0C28      00489      MOVLW  ALARMCYCNT   ; Place the number of alarm beeps into W
013D 003F      00490      MOVWF  ALARMCNT     ; Move beep count to ALARMCNT
013E 04A3      00491      BCF    ALARMOK      ; Clear flag for second pass
013F          00492 skip_wakeup
013F 090D      00493      CALL   task_scan    ; Scan the next LED digit.
0140 0679      00494      BTFSC  ALARMNOW     ; Skip if alarm clock is not set
0141 0B45      00495      GOTO   send_alarm   ; Blast out a beep
0142 0699      00496      BTFSC  EGGNOW       ; Skip if countdown timer is not alarming
0143 0B45      00497      GOTO   send_alarm   ; Blast out a beep
0144 0B4A      00498      GOTO   skip_alarm   ; Skip beeping and continue
0145          00499 send_alarm
0145 021F      00500      MOVF   ALARMCNT,W   ; Place ALARMCNT into W
0146 0643      00501      BTFSC  STATUS,Z     ; Skip if not zero
0147 0B4A      00502      GOTO   skip_alarm   ; We are done beeping - skip and continue
0148 02FF      00503      DECFSZ ALARMCNT,F   ; Decrement beep count and skip when zero
0149 0906      00504      CALL   buzz_now     ; Blast out the beep!!!
014A          00505 skip_alarm
014A 07B9      00506      BTFSS  FLAGS,5      ; Skip if it is time to scan the keys 1/2 sec
014B 0B9A      00507      goto   finish_update ; Jump to finish updates - don't scan
014C 0966      00508      CALL   scan_keys    ; Scan the keys and load value into KEYPAT
014D 090D      00509      CALL   task_scan    ; Scan the next LED digit.
014E 0798      00510      BTFSS  MODEKEY      ; Skip if the MODEKEY is pressed
014F 0B55      00511      GOTO   same_mode    ; Not pressed so it is the same mode...
0150 079C      00512      BTFSS  MODEKEYCHG   ; Skip if the is pressing edge
0151 0B55      00513      GOTO   same_mode    ; Button is held so it is the same mode...
0152 02B9      00514      INCF   FLAGS,F       ; Advance the mode by incrementing bits 0,1
0153 0459      00515      BCF    FLAGS,2      ; Force mode to wrap-around by clearing bit 2
```

```

0154 0953      00516      CALL    turnon_scan    ; Mode button pressed - must turn on LEDs
0154 0953      00517
0155           00518 same_mode
0155 090D      00519      call    task_scan     ; Scan the next LED digit.
0156 06D8      00520      BTFSC  UPKEY         ; Skip if the UP key is not pressed
0157 0B66      00521      GOTO   serve_up_key  ; UP key is pressed - jump to serve it!
0158 06B8      00522      BTFSC  DOWNKEY      ; Skip if the DOWN key is not pressed
0159 0B81      00523      GOTO   serve_down_key ; DOWN key is pressed - jump to serve it!
015A 0C08      00524      MOVLW  INIT_MODE_COUNT ; UP and DOWN not pressed - re-init mode count
015B 003E      00525      MOVWF  MODE_COUNT    ; Change back to lower digits for setting
015C 023D      00526      MOVF   DISPONCNT,F   ; Update Z bit in STATUS reg display on time
015D 0743      00527      BTFSS  STATUS,Z     ; Skip if displays should be OFF
015E 00FD      00528      DECF   DISPONCNT,F   ; Decrement display ON counter
015F 0743      00529      BTFSS  STATUS,Z     ; Skip if displays should be OFF
0160 0B9A      00530      GOTO   finish_update ; Displays are ON - jump to finish updates
0161 0419      00531      BCF    FLAGS,0      ; Restore the mode to displays OFF
0162 0439      00532      BCF    FLAGS,1      ; Restore the mode to displays OFF
0163 0066      00533      CLRF   PORTB        ; Clear out segment drives on PORTB
0164 0065      00534      CLRF   PORTA        ; Clear out common digit drives on PORTA
0165 0B9A      00535      GOTO   finish_update ; Jump to finish updates
0166           00536 serve_up_key
0166 090D      00537      call    task_scan     ; Scan the next LED digit.
0167 0619      00538      BTFSC  FLAGS,0      ; Skip if not in TIMER or CLOCK mode
0168 0B6D      00539      GOTO   no_up_display ; Currently in TIMER or CLOCK - keep mode
0169 0639      00540      BTFSC  FLAGS,1      ; Skip if not in ALARM mode
016A 0B6D      00541      GOTO   no_up_display ; Currently in ALARM - keep mode
016B 0519      00542      BSF    FLAGS,0      ; Set to CLOCK mode
016C 0539      00543      BSF    FLAGS,1      ; Set to CLOCK mode
016D           00544 no_up_display
016D 007F      00545      CLRF   ALARMCNT     ; A key was pressed, so turn off alarm
016E 0953      00546      call    turnon_scan  ; Turn on the LEDs
016F 0798      00547      BTFSS  MODEKEY      ; Skip if MODE is pressed as well
0170 0B9A      00548      GOTO   finish_update ; MODE is not pressed - jump to finish update
0171 021E      00549      MOVF   MODE_COUNT,W ; Update STATUS Z bit for mode count
0172 0743      00550      BTFSS  STATUS,Z     ; Skip if we have counted down to zero
0173 00FE      00551      DECF   MODE_COUNT,F ; Decrement the mode count
0174 090D      00552      call    task_scan     ; Scan the next LED digit.
0175 021E      00553      MOVF   MODE_COUNT,W ; Update the Z bit to check for zero
0176 0743      00554      BTFSS  STATUS,Z     ; Skip if we have incremented for 7 times
0177 0B7C      00555      GOTO   serve_min_up  ; Increment the minutes digits
0178 00D9      00556      DECF   FLAGS,W      ; Place current mode into W
0179 0900      00557      CALL   mode_timer    ; Look-up register RAM address for current mode
017A 09BA      00558      CALL   inc_hour_ld   ; Add one hour to the current display
017B 0B9A      00559      GOTO   finish_update ; Jump to finish updates
017C           00560 serve_min_up
017C 090D      00561      call    task_scan     ; Scan the next LED digit.
017D 00D9      00562      DECF   FLAGS,W      ; Place current mode into W
017E 0900      00563      CALL   mode_timer    ; Look-up register RAM address for current mode
017F 09A7      00564      CALL   inc_min_ld    ; Add one minute to the current display
0180 0B9A      00565      GOTO   finish_update ; Jump to finish updates
0181           00566 serve_down_key
0181 090D      00567      call    task_scan     ; Scan the next LED digit.
0182 0619      00568      BTFSC  FLAGS,0      ; Skip if not in TIMER or CLOCK mode
0183 0B88      00569      GOTO   no_dn_display ; Currently in TIMER or CLOCK - keep mode
0184 0639      00570      BTFSC  FLAGS,1      ; Skip if not in ALARM mode
0185 0B88      00571      GOTO   no_dn_display ; Currently in ALARM - keep mode
0186 0519      00572      BSF    FLAGS,0      ; Set to CLOCK mode
0187 0539      00573      BSF    FLAGS,1      ; Set to CLOCK mode
0188           00574 no_dn_display
0188 007F      00575      CLRF   ALARMCNT     ; A key was pressed, so turn off alarm
0189 0953      00576      CALL   turnon_scan  ; Turn on the LEDs
018A 0798      00577      BTFSS  MODEKEY      ; Skip if MODE is pressed as well
018B 0B9A      00578      GOTO   finish_update ; MODE is not pressed - jump to finish update
018C 021E      00579      MOVF   MODE_COUNT,W ; Update STATUS Z bit for mode count
018D 0743      00580      BTFSS  STATUS,Z     ; Skip if we have counted down to zero
018E 00FE      00581      DECF   MODE_COUNT,F ; Decrement the mode count

```

AN615

```
00582
018F 090D 00583      call    task_scan      ; Scan the next LED digit.
0190 021E 00584      MOVF   MODE_COUNT,W   ; Update the Z bit to check for zero
0191 0743 00585      BTFSS  STATUS,Z      ; Skip if we have incremented for 7 times
0192 0B97 00586      GOTO   serve_min_down ; Decrement the minutes digits
0193 00D9 00587      DECF   FLAGS,W       ; Place current mode into W
0194 0900 00588      CALL   mode_timer    ; Look-up register RAM address for current mode
0195 09E5 00589      CALL   dec_hour_ld   ; Subtract one hour from the current display
0196 0B9A 00590      GOTO   finish_update ; Jump to finish updates
0197      00591  serve_min_down
0197 00D9 00592      DECF   FLAGS,W       ; Place current mode into W
0198 0900 00593      CALL   mode_timer    ; Look-up register RAM address for current mode
0199 09E6 00594      CALL   dec_min_ld    ; Subtract one minute from the current display
019A      00595  finish_update
019A 090D 00596      call    task_scan      ; Scan the next LED digit.
019B 0619 00597      BTFSC  FLAGS,0       ; Skip if in mode OFF or ALARM
019C 0BA4 00598      GOTO   new_display   ; Jump to update LED display registers
019D 0639 00599      BTFSC  FLAGS,1       ; Skip if in mode OFF
019E 0BA4 00600      GOTO   new_display   ; Jump to update LED display registers
019F 0067 00601      CLRF   DISPSEGS_A    ; Clear display regs to Shut off LED display
01A0 0068 00602      CLRF   DISPSEGS_B    ; Clear display regs to Shut off LED display
01A1 0069 00603      CLRF   DISPSEGS_C    ; Clear display regs to Shut off LED display
01A2 006A 00604      CLRF   DISPSEGS_D    ; Clear display regs to Shut off LED display
01A3 0B1E 00605      GOTO   main_loop     ; We are done - go back and do it again!
01A4      00606  new_display
01A4 00D9 00607      DECF   FLAGS,W       ; Move current mode state into W
01A5 0900 00608      CALL   mode_timer    ; Look-up register address of value to display
01A6 091F 00609      CALL   disp_value    ; Update display registers with new values
01A7 0B1E 00610      GOTO   main_loop     ; We are done - go back and do it again!
00611
00612 ; *****
00613 ; * Set up and initialize the processor *
00614 ; *****
01A8      00615  init
01A8 0C03 00616      MOVLW  OPTION_SETUP  ; Place option reg setup into W
01A9 0002 00617      OPTION ; Set up OPTION register
01AA 0C05 00618      MOVLW  PORTA         ; Place beginning of RAM/Port location into W
01AB 0024 00619      MOVWF  FSR           ; Now initialize FSR with this location
01AC      00620  clear_mem
01AC 0060 00621      CLRF   INDADDR       ; Clear the FSR pointed memory location
01AD 03E4 00622      INCFSZ FSR,F        ; Point to the next location
01AE 0BAC 00623      GOTO   clear_mem     ; Jump back to clear memory routine
01AF 0572 00624      BSF   ALM_HOUR_LD,3  ; Place 8:00 into alarm register
01B0 02AE 00625      INCF  CLK_HOUR_LD,F  ; Place 1:00 into clock register
01B1 0CEE 00626      MOVLW  0EEh         ; Turn on display A scan line, others off
01B2 003B 00627      MOVWF  PREVSCAN     ;
01B3 0040 00628      CLRW                   ;
01B4 0006 00629      TRIS  PORTB         ; Make all Port B pins outputs.
01B5 0005 00630      TRIS  PORTA         ; Make all Port A pins outputs.
01B6 0539 00631      BSF   FLAGS,1       ; Set up current mode to CLOCK, display ON
01B7 0519 00632      BSF   FLAGS,0       ;
01B8 04A3 00633      BCF   ALARMOK       ; Don't want to trigger alarms
01B9 04C3 00634      BCF   EGGOK         ;
01BA 059D 00635      BSF   DISPON        ; Turn on the displays
01BB      00636  mfg_checkkey
01BB 0966 00637      CALL   scan_keys    ; Lets see what is pressed
01BC 07D8 00638      BTFSS  UPKEY        ; Goto self-test if UP key is pressed at pwr up
01BD 0B1E 00639      GOTO   main_loop     ; Normal operation - Jump to the main loop
00640
00641 ; *****
00642 ; * Self-test code for manufacturing only - test buttons and LEDs *
00643 ; *****
01BE      00644  mfg_selftest
01BE 0C70 00645      MOVLW  b'01110000'  ; Place all key on pattern into W
01BF 002D 00646      MOVWF  CLK_MIN_HD   ; Use CLK_MIN_HD for keystuck ON test
01C0 006F 00647      CLRF   CLK_HOUR_HD  ; Use CLK_HOUR_HD for keystuck OFF test
```



```

01C1          00648 mfg_display
01C1 020B    00649          MOVF   CLK_SEC,W           ; Current segment display count -> W
01C2 0949    00650          CALL  mfg_led_lookup ; Look-up the next segment pattern to display
01C3 0026    00651          MOVWF PORTB        ; Move the pattern to PORT B to display it
01C4          00652 mfg_timer
01C4 0201    00653          MOVF   TMR0,W       ; Place current TMR0 value into W
01C5 019A    00654          XORWF  PREV_TMR0,W  ; Lets see which bits have changed...
01C6 003C    00655          MOVWF  TEMP         ; All changed bits are placed in temp for test
01C7 01BA    00656          XORWF  PREV_TMR0,F  ; Update Previous TMR0 value.
01C8 07FC    00657          BTFSS  TEMP,7       ; Skip if it is not time to increment second
01C9 0BC4    00658          GOTO   mfg_timer    ; It is not time to move to next digit - go back
01CA 02AB    00659          INCF   CLK_SEC,F    ; Move to the next display pattern
01CB          00660 mfg_check_digit
01CB 07AB    00661          BTFSS  CLK_SEC,5    ; Skip if we have timed out waiting for button
01CC 0BD5    00662          GOTO   mfg_doneclk ; Jump to check for the next button press
01CD          00663 mfg_nextdigit
01CD 006B    00664          CLRF   CLK_SEC      ; Clear out timer
01CE 0906    00665          CALL  buzz_now      ; Send out a buzzer beep!
01CF 077B    00666          BTFSS  PREVSCAN,3   ; Skip if we have NOT tested the last digit
01D0 0BE5    00667          GOTO   finish_mfg_test ; Jump to the end after last digit tested
01D1 035B    00668          RLF   PREVSCAN,W    ; Select the next digit through a rotate..
01D2 037B    00669          RLF   PREVSCAN,F    ;
01D3 021B    00670          MOVF   PREVSCAN,W   ; Place next digit select into W
01D4 0025    00671          MOVWF  PORTA        ; Update port A to select next digit
01D5          00672 mfg_doneclk
01D5 0966    00673          CALL  scan_keys     ; Scan the keys to see what is pressed...
01D6 0218    00674          MOVF   KEYPAT,W     ; Place pattern into W
01D7 016D    00675          ANDWF  CLK_MIN_HD,F ; Make shure keys are not stuck ON
01D8 012F    00676          IORWF  CLK_HOUR_HD,F ; Make shure each key is pressed at least once
01D9 077B    00677          BTFSS  PREVSCAN,3   ; Skip if we are NOT at the last digit
01DA 05F8    00678          BSF   KEYPAT,7      ; Set flag bit to indicate we are done!
01DB 0C08    00679          MOVLW  .8           ; Place 8 into W
01DC 008B    00680          SUBWF  CLK_SEC,W    ; CLK_SEC - W => W
01DD 0703    00681          BTFSS  STATUS,C     ;
01DE 0078    00682          CLRF   KEYPAT       ;
01DF 03B8    00683          SWAPF  KEYPAT,F     ;
01E0 025B    00684          COMF   PREVSCAN,W   ;
01E1 0158    00685          ANDWF  KEYPAT,W     ;
01E2 0743    00686          BTFSS  STATUS,Z     ;
01E3 0BCD    00687          GOTO   mfg_nextdigit
01E4 0BC1    00688          GOTO   mfg_display
01E5          00689 finish_mfg_test
01E5 022D    00690          MOVF   CLK_MIN_HD,F ;
01E6 0743    00691          BTFSS  STATUS,Z     ;
01E7 0BEF    00692          GOTO   bad_switch
01E8 020F    00693          MOVF   CLK_HOUR_HD,W ;
01E9 0F70    00694          XORLW  070h         ;
01EA 0743    00695          BTFSS  STATUS,Z     ;
01EB 0BEF    00696          GOTO   bad_switch
01EC          00697 mfg_cleanup
01EC 006F    00698          CLRF   CLK_HOUR_HD  ; Restore temp registers to zero
01ED 006D    00699          CLRF   CLK_MIN_HD   ; Restore temp registers to zero
01EE 0B1E    00700          GOTO   main_loop    ; Jump to main loop
01EF          00701 bad_switch
01EF 026D    00702          COMF   CLK_MIN_HD,F ;
01F0 038D    00703          SWAPF  CLK_MIN_HD,W ;
01F1 0038    00704          MOVWF  KEYPAT       ;
01F2 05EF    00705          BSF   CLK_HOUR_HD,7 ;
01F3 038F    00706          SWAPF  CLK_HOUR_HD,W ;
01F4 0178    00707          ANDWF  KEYPAT,F     ;
01F5 0C7F    00708          MOVLW  07Fh         ;
01F6 0026    00709          MOVWF  PORTB        ;
01F7 006C    00710          CLRF   CLK_MIN_LD   ;
01F8 05AC    00711          BSF   CLK_MIN_LD,5  ;
01F9          00712 loop_bad_sw
01F9 0907    00713          CALL  buzz_now_dispon ; Beep the buzzer constantly for a few secs

```

AN615

```
01FA 02EC      00714      DECFSZ  CLK_MIN_LD,F      ; Decriment counter and skip when done
01FB 0BF9      00715      GOTO    loop_bad_sw      ; Not done buzzing - go back and do it again
01FC 0BEC      00716      GOTO    mfg_cleanup      ; Done buzzing - clean-up and run clock
                00717      END
```

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```
0000 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX--X
0F80 : -----
0FC0 : -----X
```

All other memory blocks unused.

```
Errors      :      0
Warnings    :      0
Messages    :      0
```

Digital Signal Processing with the PIC16C74

Author: *Darius Mostowfi*
Design Consultant

INTRODUCTION

The use of general purpose microcontrollers for low-end digital signal processing applications has become more commonplace these days with the availability of higher speed processors. Since most signal processing systems consist of a host processor and dedicated DSP chip, the use of a single microcontroller to perform both these functions provides a simpler and lower cost solution. In addition, the single chip design will consume less power which is ideal for battery powered applications. The PIC16C74 with its on-chip A/D, PWM module, and fast CPU is an ideal candidate for use in these low-bandwidth signal processing applications.

A typical signal processing system includes an A/D converter, D/A converter, and CPU that performs the signal processing algorithm as shown in Figure 1.

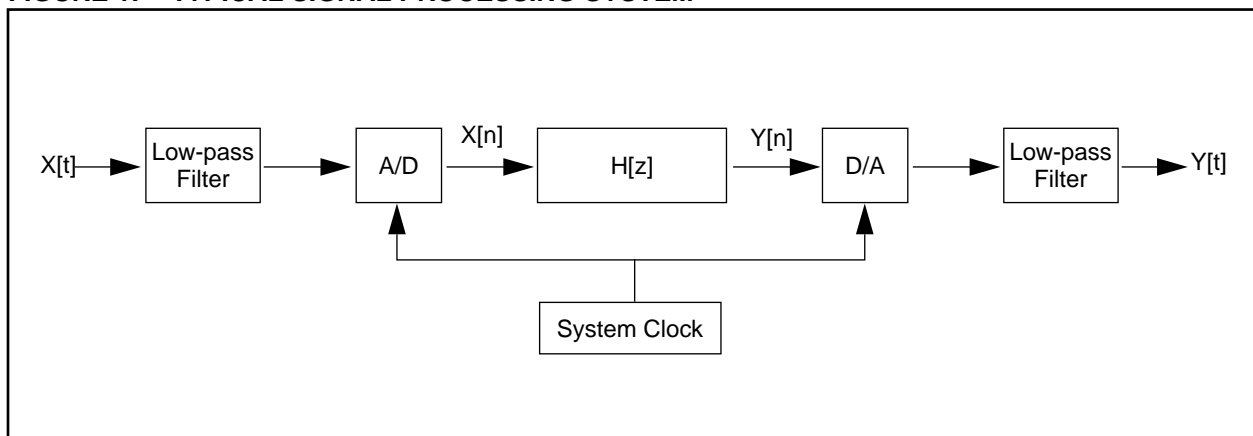
The input signal, $x(t)$, is first passed through an input filter (commonly called the anti-aliasing filter) whose function is to bandlimit the signal to below the Nyquist rate (one half the sampling frequency) to prevent aliasing. The signal is then digitized by the A/D converter at a rate determined by the sample clock to produce $x(n)$, the discrete-time input sequence. The system transfer function, $H(z)$, is typically implemented in the time-domain using a difference equation. The

output sample, $y(n)$, is then converted back into the continuous-time signal, $y(t)$, by the D/A converter and output low-pass filter.

The calculation of the output signal using a difference equation requires a multiply and accumulate (MAC) operation. This is typically a single-cycle instruction on DSP chips but can take many cycles to perform on a standard microcontroller since it must be implemented in code. Since the digitization of the signal, calculation of the output, and output to the D/A converter all must be completed within the sample clock period, the speed at which this can be done determines the maximum bandwidth that can be achieved with the system. The relatively slow speed of most microcontrollers is the major limitation when they are used in DSP applications but the PIC16C74's fast instruction execution speed (as fast as 200 ns/instruction) can provide the performance required to implement relatively low bandwidth systems. In addition, the device's on-chip A/D and PWM modules provide all the functions needed for a single chip system. Only a few external components are needed to use the PIC16C74 for tone generation, filtering of transducer signals, or low bandwidth control.

This application note describes the basic issues that need to be addressed in order to implement digital signal processing systems using the PIC16C74 and provides application code modules and examples for DTMF tone generation, a 60 Hz notch filter, and a simple PID compensator for control systems. These routines can also be used with other PIC16C6X and PIC16C7X processors with minor modifications and the addition of external analog I/O devices.

FIGURE 1: TYPICAL SIGNAL PROCESSING SYSTEM



CODE DEVELOPMENT TOOLS

The code for these applications was written using Byte Craft's MPC C compiler. The MPC compiler provides an Integrated Development Environment (IDE) and generates highly optimized code for the entire PIC16CXX/17CXX family. For new PIC16CXX/17CXX users that are familiar with C, this is an ideal way to quickly develop code for these processors. In addition, the listing files can be studied in order to learn the details of PIC16CXX/17CXX assembly language. The modules and examples for this application note use C for the main program body and in-line assembly language for the time-critical routines. MPC provides interrupt support so that interrupt service routines (ISRs) can be easily written in either C or assembly. This feature was used to provide a timer ISR for one of the code modules. The compiler proved to be a valuable tool that allowed both high level and assembly language routines to be written and tested quickly.

In order to provide the double precision math functions required for this application note, a couple of existing math functions written for the PIC16C54 (see AN525) were converted for use with MPC. The double precision multiply and addition routines were modified by first changing all RAM declarations done in EQU statements to C "unsigned char" variable declarations. The main body of assembly language code was preceded by "#asm" and ended by "#endasm" preprocessor directives which tell the compiler where the in-line assembly code starts and ends. Finally, any macro sections and register names that are defined differently in MPC were changed.

The assembly language routines for tone generation and filtering were also written as C functions using the compiler. Assembly language routines written in this way can be called directly from other assembly language modules or called directly from C by using the label name as a C function. Source listings for all the modules and example programs can be found in the appendices at the end of this application note. These modules can be directly compiled using the MPC compiler or, alternatively, the assembly language sections can be used with MPASM with minor modifications.

Number Representation and Math Routines

One of the challenges of using any general purpose microcontroller for signal processing algorithms is in implementing the finite word-length arithmetic required to perform the calculations. As mentioned before, the speed at which the MAC operations can be performed limits the maximum achievable bandwidth of the system. Therefore, the routines that perform the multiplication and the main signal processing algorithms need to be optimized for speed in order to obtain the highest possible bandwidth when using the PIC16C74.

The selection of word size and coefficient scaling are also important factors in the successful implementation of signal processing systems. The effects of using a fixed word length to represent the signal and do calculations fall into three categories: signal quantization, round-off error, and coefficient quantization. The signal quantization due to the A/D converter and round-off error due to the finite precision arithmetic affect the overall signal-to-noise performance of the system. Scaling of the input signal should be done before the A/D converter to use the full input range and maximize the input signal-to-noise ratio. The use of double precision math for all calculations and storing intermediate results, even if the input and output signals are represented as 8-bit words, will help to reduce the round-off error noise to acceptable levels. Coefficient quantization occurs when the calculated coefficients are truncated or rounded off to fit within the given word length. This has the effect of moving the system transfer function poles and zeros which can change the system gain, critical frequencies of filters, or stability of the system. The successful implementation of these systems requires careful design and modeling of these effects using one of the many software programs that are available. The code written for this application note was first modeled using PC MATLAB before being implemented on the PIC16C74.

The algorithms in this application note are all implemented using fixed point two's complement arithmetic. Two math libraries were used for the examples: one 8-bit signed multiply routine that was written specifically for the tone generation algorithm, and the modified double precision routines for the PIC16C54 that were used in the filtering routine. All numbers are stored in fractional two's complement format where the MSB is the sign bit and there is an implied decimal point right after it. This is commonly referred to as Qx format where the number after the Q represents the number of fractional bits in the word. For instance, 16 bit words with the decimal point after the MSB would be referred to as Q15. This format allows numbers over the range of -1 to 0.99 to be represented and, because the magnitude of all numbers is less than or equal to one, has the advantage that there can be no overflow from a multiplication operation.

Since calculations are done using two's complement arithmetic, values read by the PIC16C74's A/D converter need to be converted to this format. This can be easily done if the input is set up to read values in offset binary format. In this representation, the most negative input voltage is assigned to the number 0, zero volts is assigned the number 128, and the most positive voltage input is assigned 255. Since the PIC16C74 has a unipolar input A/D converter, a bipolar input signal must be scaled to be between 0 and 5V. One way to accomplish this is to use an op-amp scaling and offset circuit. The signal should be centered at 2.5V and have a peak to peak voltage swing of

4 to 4.5V. The offset binary number can be converted to two's complement format by simply complementing the MSB of the word. Once the signal processing calculations are completed, the number can be converted back to offset binary by complementing the MSB before it is written to the PWM module. A similar level shifting circuit can be used at the PWM output to restore the DC level of the signal. Using this technique allows a wide range of analog input voltages to be handled by the PIC16C74.

A/D and D/A Conversion

The PIC16C74's internal 8-bit A/D converter and PWM modules can be used to implement analog I/O for the system. The A/D converter along with an external anti-aliasing filter provides the analog input for the system. Depending on the input signal bandwidth and the sampling frequency, the filter can be a simple single pole RC filter or a multiple pole active filter. The PWM output along with an external output "smoothing" filter provides the D/A output for the system. This can be a simple RC filter if the PWM frequency is much higher (five to ten times) than the analog signal that is being output. Alternatively, an active filter can also be used at the PWM output. Since the use of the A/D and PWM modules is covered in detail in the data sheet for the part, they will not be covered here. In addition, since the PIC16C74's A/D converter is similar to the PIC16C71 and the PWM module is the same as the PIC16C74, the use of these is also covered in application notes AN546, AN538, and AN539.

Appendix A contains the listing for the C module "ANALOGIO.C" that has the functions that read the A/D converter input, initialize the PWM module, and write 8-bit values to the PWM module. The number format (offset binary or two's complement) for the A/D and PWM values as well as the PWM resolution and mode are set using "#define" pragmas at the beginning of the module. The get_sample() function takes the A/D input multiplexer channel number as an argument and returns the measured input value. The init_PWM() function takes the PWM period register PR2 value as an argument. The write_PWM() function takes the output values for PWM module 1 and 2 and writes them to the appropriate registers using the specified resolution. If the second argument to the function is 0, the registers for PWM module 2 are unaffected. The PWM resolution is always 8-bits but the mode used depends on the PWM frequency.

The A/D conversions need to be performed at the system sample rate which requires that some form of sample clock be generated internally or input from an external source. One way to generate this clock internally, in software with minimal effort, is to use the Timer2 interrupt. Since Timer2 is used to generate the PWM period, enabling the Timer2 interrupt and using the Timer2 postscaler can generate an interrupt at periods that are integer divisors of the PWM period. The ISR can set a software "sample flag" that is checked by the main routine. Once the sample flag is

asserted by the ISR, the main routine can then clear it and perform the signal processing operation, output the next sample, and then wait for the sample flag to be asserted true again. Alternatively, a separate timer/counter or external clock input can be used for the system sample clock. The latter two methods have the advantage that the PWM frequency can be set independent of the sampling period. For best results, the PWM frequency should be set for at least five times the maximum frequency of the analog signal that is being reproduced. The example programs illustrate the use of both of the methods for generating an internal sample clock.

Tone Generation

For systems that need to provide audible feedback or to provide DTMF signaling for telecom applications, the PIC16C74's PWM module can be used to generate these signals. One way to do this is to output samples of a sinusoidal waveform to the PWM module at the system sampling rate. This method is relatively simple but is limited to single tones and may require large amounts of memory depending on the number of samples used per cycle of the waveform and the number of tones that need to be generated. A more efficient method of generating both single and dual-tone signals is to use a difference equation method. This method uses a difference equation that is derived from the z-transform of a sinusoid as follows:

The z-transform of a sinusoid is

$$\frac{z^{-1}\sin\omega T}{1 - 2z^{-1}\cos\omega T + z^{-2}}$$

where the period $\omega = 2\pi f$ and T is the sampling period.

If this is interpreted as the transfer function $H(z) = Y(z)/X(z)$ then the difference equation can be found taking the inverse z-transform and applying the associated shift theorem as follows:

rearranging:

$$Y(z)(1 - 2z^{-1}\cos\omega T + z^{-2}) = X(z)(z^{-1}\sin\omega T)$$

$$Y(z) = z^{-1}X(z)\sin\omega T + z^{-1}Y(z)2\cos\omega T - z^{-2}Y(z)$$

taking the inverse z-transform:

$$Z^{-1}[Y(z)] = Z^{-1}[z^{-1}X(z)\sin\omega T + z^{-1}Y(z)2\cos\omega T - z^{-2}Y(z)]$$

$$y(n) = \sin\omega T x(n - 1) + 2\cos\omega T y(n - 1) - y(n - 2)$$

If we let $a = \sin\omega T$ and $b = \cos\omega T$, the equation can be written as:

$$y(n) = a x(n - 1) + 2b y(n - 1) - y(n - 2)$$

AN616

thus we have a difference equation with coefficients a and b . Note that only two coefficients are needed to generate a sinusoidal output sequence. These are calculated from the relationship above and stored in memory for use by the tone generation algorithm.

If we input an impulse to this system ($x(n) = 1$ at $n = 0$ and is zero elsewhere) then the output of the system will be a discrete-time sinusoidal sequence. Note that at $n = 0$, the output will always be 0 and $x(n)$ is only 1 at $n = 1$ so the sequence becomes:

$$\begin{aligned}y(0) &= 0 \\y(1) &= a \\y(n) &= 2b y(n - 1) - y(n - 2) \\&\text{for } n \text{ equal to or greater than } 2\end{aligned}$$

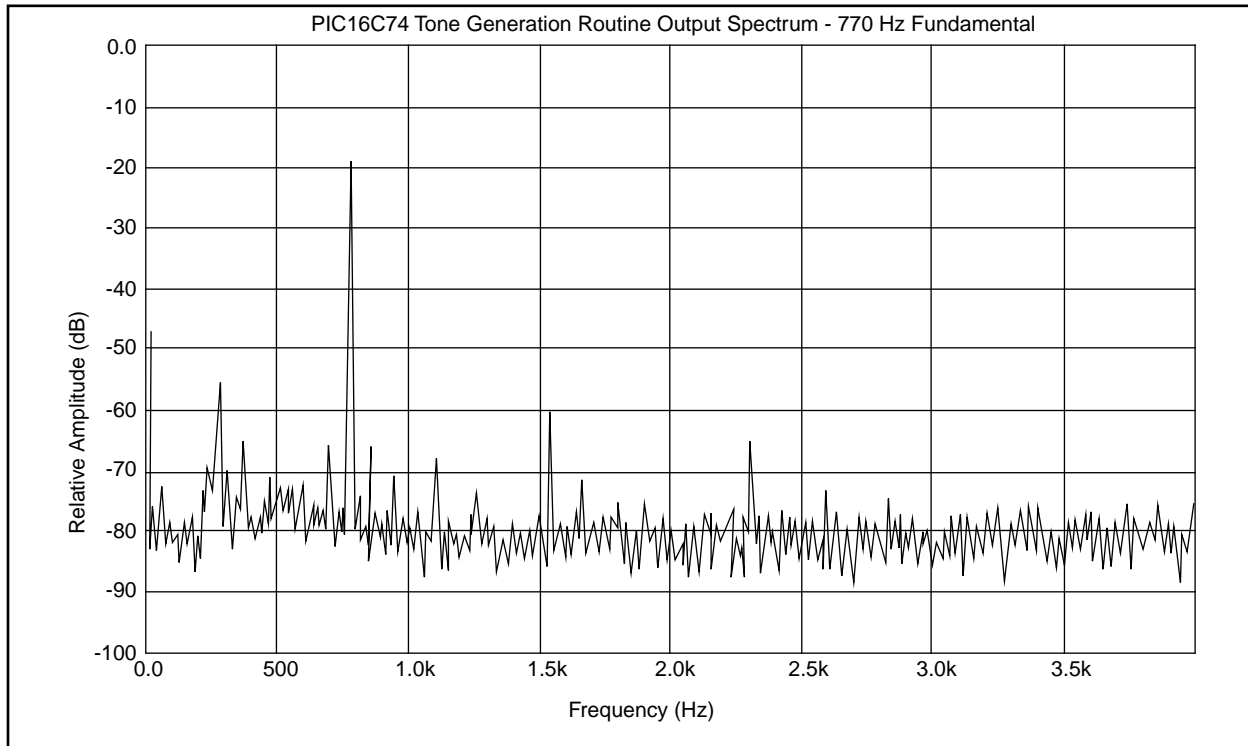
In order to further simplify the implementation of the algorithm, we can omit the first sample period. Since the output is already at 0 before starting, this will make no difference in the final output other than the fact that it will be time shifted by one sample. To generate dual tones, the algorithm is executed once for each tone and the two output samples are summed together. Since the output must be calculated and output to the D/A each sample period, a limitation exists on the frequency of the tone that can be produced for a given sample rate and processor speed. The higher the ratio of the sample clock to the tone frequency, the better, but a sample rate of at least three to four times the highest tone output should produce a sine wave with acceptable distortion.

Appendix B contains the listing for the "PICTONE.C" module which uses the difference equation method to produce variable length tones from the PWM module. Timer2 is used to generate the PWM period as well as the sample clock and tone duration timer. To send a tone, the coefficients and duration are written to the appropriate variables and then the tone routine is called. If the $a2$ and $b2$ coefficients are cleared, the routine will only generate a single tone sequence. The difference equation algorithm uses 8-bit signed math routines for the multiply operations. Using 8-bit coefficients reduces the accuracy by which the tones can be generated but greatly reduces the number of processor cycles needed to perform the algorithm since only single precision arithmetic is used. The spectrum of a single tone signal generated using this routine is shown in Figure 2.

Note that the second harmonic is better than 40 dB below the fundamental. Accuracy of this particular tone is better than 0.5%.

An example program "DTMFGEN.C" illustrates the use of the tone module to generate the 16 standard DTMF tones used for dialing on the telephone system. A sampling rate of 6.5 kHz was used which allows dual tones to be generated on a processor running at 10 MHz. Accuracy with respect to the standard DTMF frequencies is better than 1% for all tones and all harmonics above the fundamental frequency are greater than 30 dB down.

FIGURE 2: SINGLE TONE SIGNAL



Digital Filters

Digital filters with critical frequencies up to a kilohertz or so can be implemented on the PIC16C74. Digital filters fall into two classes: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. FIR filters require more coefficients and multiplication operations to implement practical filters and are not as well suited for implementation on the PIC16C74. IIR type filters are typically designed starting with an analog filter prototype and then performing an analog to digital transformation to produce the digital filter coefficients. The subject of digital filter design is not within the scope of this application note but there are many excellent texts that cover the theory and design of these filters.

The implementation of a second-order IIR filter is done by using a second-order difference equation. A second-order infinite impulse response (IIR) filter has a transfer function of the form:

$$H(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{1 + a_1z^{-1} + a_2z^{-2}}$$

Where a_1 , a_2 , b_0 , b_1 , and b_2 are the coefficients of the polynomials of the system transfer function that, when factored, yield the system poles and zeros. The difference equation found by taking the inverse z-transform and applying the shift theorem is:

$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) - a_1y(n-1) - a_2y(n-2)$$

Since the transfer function coefficients are used directly in the difference equation, this is often called the "Direct Form I" implementation of a digital filter. This form has its limitations due to numerical accuracy issues but is effective for implementing second-order systems.

Appendix C contains the listing for the general-purpose filter routine "IIR_FILT.C" that can be used to implement low-pass, high-pass, bandpass, and bandstop (notch) filters. The filter() function takes an 8-bit input value $x(n)$ and calculates the output value $y(n)$. The filter coefficients are stored as 16-bit two's complement numbers and computation of the output is done using double precision arithmetic. Since the coefficients generated from the filter design program will be in decimal form, they need to be scaled to be less than 1 and then multiplied by 32,768 to put them in Q15 format. Additional scaling by factors of two may be required to prevent overflow of the sum during calculations. If this is done, the output must be multiplied by this scale factor to account for this. The "IIR_FILT.C" module contains two other subroutines required for the filtering program. One of these is a decimal adjust subroutine to restore the decimal place after two 16-bit Q15 numbers are multiplied. The subroutine shifts the 32-bit result left by one to get rid of the extra sign bit.

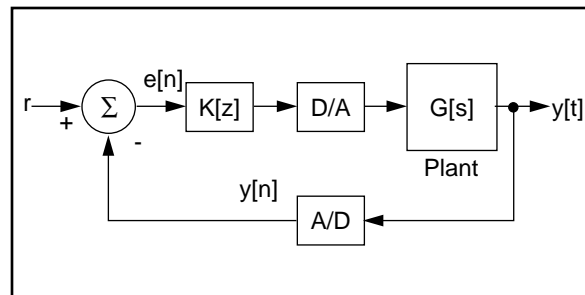
The other routine scales the output by factors of two and is used after the output of the filter has been calculated to account for the scaling of the coefficients.

An example program "NOTCH_60.C" is provided that illustrates the implementation of a 60 Hz notch filter using the "IIR_FILT.C" module. The filter was modeled and designed using PC MATLAB before being implemented on the PIC16C74. A sample rate of 1 kHz is used which means that signals up to a few hundred hertz can be processed. The filter provides an attenuation of about 40 dB at 60 Hz and can be used to remove interference from sensor signals in a system.

Digital Control

A low bandwidth digital control system can be implemented on the PIC16C74 using the analog I/O and IIR filter routines. A typical digital control system is shown below:

FIGURE 3: TYPICAL DIGITAL CONTROL SYSTEM



The input, r , is the reference input and $y(t)$ is the continuous-time output of the system. $G(s)$ is the analog transfer function of the plant (controlled system) and $K(z)$ is the digital compensator. The error signal is calculated by subtracting the measured output signal, $y(n)$, from the reference. The controller transfer function is essentially a filter that is implemented in the time-domain using a difference equation. Since digital control system design is a complex subject and the design of a suitable compensator depends on the system being controlled and the performance specifications, only the implementation issues will be discussed.

AN616

One popular and well understood compensator is the Proportional-Integral-Derivative (PID) controller whose transfer function is of the form:

$$K(z) = K_P + \frac{K_I}{1 - z^{-1}} + K_D(1 - z^{-1})$$

Where K_P is the proportional gain, K_I is the integral gain, and K_D is the derivative gain. The transfer function can be implemented directly or can be put in the form of a standard second-order difference equation from the modified transfer function as shown below:

$$H(z) = \frac{(K_I T^2 + K_P T + K_D) - (2K_D + K_P T)z^{-1} + K_D z^{-2}}{T(1 - z^{-1})}$$

$$y(n) = (K_P + K_I T + \frac{K_D}{T})x(n) - (K_P + \frac{2K_D}{T})x(n - 1) + \frac{K_D}{T}x(n - 2) - y(n - 1)$$

Since the numerator coefficients will be greater than one, a gain factor K needs to be factored out so that the resulting coefficients are less than one. In this way, the IIR filter routine can be used to implement the controller. After the filter routine, the output y needs to be multiplied by K before being output to the PWM module. Since the gain can be high, this result needs to be checked for overflow and limited to the maximum 8-bit value, if required. Saturating the final result prevents the system from going unstable if overflow in the math does occur. The gains can also be applied externally at the D/A output. For example, the PWM can drive a power op-amp driver that provides a +/- 20 volt swing for a DC motor.

RESULTS AND CONCLUSION

The results obtained using the PIC16C74 in these applications were impressive. The tone generation routines produce very clean sinusoidal signals and DTMF tones generated using this routine have been used to dial numbers over the telephone system with excellent results. In addition, tones used for audible feedback are more pleasing to the ear than those generated from a port pin as is typically done on processors without PWM modules. Using the PIC16C74 to generate these tones eliminates the need for special DTMF generator IC's thus reducing the cost and simplifying the design. The tone routine requires approximately 125 instruction cycles to calculate an output sample for a single tone output and 230 instruction cycles to calculate an output sample for a dual tone output.

The IIR filtering routines produce good results and have been used to filter 60 Hz signals on sensor lines and also to implement a simple PID controller system with excellent results. The IIR routine takes approximately 1670 instruction cycles to calculate the output. Table 1 shows the performance that can be expected with the PIC16C74 for various processor speeds.

In conclusion, the PIC16C74 provides the necessary performance to provide these simple, low bandwidth signal processing operations. This means that products using this device can benefit from cost and power savings by eliminating specialized components that normally perform these tasks.

References

- Antoniou, A. Digital Filters: Analysis and Design. NY: McGraw-Hill Book Co., 1979.
- Openheim, A.V. and Schaffer, R.W. Digital Signal Processing. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1975.

TABLE 1: PIC16C74 IIR FILTER PERFORMANCE

	4 MHz	8 MHz	10 MHz	16 MHz	20 MHz
A/D Input (35 cycles + 15 ms)	50 μ s	32.5	29	23.75	22
IIR Filter (1850 cycles)	1850	925	740	462.5	370
PWM Output (62 cycles)	62	31	24.8	15.5	12.4
Total	1962	988.5	793.8	501.75	368.4
Max. Sampling Frequency	~500 Hz	~1000 Hz	~1250 Hz	~2000 Hz	~2500 Hz

NOTES:

AN616

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: ANALOG I/O MODULE

```

/*****
* Analog I/O Module
*
* Written for "Digital Signal Processing with the PIC16C74" Application Note
*
* This module contains functions that read the A-D inputs, initialize the PWM
* ports, and write values to the PWM ports.
*
* D. Mostowfi 4/95
*****/
#define active      1      /* define active as 1 */
#define LOW        0      /* define LOW as 0 */
#define HIGH       1      /* define HIGH as 1 */

#define OFFSET     0      /* define offset binary mode as 0 */
#define TWOS       1      /* define two's compliment mode as 1 */

#define AD_FORMAT  TWOS   /* define A-D format as TWOS */
#define PWM_FORMAT TWOS   /* define PWM format as TWOS */
#define PWM_RES    HIGH   /* define PWM resolution as HIGH */

bits FLAGS;
#define sample_flag  FLAGS.1 /* define sample_flag as FLAGS.1 */

/*****
* A-D Converter Routine - reads A-D converter inputs
*
* usage:
*   - call get_sample(channel #)
*   - returns 8 bit value
*****/
char get_sample(char channel)
{
    char i;

    ADRES=0;          /* clear ADRES */
    STATUS.C=0;      /* clear carry */
    RLCF(channel);   /* and rotate channel 3 times */
    RLCF(channel);   /* to put in proper position */
    RLCF(channel);   /* for write to ADCON0 */
    ADCON0=channel;  /* write channel to ADCON0 */
    ADCON0.0=1;      /* turn on A-D */
    i=0;
    while(i++<=5){}; /* delay (to ensure min sampling time) */
    ADCON0.2=1;      /* start conversion */
    while(ADCON0.2){ /* wait for eoc */
        ADCON0.0=0;  /* turn off a-d converter */
        if(AD_FORMAT==TWOS){ /* if format is two's compliment */
            ADRES.7=!ADRES.7; /* compliment MSB */
        }
    }
    return ADRES;    /* return value in a-d result reg */
}

/*****
* PWM Initialization Routine - sets up PR2, sets output to mid-point, and
* starts timer 2 with interrupts disabled.
*
* usage:
*   - call init_PWM(PR2 register value)
*****/
```

```

*****/
void init_PWM(char _pr2)
{
    PR2=_pr2;                /* reload value for 40khz PWM period */
    CCP1CON.5=0;            /* set CCPxCON = 0 for 50% output */
    CCP1CON.4=0;
    CCP2CON.5=0;
    CCP2CON.4=0;
    if(PWM_RES==HIGH){      /* if resolution is high, set CCPRxH=0 and */
        CCPR1H=0x00;        /* CCPRxL=0x20 for 50% PWM duty cycle */
        CCPR1L=0x20;
        CCPR2H=0x00;
        CCPR2L=0x20;
    }
    else{
        CCPR1H=0x00;        /* if resolution is low, set CCPRxH=0 and */
        CCPR1L=0x80;        /* CCPRxL=0x80 for 50% PWM duty cycle */
        CCPR2H=0x00;
        CCPR2L=0x80;
    }
    T2CON.TMR2ON=1;        /* start timer 2 */
    PIE1.TMR2IE=0;        /* and disable timer 2 interrupt */
}

/*****
* PWM Output Routine - writes output values to PWM ports
*
* Both high resolution and low resolution modes write 8 bit values - use of
* high or low resolution depends on PWM output period.
*
* usage:
* - call write_PWM(channel 1 value, channel 2 value)
*   if channel 2 value=0, PWM port 2 not written
*****/
void write_PWM(bits pwm_out1, bits pwm_out2)
{
    if(PWM_FORMAT==TWOS){  /* if format is two's compliment */
        pwm_out1.7=!pwm_out1.7; /* compliment msb's */
        pwm_out2.7=!pwm_out1.7;
    }
    if(PWM_RES==HIGH){     /* if resolution is high */
        STATUS.C=0;        /* clear carry */
        pwm_out1=RRCF(pwm_out1); /* rotate right and write two lsb's */
        CCP1CON.4=STATUS.C; /* to CCP1CON4 and CCP1CON5 */
        STATUS.C=0;
        pwm_out1=RRCF(pwm_out1);
        CCP1CON.5=STATUS.C;
        if(pwm_out2!=0){   /* if pwm_out2 not 0, do the same */
            STATUS.C=0;    /* for channel 2 */
            pwm_out2=RRCF(pwm_out2);
            CCP2CON.4=STATUS.C;
            STATUS.C=0;
            pwm_out2=RRCF(pwm_out2);
            CCP2CON.5=STATUS.C;
        }
    }
    CCP1L=pwm_out1;        /* write value to CCP1L */
    if(pwm_out2!=0){      /* if pwm_out2 not 0, do the same */
        CCP2L=pwm_out2;   /* for CCP2L */
    }
}
/* done */

```

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX B: TONE GENERATION MODULE

```
/******  
* Tone Generation Module  
*  
* Written for "Digital Signal Processing with the PIC16C74" Application Note.  
*  
* This module contains a C callable module that generates single or dual  
* tones using a difference equation method:  
*  
*  $y_1(n) = a_1 * x(n-1) + b_1 * y_1(n-1) - y_1(n-2)$   
*  $y_2(n) = a_2 * x(n-1) + b_2 * y_2(n-1) - y_2(n-2)$   
*  
* The routine is written in assembly language and uses the optimized signed  
* 8x8 multiply routine and scaling routine in the file 8BITMATH.C.  
*  
* D. Mostowfi 2/95  
*****/  
#include "\mpc\apnotes\8bitmath.c" /* 8 bit signed math routines */  
  
#define sample_flag FLAGS.1 /* sample flag */  
#define no_tone2 FLAGS.2 /* no tone 2 flag */  
  
extern char ms_cntr; /* millisecond counter for tone loop */  
  
char a1; /* first tone (low-group) coefficient 1 */  
char a2; /* first tone (low-group) coefficient 2 */  
char b1; /* second tone (high group) coefficient 1 */  
char b2; /* second tone (high group) coefficient 2 */  
char duration; /* tone duration */  
  
char y1; /* output sample  $y_1(n)$  for tone 1 */  
char y2; /* output sample  $y_2(n)$  for tone 2 */  
  
/******  
* Tone function - generates single or dual tone signals out PWM port 1.  
*  
* usage:  
* - write coefficients for tone 1 to a1 and b1  
* - write coefficients for tone 2 to a2 and b2 (0 if no tone 2)  
* - write duration of tone in milliseconds to duration  
* - call tone() function  
*****/  
void tone(void)  
{  
  
char y1_1; /*  $y_1(n-1)$  */  
char y1_2; /*  $y_1(n-2)$  */  
char y2_1; /*  $y_2(n-1)$  */  
char y2_2; /*  $y_2(n-2)$  */  
  
PIR1.TMR2IF=0; /* clear timer 2 interrupt flag */  
PIE1.TMR2IE=1; /* and enable timer 2 interrupt */  
ms_cntr=0; /* clear ms counter */  
STATUS.RP0=0; /* set proper bank!!! */  
  
#asm  
    clrf    y1          ; clear output byte and taps  
    clrf    y2          ;  
    clrf    y1_1        ;  
    clrf    y1_2        ;  
    clrf    y2_1        ;  
    clrf    y2_2        ;
```

```

        clrfs    y2_2            ;

        bcf     no_tone2        ; clear no tone 2 flag
        clrfs    ms_cntr        ; clear millisecond counter

first_sample:
        movf    a1,W            ; first iteration
        movwf   y1              ; y1(n)=a1
        movwf   y1_1            ;
        movlw   0x00            ;
        iorwf   a2,W            ;
        btfsc   STATUS,Z        ; generate second tone (a2 !=0) ?
        bsf     no_tone2        ;
        movf    a2,W            ; y2(n)=a2
        movwf   y2              ;
        movwf   y2_1            ;
        movf    y2,W            ;
        addwf   y1,F            ; y1(n)=y1(n)+y2(n) (sum two tone outputs)

tone_loop:
        movf    ms_cntr,W       ; test to see if ms=duration (done?)
        subwf   duration,W       ;
        btfsc   STATUS,Z        ;
        goto    tone_done        ;

wait_PWM:
        btfss   FLAGS,1         ; test sample flag (sample period elapsed?)
        goto    wait_PWM         ; loop if not

        bcf     FLAGS,1         ; if set, clear sample flag

#endasm
        write_PWM((char)y1,0);   /* write y1 to PWM port */
#asm

next_sample:
        movf    b1,W            ; y1(n)=b1*y1(n-1)-y1(n-2)
        movwf   multcnd         ;
        movf    y1_1,W          ;
        movwf   multplr         ;
        call    _8x8smul        ;
        call    scale_16        ;
        movf    y1_2,W          ;
        subwf   result_1,W      ;
        movwf   y1              ;
        movf    y1_1,W          ; y1(n-2)=y1(n-1)
        movwf   y1_2           ;
        movf    y1,W            ; y1(n-1)=y1(n)
        movwf   y1_1           ;
        btfsc   no_tone2        ;
        goto    tone_loop       ;
        movf    b2,W            ; y2(n)=b2*y2(n-1)-y2(n-2)
        movwf   multcnd         ;
        movf    y2_1,W          ;
        movwf   multplr         ;
        call    _8x8smul        ;
        call    scale_16        ;
        movf    y2_2,W          ;
        subwf   result_1,W      ;
        movwf   y2              ;
        movf    y2_1,W          ; y2(n-2)=y2(n-1)
        movwf   y2_2           ;
        movf    y2,W            ; y2(n-1)=y2(n)
        movwf   y2_1           ;

        movf    y2,W            ;

```

AN616

```
        addwf   y1,F           ; y1(n)=y1(n)+y2(n) (sum two tone outputs)

        goto   tone_loop     ; go and calculate next sample

tone_done:

#endasm

        CCP1CON.5=0;          /* reset PWM outputs to mid value */
        CCP1CON.4=0;
        CCP2CON.5=0;
        CCP2CON.4=0;
        CCPR1H=0x00;
        CCPR1L=0x20;
        CCPR2H=0x00;
        CCPR2L=0x20;

        PIE1.TMR2IE=0;       /* disable timer 2 interrupts */
        PIR1.TMR2IF=0;       /* and clear timer 2 interrupt flag */

}
```


Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX C: DTMF TONE GENERATION

```

/*****
* DTMF tone generation using PIC16C74
*
* Written for the "Digital Signal Processing Using the PIC16C74" Ap Note
*
* Generates 16 DTMF tones (1-9,0,*,#,A,B,C,D) out PWM port 1
*
* Uses PICTONE.C and ANALOGIO.C modules
*
* D. Mostowfi 4/95
*****/
#include "\mpc\include\delay14.h"
#include "\mpc\include\16c74.h" /* c74 header file */
#include "\mpc\math.h"

#include "\mpc\apnotes\analogio.c" /* analog I/O module */
#include "\mpc\apnotes\pictone.c" /* tone generation module */

bits pwml;

/* Function Prototypes */

void main_isr();
void timer2_isr();

/* 16C74 I/O port bit declarations */

/* global program variables */

char tmr2_cntr; /* timer 2 interrupt counter */
char delay_cntr; /* delay time counter (10ms ticks)*/

/* Tone Coefficients for DTMF Tones */

const DTMF_1[4]={30, 51, 48, 27};
const DTMF_2[4]={30, 51, 56, 19};
const DTMF_3[4]={30, 51, 64, 11};
const DTMF_4[4]={33, 48, 48, 27};
const DTMF_5[4]={33, 48, 56, 19};
const DTMF_6[4]={33, 48, 64, 11};
const DTMF_7[4]={36, 45, 48, 27};
const DTMF_8[4]={36, 45, 56, 19};
const DTMF_9[4]={36, 45, 64, 11};
const DTMF_0[4]={40, 41, 56, 19};
const DTMF_star[4]={40, 41, 48, 27};
const DTMF_pound[4]={40, 41, 64, 11};
const DTMF_A[4]={30, 51, 75, 2};
const DTMF_B[4]={33, 48, 75, 2};
const DTMF_C[4]={36, 45, 75, 2};
const DTMF_D[4]={40, 41, 75, 2};

/*****
* main_isr - 16C74 vectors to 0004h (MPC __INT() function) on any interrupt *
* assembly language routine saves W and Status registers then tests flags in
* INTCON to determine source of interrupt. Routine then calls appropriate_isr.
* Restores W and status registers when done.
*****/

```

AN616

```
void __INT(void)
{
    if(PIR1.TMR2IF){                /* timer 2 interrupt ? */
        PIR1.TMR2IF=0;             /* clear interrupt flag */
        timer2_isr();              /* and call timer 2 isr */
    }

    /* Restore W, WImage, and STATUS registers */

    #asm
        BCF STATUS,RP0             ;Bank 0
        MOVF temp_PCLATH, W
        MOVWF PCLATH               ;PCLATH restored
        MOVF temp_WImage, W
        MOVWF __WImage             ;__WImage restored
        MOVF temp_FSR, W
        MOVWF FSR                  ;FSR restored
        SWAPF temp_STATUS,W
        MOVWF STATUS               ;RP0 restored
        SWAPF temp_WREG,F
        SWAPF temp_WREG,W          ;W restored
    #endasm
}

/*****
 * timer 2 isr - provides PWM sample clock generation and millisecond counter
 * for tone routine
 *****/
void timer2_isr(void)
{
    sample_flag=active;            /* set sample flag (150us clock) */
    PORTB.7=!PORTB.7;             /* toggle PORTB.7 at sample rate */
    if(tmr2_cntr++==7){           /* check counter */
        tmr2_cntr=0;              /* reset if max */
        ms_cntr++;                /* and increment millisecond ticks */
    }
}

void main()
{
    /* initialize OPTION register */
    OPTION=0b11001111;

    /* initialize INTCON register (keep GIE inactive!) */
    INTCON=0b00000000;           /* disable all interrupts */

    /* initialize PIE1 and PIE2 registers (peripheral interrupts) */
    PIE1=0b00000000;            /* disable all interrupts */
    PIE2=0b00000000;

    /* initialize T1CON and T2CON registers */
    T1CON=0b00000000;           /* T1 not used */
    T2CON=0b00101000;           /* T2 postscaler=5 */

    /* initialize CCPxCON registers */
    CCP1CON=0b00001100;         /* set CCP1CON for PWM mode */
    CCP2CON=0b00001100;         /* set CCP2CON for PWM mode (not used in demo) */

    /* initialize SSPCON register */
    SSPCON=0b00000000;         /* serial port - not used */

    /* initialize ADCONx registers */
```

```
ADCON0=0b00000000;      /* A-D converter */
ADCON1=0b00000010;

/* initialize TRISx register (port pins as inputs or outputs) */
TRISA=0b00001111;
TRISB=0b00000000;
TRISC=0b10000000;
TRISD=0b00001111;
TRISE=0b00000000;

/* clear watchdog timer (not used) */
CLRWDT();

/* initialize program variables */
tmr2_cntr=0;

/* initialize program bit variables */
FLAGS=0b00000000;

/* initialize output port pins (display LED's on demo board) */
PORTB=0;

/* enable interrupts... */

INTCON.ADIE=1;          /* Periphereal interrupt enable */
INTCON.GIE=1;          /* global interrupt enable */

init_PWM(0x3e);        /* initialize PWM port */

PORTB=0x01;           /* write a 1 to PORTB */
a1=DTMF_1[0];        /* and send a DTMF "1" */
b1=DTMF_1[1];
a2=DTMF_1[2];
b2=DTMF_1[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x02;           /* write a 2 to PORT B */
a1=DTMF_2[0];        /* and send a DTMF "2" */
b1=DTMF_2[1];
a2=DTMF_2[2];
b2=DTMF_2[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x03;           /* write a 3 to PORTB */
a1=DTMF_3[0];        /* and send a DTMF "3" */
b1=DTMF_3[1];
a2=DTMF_3[2];
b2=DTMF_3[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x04;           /* write a 4 to PORTB */
a1=DTMF_4[0];        /* and send a DTMF "4" */
b1=DTMF_4[1];
a2=DTMF_4[2];
```

AN616

```
b2=DTMF_4[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
PORTB=0x05;          /* write a 5 to PORTB */
a1=DTMF_5[0];        /* and send a DTMF "5" */
b1=DTMF_5[1];
a2=DTMF_5[2];
b2=DTMF_5[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x06;          /* write a 6 to PORTB */
a1=DTMF_6[0];        /* and send a DTMF "6" */
b1=DTMF_6[1];
a2=DTMF_6[2];
b2=DTMF_6[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x07;          /* write a 7 to PORTB */
a1=DTMF_7[0];        /* and send a DTMF "7" */
b1=DTMF_7[1];
a2=DTMF_7[2];
b2=DTMF_7[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x08;          /* write a 8 to PORTB */
a1=DTMF_8[0];        /* and send a DTMF "8" */
b1=DTMF_8[1];
a2=DTMF_8[2];
b2=DTMF_8[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x09;          /* write a 9 to PORTB */
a1=DTMF_9[0];        /* and send a DTMF "9" */
b1=DTMF_9[1];
a2=DTMF_9[2];
b2=DTMF_9[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0;           /* write a 0 to PORTB */
a1=DTMF_0[0];        /* and send a DTMF "0" */
b1=DTMF_0[1];
a2=DTMF_0[2];
b2=DTMF_0[3];
duration=150;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0e;          /* write a 0x0e to PORTB */
```

```
al=DTMF_star[0];      /* and send a DTMF "*" */
bl=DTMF_star[1];
a2=DTMF_star[2];
b2=DTMF_star[3];
duration=250;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0f;          /* write a 0x0f to PORTB */
al=DTMF_pound[0];   /* and send a DTMF "#" */
bl=DTMF_pound[1];
a2=DTMF_pound[2];
b2=DTMF_pound[3];
duration=250;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0a;          /* write a 0x0a to PORTB */
al=DTMF_A[0];       /* and send a DTMF "A" */
bl=DTMF_A[1];
a2=DTMF_A[2];
b2=DTMF_A[3];
duration=250;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0b;          /* write a 0x0b to PORTB */
al=DTMF_B[0];       /* and send a DTMF "B" */
bl=DTMF_B[1];
a2=DTMF_B[2];
b2=DTMF_B[3];
duration=250;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0c;          /* write a 0x0c to PORTB */
al=DTMF_C[0];       /* and send a DTMF "C" */
bl=DTMF_C[1];
a2=DTMF_C[2];
b2=DTMF_C[3];
duration=250;
tone();
Delay_Ms_20MHz(200); /* delay 100ms (200/2 using MPC delays) */

PORTB=0x0d;          /* write a 0x0d to PORTB */
al=DTMF_D[0];       /* and send a DTMF "D" */
bl=DTMF_D[1];
a2=DTMF_D[2];
b2=DTMF_D[3];
duration=250;
tone();

PORTB=0;             /* write a 0 to PORTB */

while(1){           /* done (loop) */
}
}
```

AN616

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX D: IIR FILTER MODULE

```

/*****
* Second-Order IIR Filter Module
*
* Written for "Digital SIngnal Processing with the PIC16C74" Application Note.
*
* This routine implements an IIR filter using a second order difference
* equation of the form:
*
*  $y(n) = b_0*x(n)+b_1*x(n-1)+b_2*x(n-2)+a_1*y(n-1)+a_2*y(n-2)$ 
*
* D. Mostowfi 3/95
*****/
#include "\mpc\apnotes\dbl_math.c"

bits          x_n;          /* input sample x(n) */
unsigned long y_n;          /* output sample y(n) */
unsigned long x_n_1;        /* x(n-1) */
unsigned long x_n_2;        /* x(n-2) */
unsigned long y_n_1;        /* y(n-1) */
unsigned long y_n_2;        /* y(n-2) */

char          rmndr_h;      /* high byte of remainder from multiplies */
char          rmndr_l;      /* low byte of remainder from multiplies */

#define A1_H   0xd2          /* filter coefficients */
#define A1_L   0x08          /* for 60Hz notch filter */
#define A2_H   0x11          /* Fs= 1kHz */
#define A2_L   0x71
#define B0_H   0x18
#define B0_L   0xbb
#define B1_H   0xd2
#define B1_L   0x08
#define B2_H   0x18
#define B2_L   0xb9

/*****
* Filter initialization - clears all taps in memory.
*
* usage:
*   - call init_filter()
*   use at program initialization
*****/
void init_filter(){
#asm

    clrf    y_n          ; clear output value
    clrf    y_n+1        ;
    clrf    y_n_1        ; and all filter "taps"
    clrf    y_n_1+1      ;
    clrf    y_n_2        ;
    clrf    y_n_2+1      ;
    clrf    x_n_1        ;
    clrf    x_n_1+1      ;
    clrf    x_n_2        ;
    clrf    x_n_2+1      ;

#endasm
}

```

```

/*****
* Assembly language subroutines for main filter() function
*****/
#asm

;
; Add Remainder subroutine - adds remainder from multiplies to ACCc
;

add_rmndr:
    btfss    sign,7           ; check if number is negative
    goto    add_r_start      ; go to add_r_start if not
    comf    ACCcLO           ; if so, negate number in ACC
    incf    ACCcLO           ;
    btfsc   STATUS,Z         ;
    decf    ACCcHI           ;
    comf    ACCcHI           ;
    btfsc   STATUS,Z         ;
    comf    ACCbLO           ;
    incf    ACCbLO           ;
    btfsc   STATUS,Z         ;
    decf    ACCbHI           ;
    comf    ACCbHI           ;

add_r_start:
    movf    rmndr_l,W        ; get low byte of remainder
    addwf   ACCcLO           ; and add to ACCcLO
    btfsc   STATUS,C         ; check for overflow
    incf    ACCcHI           ; if overflow, increment ACCcHI
    movf    rmndr_h,W        ; get high byte of remainder
    addwf   ACCcHI           ; and add to ACCcHI
    btfsc   STATUS,C         ; check for overflow
    incf    ACCbLO           ; if overflow, increment ACCbLO

    btfss   sign,7           ; check if result negative
    goto    add_r_done      ; if not, go to add_r_done
    comf    ACCcLO           ; if so, negate result
    incf    ACCcLO           ;
    btfsc   STATUS,Z         ;
    decf    ACCcHI           ;
    comf    ACCcHI           ;
    btfsc   STATUS,Z         ;
    comf    ACCbLO           ;
    incf    ACCbLO           ;
    btfsc   STATUS,Z         ;
    decf    ACCbHI           ;
    comf    ACCbHI           ;

add_r_done:
    retlw   0                ; done

;
; Decimal Adjust Subroutine - used after each Q15 multiply to convert Q30 result
; to Q15 number

dec_adjust:
    bcf     sign,7           ; clear sign
    btfss   ACCbHI,7        ; test if number is negative
    goto    adjust          ; go to adjust if not
    bsf     sign,7           ; set sign if negative

    comf    ACCcLO           ; and negate number
    incf    ACCcLO           ;
    btfsc   STATUS,Z         ;

```

AN616

```
    decf    ACCcHI
    comf    ACCcHI
    btfsc   STATUS,Z
    comf    ACCbLO
    incf    ACCbLO
    btfsc   STATUS,Z
    decf    ACCbHI
    comf    ACCbHI

adjust:
    rlf     ACCcHI        ; rotate ACC left 1 bit
    rlf     ACCbLO        ;
    rlf     ACCbHI        ;

    btfss   sign,7        ; check if result should be negative
    goto    adj_done      ; if not, done
    comf    ACCbLO        ; if result negative, negate ACC
    incf    ACCbLO
    btfsc   STATUS,Z
    decf    ACCbHI
    comf    ACCbHI

adj_done:
    retlw   0             ; done

;
; Output Scaling Routine - used to scale output samples by factors of
; 2, 4, or 8 at end of filter routine
;
scale_y_n:
    bcf     sign,7        ; clear sign,7
    btfss   y_n+1,7      ; test if y(n) negative
    goto    start_scale   ; go to start_scale if not
    bsf     sign,7        ; set sign,7 if negative
    comf    y_n           ; and compliment y(n)
    incf    y_n           ;
    btfsc   STATUS,Z     ;
    decf    y_n+1        ;
    comf    y_n+1        ;

start_scale:
    bcf     STATUS,C     ; clear carry
    rlf     y_n+1        ; and rotate y(n) left
    rlf     y_n          ;
    bcf     STATUS,C     ;
    rlf     y_n+1        ;
    rlf     y_n          ;
    bcf     STATUS,C     ;
    rlf     y_n+1        ;
    rlf     y_n          ;

    btfss   sign,7        ; test if result is negative
    goto    scale_y_done  ; go to scale_y_done if not
    comf    y_n           ; negate y(n) if result is negative
    incf    y_n          ;
    btfsc   STATUS,Z     ;
    decf    y_n+1        ;
    comf    y_n+1        ;

scale_y_done:
    retlw   0             ; done

#endasm
```



```

/*****
* Filter function - filter takes current input sample, x(n), and outputs next
* output sample, y(n).
*
* usage:
*   - write sample to be filtered to x_n
*   - call filter()
*   - output is in MSB of y_n (y_n=MSB, y_n+1=LSB)
*
*****/
void filter(){

#asm

    clrf    y_n            ; clear y(n) before starting
    clrf    y_n+1        ;

    clrf    ACCbLO        ; move x(n) to ACCbHI
    movf    x_n,W         ; (scale 8 bit - 16 bit input)
    movwf   ACCbHI        ;

    movlw   B0_H          ; get coefficient b0
    movwf   ACCaHI        ; y(n)=b0*x(n)
    movlw   B0_L          ;
    movwf   ACCaLO        ;
    call    D_mpyF        ;
    movf    ACCcHI,W      ; save remainder from multiply
    movwf   rmndr_h       ;
    movf    ACCcLO,W      ;
    movwf   rmndr_l       ;
    call    dec_adjust    ;
    movf    ACCbHI,W      ;
    movwf   y_n+1         ;
    movf    ACCbLO,W      ;
    movwf   y_n           ;

    movlw   B1_H          ; get coefficient b1
    movwf   ACCaHI        ; y(n)=y(n)+b1*x(n-1)
    movlw   B1_L          ;
    movwf   ACCaLO        ;
    movf    x_n_1+1,W     ;
    movwf   ACCbHI        ;
    movf    x_n_1,W       ;
    movwf   ACCbLO        ;
    call    D_mpyF        ;
    call    add_rmndr     ; add in remainder from previous multiply
    movf    ACCcHI,W      ; and save new remainder
    movwf   rmndr_h       ;
    movf    ACCcLO,W      ;
    movwf   rmndr_l       ;
    call    dec_adjust    ;
    movf    y_n+1,W       ;
    movwf   ACCaHI        ;
    movf    y_n,W         ;
    movwf   ACCaLO        ;
    call    D_add         ;
    movf    ACCbHI,W      ;
    movwf   y_n+1         ;
    movf    ACCbLO,W      ;
    movwf   y_n           ;

    movlw   B2_H          ; get coefficient b2
    movwf   ACCaHI        ; y(n)=y(n)+b2*x(n-2)
    movlw   B2_L          ;
    movwf   ACCaLO        ;
    movf    x_n_2+1,W     ;
    movwf   ACCbHI        ;

```

AN616

```
movf    x_n_2,W           ;
movwf   ACCbLO            ;
call    D_mpyF            ;
call    add_rmndr         ; add in remainder from previous multiply
movf    ACCcHI,W         ; and save new remainder
movwf   rmndr_h           ;
movf    ACCcLO,W         ;
movwf   rmndr_l           ;
call    dec_adjust       ;
movf    y_n+1,W          ;
movwf   ACCaHI           ;
movf    y_n,W            ;
movwf   ACCaLO           ;
call    D_add            ;
movf    ACCbHI,W         ;
movwf   y_n+1            ;
movf    ACCbLO,W         ;
movwf   y_n              ;

movlw   A1_H              ; get coefficient a1
movwf   ACCaHI           ; y(n)=y(n)+a1*y(n-1)
movlw   A1_L              ;
movwf   ACCaLO           ;
movf    y_n_1+1,W        ;
movwf   ACCbHI           ;
movf    y_n_1,W          ;
movwf   ACCbLO           ;
call    D_mpyF            ;
call    add_rmndr         ; add in remainder from previous multiply
movf    ACCcHI,W         ; and save new remainder
movwf   rmndr_h           ;
movf    ACCcLO,W         ;
movwf   rmndr_l           ;
call    dec_adjust       ;
movf    y_n+1,W          ;
movwf   ACCaHI           ;
movf    y_n,W            ;
movwf   ACCaLO           ;
call    D_sub            ;
movf    ACCbHI,W         ;
movwf   y_n+1            ;
movf    ACCbLO,W         ;
movwf   y_n              ;

movlw   A2_H              ; get coefficient a2
movwf   ACCaHI           ; y(n)=y(n)+a2*y(n-2)
movlw   A2_L              ;
movwf   ACCaLO           ;
movf    y_n_2+1,W        ;
movwf   ACCbHI           ;
movf    y_n_2,W          ;
movwf   ACCbLO           ;
call    D_mpyF            ;
call    add_rmndr         ;
call    dec_adjust       ;
movf    y_n+1,W          ;
movwf   ACCaHI           ;
movf    y_n,W            ;
movwf   ACCaLO           ;
call    D_sub            ;
movf    ACCbHI,W         ;
movwf   y_n+1            ;
movf    ACCbLO,W         ;
movwf   y_n              ;

movf    x_n_1,W          ; x(n-2)=x(n-1)
```

```
movwf x_n_2           ;
movf x_n_1+1,W       ;
movwf x_n_2+1        ;
movf x_n,W           ; x(n-1)=x(n)
movwf x_n_1+1        ;
clrf x_n_1           ;

movf y_n_1,W         ; y(n-2)=y(n-1)
movwf y_n_2          ;
movf y_n_1+1,W       ;
movwf y_n_2+1        ;
movf y_n,W           ; y(n-1)=y(n)
movwf y_n_1          ;
movf y_n+1,W         ;
movwf y_n_1+1        ;

call scale_y_n       ;

movf y_n+1,W         ; shift lsb of y_n to msb
movwf y_n            ;

#endasm
}
```

AN616

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX E: NOTCH FILTER

```
/*
 * 60 Hertz Notch Filter
 *
 * Written for "Digital Signal Processing with the PIC16C74" Application Note.
 *
 * This example program use the filter() function to implement a 60Hz notch
 * filter. T0 is used to generate a 1kHz sample clock. The program samples the
 * input signal x(n) on A-D channel 1, calls the filter routine signal, and
 * outputs y(n) to PWM channel 1.
 *
 * If FILTER set to 0, performs straight talkthru from A-D to PWM output.
 * T0 period can be changed to cary the sample rate.
 *
 * D. Mostowfi 4/95
 */
#include "\mpc\include\16c74.h" /* c74 header file */

#include "\mpc\apnotes\analogio.c" /* analog I/O module */
#include "\mpc\apnotes\iir_filt.c" /* iir filter module */

#define FILTER 1

/* Function Prototypes */
void main_isr();
void timer0_isr();

/*
 * main_isr - 16C74 vectors to 0004h (MPC __INT() function) on any interrupt *
 * assembly language routine saves W and Status registers then tests flags in
 * INTCON to determine source of interrupt. Routine then calls appropriate isr.
 * Restores W and status registers when done.
 */
void __INT(void)
{
    if(INTCON.TOIF){
        INTCON.TOIF=0;
        timer0_isr();
    }

    /* Restore W, WImage, and STATUS registers */
}

#asm
    BCF    STATUS,RP0        ;Bank 0
    MOVF  temp_PCLATH, W
    MOVWF PCLATH            ;PCLATH restored
    MOVF  temp_WImage, W
    MOVWF __WImage         ;WImage restored
    MOVF  temp_FSR, W
    MOVWF FSR              ;FSR restored
    SWAPF temp_STATUS,W
    MOVWF STATUS           ;RP0 restored
    SWAPF temp_WREG,F
    SWAPF temp_WREG,W      ;W restored
#endasm

}

/*
 * timer 0 interrupt service routine
 */
void timer0_isr(void)
{

```

```

    TMR0=100;                /* reload value for lms period */
    PORTB.0=!PORTB.0;        /* toggle PORTB.0 */
    sample_flag=active;      /* set sample flag */
}

void main()
{
    /* initialize OPTION register */
    OPTION=0b00000011;      /* assign prescaler to T0 */

    /* initialize INTCON register (keep GIE inactive!) */
    INTCON=0b00000000;      /* disable all interrupts */

    /* initialize PIE1 and PIE2 registers (periphreal interrupts) */
    PIE1=0b00000000;        /* disable all peripheral interrupts */
    PIE2=0b00000000;

    /* initialize T1CON and T2CON registers */
    T1CON=0b00000000;        /* T1 not used */
    T2CON=0b00000000;        /* T2 not used */

    /* initialize CCPxCON registers */
    CCP1CON=0b00001100;      /* set CCP1CON for PWM mode */
    CCP2CON=0b00000000;      /* CCP2CON=0 (PWM 2 not used) */

    /* initialize SSPCON register */
    SSPCON=0b00000000;      /* serial port - not used */

    /* initialize ADCONx registers */
    ADCON0=0b00000000;      /* a-d converter */
    ADCON1=0b00000010;

    /* initialize TRISx register (port pins as inputs or outputs) */
    TRISA=0b00001111;
    TRISB=0b00000000;
    TRISC=0b11111011;
    TRISD=0b11111111;
    TRISE=0b11111111;

    /* clear watchdog timer (not used) */
    CLRWDT();

    /* initialize program bit variables */
    FLAGS=0b00000000;

    /* intialize output port pins */
    PORTB=0;

    /* enable interrupts... */

    INTCON.T0IE=1;          /* peripheral interrupt enable */
    INTCON.GIE=1;          /* global interrupt enable */

    init_PWM(0x40);         /* init PWM port */
    init_filter();         /* init filter */
    while(1){
        while(!sample_flag){ /* wait for sample clock flag to be set */
            sample_flag=0;    /* clear sample clock flag */
            x_n=get_sample(1); /* read ADC channel 1 into x(n) */
            if(FILTER==1){    /* if filter enabled */
                filter();     /* call filter routine */
            }
            else{             /* or else write x(n) to y(n) (talkthru) */
                y_n=x_n;
            }
            write_PWM((char)y_n,0); /* write y_n to PWM port 1 */
        }
    }
}

```

AN616

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX F: 8-BIT MULTIPLY AND SCALING ROUTINES

```
/*
 * 8 bit Multiply and Scaling Routines
 *
 * Written for "Digital Signal Processing with the PIC16C74" Application Note.
 *
 * This module provides a 8 bit signed multiply and scaling routine for the
 * PICTONE.C tone generation program. The routines are adapted from "Math
 * Routines for the 16C5x" in Microchip's Embedded Controller Handbook.
 *
 * All numbers are assumed to be signed 2's compliment format.
 *
 * D. Mostowfi 11/94
 */
char multcnd;          /* 8 bit multiplicand */
char multplr;         /* 8 bit multiplier */
char result_h;        /* result - high byte */
char result_l;        /* result - low byte */
char sign;            /* result sign */

#asm

;
; 8x8 signed multiply routine
; called from PICTONE.C module (assembly language routine)
;
.MACRO mult_core bit
    btfss    multplr,bit
    goto     \no_add
    movf     multcnd,W
    addwf    result_h,F
\endasm

\no_add:
    rrf     result_h
    rrf     result_l
.ENDM

_8x8smul:
    movf    multcnd,W          ; get multiplicand
    xorwf   multplr,W         ; and xor with multiplier
    movwf   sign              ; and save sign of result
    btfss   multcnd,7         ; check sign bit of multiplicand
    goto    chk_multplr       ; go and check multiplier if positive
    comf    multcnd           ; negate if negative
    incf    multcnd           ;

chk_multplr:
    btfss   multplr,7         ; check sign bit of multiplier
    goto    multiply          ; go to multiply if positive
    comf    multplr           ; negate if negative
    incf    multplr           ;

multiply:
    movf    multcnd,W         ; set up multiply registers
    bcf     STATUS,C          ;
    clrf    result_h          ;
    clrf    result_l          ;
    mult_core 0                ; and do multiply core 8 times
    mult_core 1                ;
    mult_core 2                ;
```

```
mult_core 3      ;
mult_core 4      ;
mult_core 5      ;
mult_core 6      ;
mult_core 7      ;

set_sign:
    btfss sign,7      ; test sign to see if result negative
    retlw 0           ; done if not! (clear W)
    comf result_l     ; negate result if sign set
    incf result_l     ;
    btfsc STATUS,Z   ;
    decf result_h     ;
    comf result_h     ;

    retlw 0           ; done (clear W)

;
; Scaling Routine (used after a multiply to scale 16 bit result)
; Operates on result_h and result_l - final result is in result_l
; routine divides by 32 to restore Q7 result of 2*b*y(n-1) in tone
; generation algorithm
;
scale_16:
    btfss sign,7      ; test if negative (sign set from mult)
    goto start_shift ; go to start shift if pos.
    comf result_l     ; negate first if neg.
    incf result_l     ;
    btfsc STATUS,Z   ;
    decf result_h     ;
    comf result_h     ;

start_shift:
    bcf STATUS,C      ; clear status
    rrf result_h      ; and shift result left 5x (/32)
    rrf result_l      ;
    rrf result_h      ;
    rrf result_l      ;
    rrf result_h      ;
    rrf result_l      ;
    rrf result_h      ;
    rrf result_l      ;
    rrf result_h      ;
    rrf result_l      ;

    btfss sign,7      ; test if result negative
    goto scale_done   ; done if not negative
    comf result_l     ; negate result if negative
    incf result_l     ;
    btfsc STATUS,Z   ;
    decf result_h     ;
    comf result_h     ;

scale_done:
    ;

    retlw 0           ; done (clear W)

#endasm
```

AN616

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX G: DOUBLE PRECISION MATH ROUTINES

```

/*****
* Double Precision Math Routines
*
* This module contains assembly language routines from "Math Routines for the
* 16C5x" from Microchip's Embedded Controller Handbook that have been adapted
* for use with the Bytecraft MPC C Compiler.
*
* Routines are used IIR_FILT.C module written for "Digital Signal Processing
* with the PIC16C74" Application Note.
*
* D. Mostowfi 3/95
*****/

/*
Start of converted MPASM modules:

;*****
;
;           Double Precision Addition & Subtraction
;
;*****;
; Addition : ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
;   (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
;   (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
;   (c) CALL D_add
;   (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
;
; Performance :
;   Program Memory :      07
;   Clock Cycles   :      08
;*****;
; Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
;   (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
;   (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
;   (c) CALL D_sub
;   (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
;
; Performance :
;   Program Memory :      14
;   Clock Cycles   :      17
;*****;
*/

char ACCaLO; //equ    10    changed equ statements to C char variables
char ACCaHI; //equ    11
char ACCbLO; //equ    12
char ACCbHI; //equ    13
;

#asm          /* start of in-line assembly code */

;   include "mpreg.h"      commented out these
;   org      0             two lines (MPASM specific)

;*****
;           Double Precision Subtraction ( ACCb - ACCa -> ACCb )
;
D_sub call    neg_A2          ; At first negate ACCa; Then add
;
;*****
;           Double Precision Addition ( ACCb + ACCa -> ACCb )

```



```

;
D_add  movf   ACCaLO,W
      addwf  ACCbLO           ;add lsb
      btfsc  STATUS,C        ;add in carry
      incf   ACCbHI
      movf   ACCaHI,C
      addwf  ACCbHI           ;add msb
      retlw  0

neg_A2  comf   ACCaLO           ; negate ACCa ( -ACCa -> ACCa )
      incf   ACCaLO
      btfsc  STATUS,Z
      decf   ACCaHI
      comf   ACCaHI
      retlw  0

;*****
;                               Double Precision Multiplication
;
;                               ( Optimized for Speed : straight Line Code )
;
;*****;
; Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc ( 32 bits )
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
; (c) CALL D_mpy
; (d) The 32 bit result is in location ( ACCbHI,ACCbLO,ACCcHI,ACCcLO )
;
; Performance :
; Program Memory :      240
; Clock Cycles   :      233
;
; Note : The above timing is the worst case timing, when the
;        register ACCb = FFFF. The speed may be improved if
;        the register ACCb contains a number ( out of the two
;        numbers ) with less number of 1s.
;
; The performance specs are for Unsigned arithmetic ( i.e,
; with "SIGNED equ FALSE ").
;*****;
;

#endasm
//char ACCaLO; equ    10   Commented out - already defined in Dbl_add
//char ACCaHI; equ    11
//char ACCbLO; equ    12
//char ACCbHI; equ    13
char ACCcLO; //equ    14   changed equ statements to C char variables
char ACCcHI; //equ    15
char ACCdLO; //equ    16
char ACCdHI; //equ    17
char temp;   //equ    18
char sign;   //equ    19

#asm
;
; include "mpreg.h"      commented out these
; org 0                 two lines (MPASM specific)
;*****;
SIGNED equ 1           ; Set This To 'TRUE' if the routines
;                       ; for Multiplication & Division needs
;                       ; to be assembled as Signed Integer
;                       ; Routines. If 'FALSE' the above two
;                       ; routines ( D_mpy & D_div ) use

```

AN616

```
;                                     ; unsigned arithmetic.
;*****
;      multiplication macro
;
.MACRO mulMac                          ; changed macro to conform to MPC macro
;      LOCAL    NO_ADD                  ; language - declaration is different
;                                     ; and macro labels are preceded by "/"

    rrf    ACCdHI                        ; rotate d right
    rrf    ACCdLO
    btfss  STATUS,C                      ; need to add?
    goto   \NO_ADD                       ; no addition necessary
    movf   ACCaLO,W                      ; Addition ( ACCb + ACCa -> ACCb )
    addwf  ACCbLO                          ; add lsb
    btfsc  STATUS,C                      ; add in carry
    incf   ACCbHI
    movf   ACCaHI,W
    addwf  ACCbHI                          ;add msb
\NO_ADD rrf    ACCbHI
    rrf    ACCbLO
    rrf    ACCcHI
    rrf    ACCcLO
;
    .ENDM                                ; end of modified macro
;
;*****
;      Double Precision Multiply ( 16x16 -> 32 )
;      ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
;      in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
;
D_mpyF                                ;results in ACCb(16 msb's) and ACCc(16 lsb's)
;
    .IF   SIGNED
        CALL   S_SIGN
    .ENDIF
;
    call    setup
;
; use the mulMac macro 16 times
;
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
mulMac
;
    .IF   SIGNED
    btfss  sign,7
    retlw  0
    comf   ACCcLO                          ; negate ACCa ( -ACCa -> ACCa )
    incf   ACCcLO
    btfsc  STATUS,Z
    decf   ACCcHI
    comf   ACCcHI
    btfsc  STATUS,Z
```

```

neg_B   comf   ACCbLO           ; negate ACCb
        incf   ACCbLO
        btfsc STATUS,Z
        decf   ACCbHI
        comf   ACCbHI
        retlw  0
        .ELSE
        retlw  0
        .ENDIF
;
;*****
;
setup   movlw  16               ; for 16 shifts
        movwf temp
        movf  ACCbHI,W         ;move ACCb to ACCd
        movwf ACCdHI
        movf  ACCbLO,W
        movwf ACCdLO
        clrf  ACCbHI
        clrf  ACCbLO
        retlw  0
;
;*****
;
neg_A   comf   ACCaLO           ; negate ACCa ( -ACCa -> ACCa )
        incf   ACCaLO
        btfsc STATUS,Z
        decf   ACCaHI
        comf   ACCaHI
        retlw  0
;
;*****
; Assemble this section only if Signed Arithmetic Needed
;
        .IF    SIGNED
;
S_SIGN  movf   ACCaHI,W
        xorwf ACCbHI,W
        movwf sign
        btfss ACCbHI,7         ; if MSB set go & negate ACCb
        goto  chek_A
;
        comf   ACCbLO           ; negate ACCb
        incf   ACCbLO
        btfsc STATUS,Z
        decf   ACCbHI
        comf   ACCbHI
;
chek_A  btfss  ACCaHI,7         ; if MSB set go & negate ACCa
        retlw  0
        goto  neg_A
;
        .ENDIF

#endasm

```

AN616

NOTES:

Fixed Point Routines

*Author: Frank J. Testa
Design Consultant*

INTRODUCTION

This application note presents an implementation of the following fixed point math routines for the PIC16/17 microcontroller families:

- Multiplication
- Division

Routines for the PIC16/17 families are provided in a variety of fixed point formats, including both unsigned and signed two's complement arithmetic.

FIXED POINT ARITHMETIC

Unsigned fixed point binary numbers A , can be represented in the form

$$A = \sum_{k=0}^{n-1} a(k) \cdot 2^{k-r} = 2^{-r} \sum_{k=0}^{n-1} a(k) \cdot 2^k$$

where n is the number of bits, $a(k)$ is the k th bit with $a(0) = \text{LSb}$, and r indicates the location of the radix point. For example, in the case where A is an integer, $r = 0$ and when A is a fraction less than one, $r = n$. The value of r only affects the interpretation of the numbers in a fixed point calculation, with the actual binary representation of the numbers independent of the value of r . Factoring out of the above sum, it simply locates the radix point of the representation and is analogous to an exponent in a floating point system. Using the notation $Q_{i,j}$ to denote a fixed point binary number with i bits to the left of the radix point and j to the right, the above n -bit format is in $Q_{n-r,r}$. With care, fixed point calculations can be performed on operands in different Q formats. Although the radix point must be aligned for addition or subtraction, multiplication provides an illustrative example of the simple interpretive nature of r . Consider the unsigned product of a $Q_{20,4}$ number with a $Q_{8,8}$. After calling the appropriate unsigned 24×16 bit multiply for these fixed point arguments, the 40-bit fixed point result is in $Q_{28,12}$, where the arguments of the Q notation are summed respectively. Similar arguments can be made for two's complement arithmetic, where the negative representation of a positive number is obtained by reversing the value of each bit and incrementing the result by one. Producing a unique representation of zero, and covering the range -2^{n-1} to $2^{n-1} - 1$, this is more easily applied in addition and subtraction operations and is therefore the most commonly used method of representing positive and negative numbers in fixed point arithmetic.

The above analysis in Q notation can be employed to build dedicated fixed point algorithms, leading to improved performance over floating point methods in cases where the size of the arguments required for the range and precision of the calculations is not large enough to destroy gains made by fixed point methods.

AN617

FIXED POINT FORMATS

The fixed point library routines supports 8-, 16-, 24- and 32-bit formats in the following combinations:

Division Library Names	Format	Multiplication Library Names	Format
PIC16C5X/PIC16CXX Routines			
FXD0808S, FXD0808U, FXD0807U, FXD0707U	8/8	FXM0808S, FXM0808U, FXM0807U	8•8
FXD1608S, FXD1608U, FXD1607U, FXD1507U	16/8	FXM1608S, FXM1608U, FXM1607U, FXM1507U	16•8
FXD1616S, FXD1616U, FXD1515U	16/16	FXM1616S, FXM1616U, FXM1515U	16•16
FXD2416S, FXD2416U, FXD2315U	24/16	FXM2416S, FXM2416U, FXM2315U	24•16
FXD2424S, FXD2424U, FXD2323U	24/24	FXM2424S, FXM2424U, FXM2323U	24•24
FXD3216S, FXD3216U, FXD3115U	32/16	FXM3216S, FXM3216U, FXM3115U	32•16
FXD3224S, FXD3224U, FXD3123U	32/24	FXM3224S, FXM3224U, FXM3123U	32•24
FXD3232S, FXD3232U, FXD3131U	32/32	FXM3232S, FXM3232U, FXM3131U	32•32
PIC17CXX Functions			
FXD0808S, FXD0808U, FXD0807U, FXD0707U	8/8	FXM0808S, FXM0808U, FXM0807U	8•8
FXD1608S, FXD1608U, FXD1607U, FXD1507U	16/8	FXM1608S, FXM1608U, FXM1507U	16•8
FXD1616S, FXD1616U, FXD1615U, FXD1515U	16/16	FXM1616S, FXM1616U, FXM1515U	16•16
FXD2416S, FXD2416U, FXD2415U, FXD2315U	24/16	FXM2416S, FXM2416U, FXM2315U	24•16
FXD2424S, FXD2424U, FXD2423U, FXD2323U	24/24	FXM2424S, FXM2424U, FXM2323U	24•24
FXD3216S, FXD3216U, FXD3215U, FXD3115U	32/16	FXM3216S, FXM3216U, FXM3115U	32•16
FXD3224S, FXD3224U, FXD3223U, FXD3123U	32/24	FXM3224S, FXM3224U, FXM3123U	32•24
FXD3232S, FXD3232U, FXD3231U, FXD3131U	32/32	FXM3232S, FXM3232U, FXM3131U	32•32

Note: U - unsigned math operation, S - signed math operation

These general format combinations are implemented in both signed and unsigned versions. Additional unsigned routines are implemented with arguments reduced by one bit to accommodate the case of operations on signed numbers, with arguments known to be nonnegative, thereby, resulting in some performance improvement.

DATA RAM REQUIREMENTS

The following contiguous data ram locations are used by the library:

ACCB7	=	REMB3	=	AEXP	=	EXP	AARG and ACC exponent
ACCB6	=	REMB2	=	BEXP			BARG exponent
ACCB5	=	REMB1					
ACCB4	=	REMB0					remainder
ACCB3	=	AARGB3					
ACCB2	=	AARGB2					
ACCB1	=	AARGB1					
ACCB0	=	AARGB0	=	ACC			AARG and ACC
SIGN							sign in MSB
FPFLAGS							exception flags and option bits
BARGB3							
BARGB2							
BARGB1							
BARGB0							BARG
TEMPB3							
TEMPB2							
TEMPB1							
TEMPB0							temporary storage

These definitions are identical with those used by the IEEE 754 compliant floating point library[6], AN575.

USAGE

Multiplication assumes the multiplicand in AARG, multiplier in BARG, and produces the result in ACC. Division assumes a dividend in AARG, divisor in BARG, and quotient in ACC with remainder in REM.

ADDITION/SUBTRACTION

Because of the generally trivial nature of addition and subtraction, the call and return overhead outweighs the need for explicit routines and so they are not included in the library. However, the PIC16C5X/PIC16CXX families do not have an add with carry or subtract with borrow instruction, leading to subtleties regarding production of a correct carry-out in a multiple byte add or subtract. In the case of a two byte add or subtract, the most elegant solution to these difficulties, requiring 6 cycles, appears to be given by the following code in Example 1.

EXAMPLE 1: TWO BYTE ADDITION/SUBTRACTION ROUTINES

```

ADD      MOVF      AARGB1,W
         ADDWF    BARGB1
         MOVF    AARGB0,W
         BTFSC   _C
         INCF   AARGB0,W
         ADDWF  BARGB0
SUB      MOVF      AARGB1,W
         SUBWF   BARGB1
         MOVF    AARGB0,W
         BTFSS  _C
         INCF   AARGB0,W
         SUBWF  BARGB0

```

The four instructions after the initial add/subtract, can be easily concatenated for operations involving more than two bytes. Because addition and subtraction are required in standard algorithms for multiplication and division, these issues permeate the implementation of both fixed and floating point algorithms for the PIC16C5X/PIC16CXX families.

MULTIPLICATION

The fixed point multiply routine FXPMxxyy, takes an xx-bit multiplicand in AARG, a yy-bit multiplier in BARG and returns the (xx+yy)-bit product in ACC. The implementation uses a standard sequential add-shift algorithm, negating both factors if BARG < 0, to produce the positive multiplier required by the method. Analogous to simple longhand binary multiplication, the multiplier bits are sequentially tested, with one indicating an add-shift and zero simply a shift. The shift

AN617

is required to align the partial product for the next possible add[1]. Several examples are shown in Example 2.

EXAMPLE 2: MULTIPLICATION EXAMPLES

FXM2416S(0xC11682,0x608B)
 = FXM2416S(-4123006,24715)
 = 0xE84647F896
 = -101900093290

FXM1616U(0x0458,0x822C)
 = FXM1616U(1112,33324)
 = 0x02356F20
 = 37056288

TABLE 1: PIC17CXX FIXED POINT MULTIPLY PERFORMANCE DATA: APPENDIX E

Routine	Max Cyc	Min Cyc	PM	DM	Page
FXM0808S	50/53	26	65	5	2-375
FXM0808U	39	23	53	3	2-375
FXM0707U	37	21	49	3	2-375
FXM1608S	74/79	35	100	6	2-376
FXM1608U	75	24	90	6	2-376
FXM1507U	69	24	82	6	2-376
FXM1616S	168/175	24	197	8	2-377
FXM1616U	156	41	191	8	2-377
FXM1515U	150	39	185	8	2-377
FXM2416S	213/223	43	253	10	2-378
FXM2416U	203	43	239	10	2-378
FXM2315U	194	41	229	10	2-378
FXM2424S	334/346	60	392	12	2-379
FXM2424U	316	60	364	12	2-379
FXM2323U	308	58	354	12	2-380
FXM3216S	258/270	46	301	13	2-380
FXM3216U	265	44	298	13	2-381
FXM3115U	254	42	285	13	2-381
FXM3224S	403/417	62	464	15	2-381
FXM3224U	410	61	459	15	2-382
FXM3123U	399	59	446	15	2-382
FXM3232S	556/572	79	635	16	2-383
FXM3232U	563	78	628	16	2-383
FXM3131U	543	76	606	16	2-384

Legend: PM - Program memory, DM - Data Memory

TABLE 2: PIC16C5X/PIC16CXX FIXED POINT MULTIPLY PERFORMANCE DATA: APPENDIX C

Routine	Max Cyc	Min Cyc	PM	DM	Page
FXM0808S	79/82	55	33	5	2-252
FXM0808U	73	54	21	4	2-252
FXM0707U	67	48	23	4	2-252
FXM1608S	122/128	55	44	7	2-246
FXM1608U	126	59	31	7	2-247
FXM1507U	114	52	35	7	2-247
FXM1616S	260/269	105	74	9	2-240
FXM1616U	256	107	58	9	2-241
FXM1515U	244	103	63	9	2-241
FXM2416S	334/346	108	92	12	2-231
FXM2416U	334	110	70	12	2-232
FXM2315U	319	104	76	12	2-232
FXM2424S	520/535	157	126	13	2-221
FXM2424U	512	159	98	13	2-222
FXM2323U	497	154	107	13	2-222
FXM3216S	408/423	111	98	9	2-208
FXM3216U	412	114	84	9	2-208
FXM3115U	392	106	91	9	2-209
FXM3224S	634/652	160	152	15	2-196
FXM3224U	630	162	151	15	2-197
FXM3123U	610	157	129	15	2-197
FXM3232S	868/889	207	189	17	2-181
FXM3232U	856	209	168	17	2-182
FXM3131U	836	204	168	17	2-182

Legend: PM - Program memory, DM - Data Memory

DIVISION

The fixed point divide routine FXPDxxyy, takes an xx-bit dividend in AARG, a yy-bit divisor in BARG and returns the xx-bit quotient in ACC and yy-bit remainder in REM. Unlike multiplication, division is not deterministic, requiring a trial-and-error sequential shift and subtract process. Binary division is less complicated than decimal division because the possible quotient digits are only zero or one. If the divisor is less than the partial remainder, the corresponding quotient bit is set to one followed by a shift and subtract. Otherwise, the divisor is greater than the partial remainder, the quotient bit is set to zero and only a shift is performed. The intermediate partial remainder may be restored at each stage as in restoring division, or corrected at the end as in nonrestoring division. Implementation dependent trade-offs between worst case versus average performance affect the choice between these two approaches, and therefore, macros for each method are provided.

<p>Note: A test for divide by zero exception is not performed and must be explicitly provided by the user.</p>

The results of the division process for AARG/BARG, satisfy the relation

$$AARG = BARG \bullet QUOTIENT + REMAINDER,$$

where the remainder has the same sign as the quotient, and represents the fraction of the result in units of the denominator BARG. Some simple examples are given in Example 3.

EXAMPLE 3: DIVISION EXAMPLES

$$FXD1608S(0xC116,0x60) = 0xFF59, 0xB6$$

$$FXD1616U(0x9543,0x4AA1) = 0x0002, 0x0001$$

TABLE 3: PIC17CXX FIXED POINT DIVIDE PERFORMANCE DATA: APPENDIX F

Routine	Max Cyc	Min Cyc	PM	DM	Page
FXD0808S	71/77	71/77	77	4	2-449
FXD0808U	75	67	74	3	2-449
FXD0807U	66	66	65	3	2-450
FXD0707U	61	61	60	3	2-450
FXD1608S	135/146	135/146	146	5	2-448
FXD1608U	196	156	195	4	2-448
FXD1607U	130	130	129	4	2-448
FXD1507U	125	125	124	4	2-449
FXD1616S	201/214	187/200	241	7	2-446
FXD1616U	244	180	243	6	2-447
FXD1615U	197	182	216	6	2-447
FXD1515U	191	177	218	6	2-447
FXD2416S	297/314	272/289	353	8	2-444
FXD2416U	365	339	453	8	2-445
FXD2415U	294	268	339	8	2-445
FXD2315U	287	262	330	8	2-446
FXD2424S	371/390	344/363	482	10	2-475
FXD2424U	440	412	577	10	2-476
FXD2423U	369	341	460	9	2-476
FXD2323U	361	334	448	9	2-476
FXD3216S	393/414	363/384	476	9	2-473
FXD3216U	485	459	608	9	2-474
FXD3215U	390	359	451	8	2-474
FXD3115U	383	353	442	8	2-475
FXD3224S	491/514	456/479	639	11	2-419
FXD3224U	584	548	769	11	2-420
FXD3223U	489	453	612	10	2-421
FXD3123U	481	446	600	10	2-421
FXD3232S	589/614	552/577	800	13	2-418
FXD3232U	683	645	930	13	2-419
FXD3231U	588	550	773	12	2-419
FXD3131U	579	542	758	12	2-419

Legend: PM - Program memory, DM - Data Memory

TABLE 4: PIC16C5X/PIC16CXX FIXED POINT DIVIDE PERFORMANCE DATA: APPENDIX D

Routine	Max Cyc	Min Cyc	PM	DM	Page
FXD0808S	90/96	90/96	41	5	2-344
FXD0808U	100	92	15	4	2-345
FXD0807U	88	88	21	4	2-345
FXD0707U	80	80	44	4	2-345
FXD1608S	176/188	176/188	67	6	2-338
FXD1608U	294	230	41	7	2-339
FXD1607U	174	174	41	5	2-339
FXD1507U	166	166	44	5	2-340
FXD1616S	304/319	269/284	74	8	2-330
FXD1616U	373	277	27	7	2-330
FXD1515U	294	274	45	7	2-331
FXD2416S	417/438	389/410	140	8	2-321
FXD2416U	529	501	172	8	2-322
FXD2315U	407	379	120	7	2-322
FXD2424S	541/565	509/533	253	12	2-311
FXD2424U	676	644	226	13	2-312
FXD2323U	531	499	211	12	2-313
FXD3216S	551/578	515/551	201	10	2-299
FXD3216U	703	667	243	9	2-300
FXD3115U	541	505	160	9	2-300
FXD3224S	715/742	675/702	280	11	2-287
FXD3224U	867	827	299	11	2-288
FXD3123U	705	665	232	10	2-288
FXD3232S	879/912	835/868	357	13	2-271
FXD3232U	1031	987	364	13	2-272
FXD3131U	869	825	304	13	2-272

Legend: PM - Program memory, DM - Data Memory

REFERENCES

1. Cavanagh, J.J.F., "Digital Computer Arithmetic," McGraw-Hill, 1984.
2. Hwang, K., "Computer Arithmetic," John Wiley & Sons, 1979.
3. Scott, N.R., "Computer Number Systems & Arithmetic," Prentice Hall, 1985.
4. IEEE Standards Board, "IEEE Standard for Floating-Point Arithmetic," ANSI/IEEE Std 754-1985, IEEE, 1985.
5. Knuth, D.E., "The Art of Computer Programming, Volume 2," Addison-Wesley, 1981.
6. F.J. Testa, "IEEE 754 Compliant Floating Point Routines," AN575, Embedded Control Handbook, Microchip Technology Inc., 1995.

APPENDIX A: ALGORITHMS

Several algorithms for decimal to binary conversion are given below. The integer and fractional conversion algorithms are useful in both native assembly as well as high level languages.

A.1 Integer conversion algorithm[3]:

Given an integer I, where d(k) are the bit values of its n bit binary representation with d(0) = LSB,

$$I = \sum_{k=0}^{n-1} d(k) \cdot 2^k$$

k=0

I(k) = I

while I(k) != 0

d(k) = remainder of I(k)/2

I(k+1) = $\lceil I(k)/2 \rceil$

k = k + 1

endw

where $\lceil \rceil$ denotes the greatest integer function (or ceiling function).

A.2 Fractional conversion algorithm[3]:

Given a fraction F, where d(k) are the bit values of its n bit binary representation with d(1)=MSB,

$$F = \sum_{k=1}^n d(k) \cdot 2^{-k}$$

k=0

F(k) = F

while k <= n

d(k) = $\lceil F(k) \cdot 2 \rceil$

F(k+1) = fractional part of F(k)•2

k = k + 1

endw

APPENDIX B: FLOWCHARTS

FIGURE B-1: MULTIPLICATION FLOWCHART

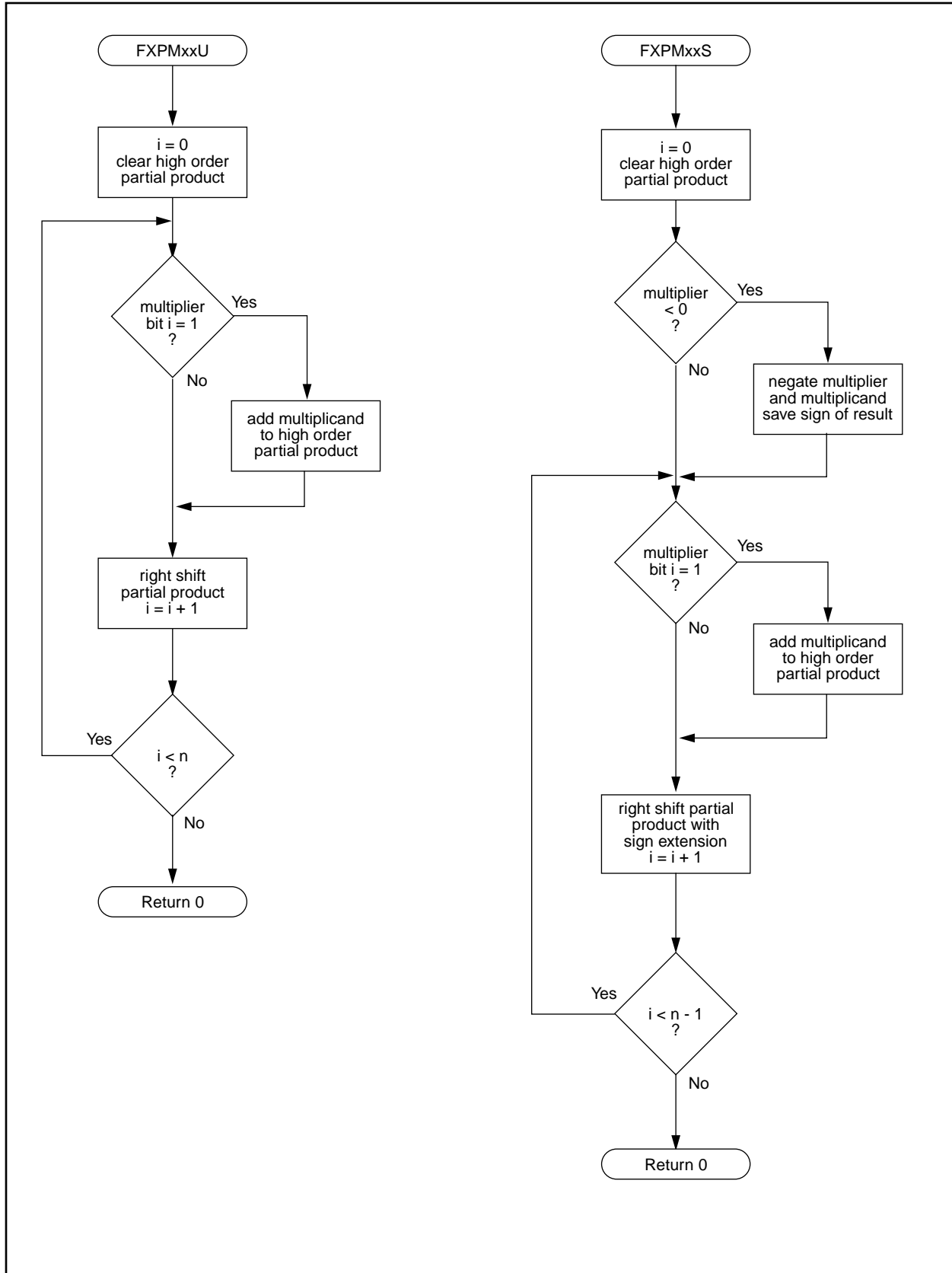
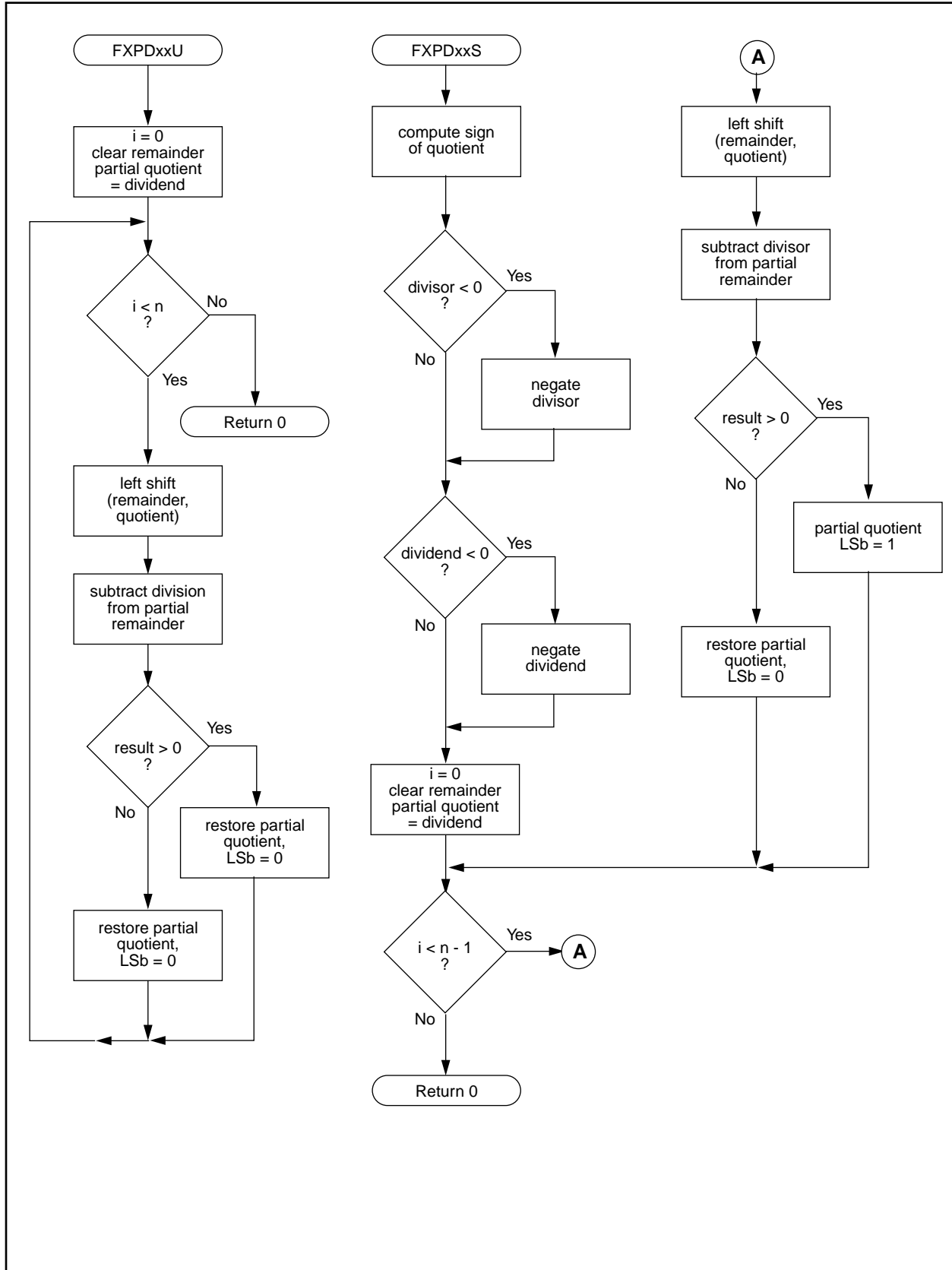


FIGURE B-2: DIVISION FLOWCHART



Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX C: MULTIPLY ROUTINES FOR THE PIC16C5X/PIC16CXX

Table of Contents for Appendix C

C.1	32x32 PIC16C5X/PIC16CXX Fixed Point Multiply Routines	2-164
C.2	32x24 PIC16C5X/PIC16CXX Fixed Point Multiply Routines	2-182
C.3	32x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines	2-197
C.4	24x24 PIC16C5X/PIC16CXX Fixed Point Multiply Routines	2-209
C.5	24x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines	2-222
C.6	16x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines	2-232
C.7	16x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines	2-241
C.8	8x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines	2-247

C.1 32x32 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

```

; 32x32 PIC16 FIXED POINT MULTIPLY ROUTINES      VERSION 1.2
; Input:  fixed point arguments in AARG and BARG
; Output: product AARGxBARG in AARG
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed multiply application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in performance.
;
; Routine      Clocks      Function
; FXM3232S     889         32x32 -> 64 bit signed fixed point multiply
; FXM3232U     856         32x32 -> 64 bit unsigned fixed point multiply
; FXM3131U     836         31x31 -> 62 bit unsigned fixed point multiply
;
; The above timings are based on the looped macros. If space permits,
; approximately 128-168 clocks can be saved by using the unrolled macros.
;
; list      r=dec,x=on,t=off
; include <PIC16.INC>      ; general PIC16 definitions
; include <MATH16.INC>    ; PIC16 math library definitions
;*****
;*****
;
; Test suite storage
RANDHI      equ      0x1A      ; random number generator registers
RANDLO      equ      0x1B
DATA        equ      0x20      ; beginning of test data
;*****
;*****
;
; Test suite for 32x32 bit fixed point multiply algorithms
;
; org      0x0005
MAIN        MOVLW      RAMSTART
;
; MOVWF    FSR
MEMLOOP     CLRF       INDF
;
; INCF     FSR
;
; MOVLW   RAMSTOP
;
; SUBWF   FSR,W
;
; BTFSS   _Z
;
; GOTO    MEMLOOP
;
; MOVLW   0x45      ; seed for random numbers
;
; MOVWF   RANDLO
;
; MOVLW   0x30
;
; MOVWF   RANDHI
;
; MOVLW   DATA
;
; MOVWF   FSR
;
; BCF     _RP0
;
; BCF     _RP1
;
; BCF     _IRP
;
; CALL    TFXM3232
;
; SELF    GOTO        SELF
RANDOM16   RLF        RANDHI,W      ; random number generator
;
; XORWF   RANDHI,W
;
; MOVWF   TEMPB0
;
; SWAPF   RANDHI
;
; SWAPF   RANDLO,W

```

```

MOVWF    TEMPB1
RLF      TEMPB1,W
RLF      TEMPB1
MOVF     TEMPB1,W
XORWF   RANDHI,W
SWAPF   RANDHI
ANDLW   0x01
RLF      TEMPB0
RLF      RANDLO
XORWF   RANDLO
RLF      RANDHI
RETEW   0
;      Test suite for FXM3232
TFXM3232
CALL     RANDOM16
MOVF     RANDHI,W
MOVWF   BARGB0
BCF     BARGB0,MSB
MOVF     BARGB0,W
MOVWF   INDF
INCF    FSR
MOVF     RANDLO,W
MOVWF   BARGB1
MOVWF   INDF
INCF    FSR
CALL     RANDOM16
MOVF     RANDHI,W
MOVWF   BARGB2
MOVWF   INDF
INCF    FSR
MOVF     RANDLO,W
MOVWF   BARGB3
MOVWF   INDF
INCF    FSR
CALL     RANDOM16
MOVF     RANDHI,W
MOVWF   AARGB0
BCF     AARGB0,MSB
MOVF     AARGB0,W
MOVWF   INDF
INCF    FSR
MOVF     RANDLO,W
MOVWF   AARGB1
MOVWF   INDF
INCF    FSR
CALL     RANDOM16
MOVF     RANDHI,W
MOVWF   AARGB2
MOVWF   INDF
INCF    FSR
MOVF     RANDLO,W
MOVWF   AARGB3
MOVWF   INDF
INCF    FSR
CALL     FXM3131U
MOVF     AARGB0,W
MOVWF   INDF
INCF    FSR
MOVF     AARGB1,W
MOVWF   INDF
INCF    FSR
MOVF     AARGB2,W
MOVWF   INDF
INCF    FSR
MOVF     AARGB4,W
MOVWF   INDF

```

AN617

```

    INCF          FSR
    MOVF         AARGB5,W
    MOVWF       INDF
    INCF          FSR
    MOVF         AARGB6,W
    MOVWF       INDF
    INCF          FSR
    MOVF         AARGB7,W
    MOVWF       INDF
    INCF          FSR
    RETLW        0x00
;*****
;*****
;    32x32 Bit Multiplication Macros
SMUL3232L    macro
;    Max Timing:    2+13+6*26+25+2+7*27+26+2+7*28+27+2+6*29+28+9 = 851 clks
;    Min Timing:    2+7*6+5+1+7*6+5+1+7*6+5+2+6*6+5+6 = 192 clks
;    PM: 31+25+2+26+2+27+2+28+9 = 152          DM: 17
    MOVLW        0x8
    MOVWF       LOOPCOUNT
LOOPSM3232A
    RRF          BARGB3
    BTFSC        _C
    GOTO         ALSM3232NA
    DECFSZ       LOOPCOUNT
    GOTO         LOOPSM3232A
    MOVWF       LOOPCOUNT
LOOPSM3232B
    RRF          BARGB2
    BTFSC        _C
    GOTO         BLSM3232NA
    DECFSZ       LOOPCOUNT
    GOTO         LOOPSM3232B
    MOVWF       LOOPCOUNT
LOOPSM3232C
    RRF          BARGB1
    BTFSC        _C
    GOTO         CLSM3232NA
    DECFSZ       LOOPCOUNT
    GOTO         LOOPSM3232C
    MOVLW        0x7
    MOVWF       LOOPCOUNT
LOOPSM3232D
    RRF          BARGB0
    BTFSC        _C
    GOTO         DLSM3232NA
    DECFSZ       LOOPCOUNT
    GOTO         LOOPSM3232D
    CLRF         AARGB0
    CLRF         AARGB1
    CLRF         AARGB2
    CLRF         AARGB3
    RETLW        0x00
ALOOPSM3232
    RRF          BARGB3
    BTFSS        _C
    GOTO         ALSM3232NA
    MOVF         TEMPB3,W
    ADDWF        ACCB3
    MOVF         TEMPB2,W
    BTFSC        _C
    INCF         TEMPB2,W
    ADDWF        ACCB2
    MOVF         TEMPB1,W
    BTFSC        _C
    INCF         TEMPB1,W

```


	ADDWF	ACCB1
	MOVF	TEMPB0,W
	BTFSC	_C
	INCFSZ	TEMPB0,W
ALSM3232NA	ADDWF	ACCB0
	RLF	TEMPB0,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPSM3232
	MOVLW	0x8
	MOVWF	LOOPCOUNT
BLOOPSM3232		
	RRF	BARGB2
	BTFSS	_C
	GOTO	BLSM3232NA
	MOVF	TEMPB3,W
	ADDWF	ACCB3
	MOVF	TEMPB2,W
	BTFSC	_C
	INCFSZ	TEMPB2,W
	ADDWF	ACCB2
	MOVF	TEMPB1,W
	BTFSC	_C
	INCFSZ	TEMPB1,W
	ADDWF	ACCB1
	MOVF	TEMPB0,W
	BTFSC	_C
	INCFSZ	TEMPB0,W
	ADDWF	ACCB0
BLSM3232NA	RLF	TEMPB0,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	RRF	ACCB5
	DECFSZ	LOOPCOUNT
	GOTO	BLOOPSM3232
	MOVLW	0x8
	MOVWF	LOOPCOUNT
CLOOPSM3232		
	RRF	BARGB1
	BTFSS	_C
	GOTO	CLSM3232NA
	MOVF	TEMPB3,W
	ADDWF	ACCB3
	MOVF	TEMPB2,W
	BTFSC	_C
	INCFSZ	TEMPB2,W
	ADDWF	ACCB2
	MOVF	TEMPB1,W
	BTFSC	_C
	INCFSZ	TEMPB1,W
	ADDWF	ACCB1
	MOVF	TEMPB0,W
	BTFSC	_C
	INCFSZ	TEMPB0,W
	ADDWF	ACCB0
CLSM3232NA	RLF	TEMPB0,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2

AN617

```

                RRF          ACCB3
                RRF          ACCB4
                RRF          ACCB5
                RRF          ACCB6
                DECF SZ      LOOPCOUNT
                GOTO         CLOOPSM3232
                MOVLW        0x7
                MOVWF        LOOPCOUNT
DLOOPSM3232
                RRF          BARGB0
                BTFSS        _C
                GOTO         DLSM3232NA
                MOVF         TEMPB3,W
                ADDWF        ACCB3
                MOVF         TEMPB2,W
                BTFSC        _C
                INCF SZ      TEMPB2,W
                ADDWF        ACCB2
                MOVF         TEMPB1,W
                BTFSC        _C
                INCF SZ      TEMPB1,W
                ADDWF        ACCB1
                MOVF         TEMPB0,W
                BTFSC        _C
                INCF SZ      TEMPB0,W
                ADDWF        ACCB0
DLSM3232NA
                RLF          TEMPB0,W
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                RRF          ACCB4
                RRF          ACCB5
                RRF          ACCB6
                RRF          ACCB7
                DECF SZ      LOOPCOUNT
                GOTO         DLOOPSM3232
                RLF          TEMPB0,W
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                RRF          ACCB4
                RRF          ACCB5
                RRF          ACCB6
                RRF          ACCB7
                endm
UMUL3232L      macro
;           Max Timing:      2+15+6*25+24+2+7*26+25+2+7*27+26+2+7*28+27 = 842 clks
;           Min Timing:      2+7*6+5+1+7*6+5+1+7*6+5+1+7*6+5+6 = 197 clks
;           PM: 38+24+2+25+2+26+2+27+9 = 155           DM: 17
                MOVLW        0x08
                MOVWF        LOOPCOUNT
LOOPUM3232A
                RRF          BARGB3
                BTFSC        _C
                GOTO         ALUM3232NAP
                DECF SZ      LOOPCOUNT
                GOTO         LOOPUM3232A
                MOVWF        LOOPCOUNT
LOOPUM3232B
                RRF          BARGB2
                BTFSC        _C
                GOTO         BLUM3232NAP
                DECF SZ      LOOPCOUNT
                GOTO         LOOPUM3232B
```

	MOVWF	LOOPCOUNT
LOOPUM3232C	RRF	BARGB1
	BTFSC	_C
	GOTO	CLUM3232NAP
	DECFSZ	LOOPCOUNT
	GOTO	LOOPUM3232C
	MOVWF	LOOPCOUNT
LOOPUM3232D	RRF	BARGB0
	BTFSC	_C
	GOTO	DLUM3232NAP
	DECFSZ	LOOPCOUNT
	GOTO	LOOPUM3232D
	CLRF	AARGB0
	CLRF	AARGB1
	CLRF	AARGB2
	CLRF	AARGB3
	RETLW	0x00
ALUM3232NAP	BCF	_C
	GOTO	ALUM3232NA
BLUM3232NAP	BCF	_C
	GOTO	BLUM3232NA
CLUM3232NAP	BCF	_C
	GOTO	CLUM3232NA
DLUM3232NAP	BCF	_C
	GOTO	DLUM3232NA
ALOOPUM3232	RRF	BARGB3
	BTFSS	_C
	GOTO	ALUM3232NA
	MOVF	TEMPB3 , W
	ADDWF	ACCB3
	MOVF	TEMPB2 , W
	BTFSC	_C
	INCFBSZ	TEMPB2 , W
	ADDWF	ACCB2
	MOVF	TEMPB1 , W
	BTFSC	_C
	INCFBSZ	TEMPB1 , W
	ADDWF	ACCB1
	MOVF	TEMPB0 , W
	BTFSC	_C
	INCFBSZ	TEMPB0 , W
	ADDWF	ACCB0
ALUM3232NA	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPUM3232
	MOVLW	0x08
	MOVWF	LOOPCOUNT
BLOOPUM3232	RRF	BARGB2
	BTFSS	_C
	GOTO	BLUM3232NA
	MOVF	TEMPB3 , W
	ADDWF	ACCB3
	MOVF	TEMPB2 , W
	BTFSC	_C
	INCFBSZ	TEMPB2 , W
	ADDWF	ACCB2
	MOVF	TEMPB1 , W

AN617

	BTFSC	_C
	INCFSSZ	TEMPB1 , W
	ADDWF	ACCB1
	MOVF	TEMPB0 , W
	BTFSC	_C
	INCFSSZ	TEMPB0 , W
	ADDWF	ACCB0
BLUM3232NA		
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	RRF	ACCB5
	DECFSSZ	LOOPCOUNT
	GOTO	BLOOPUM3232
	MOVLW	0x08
	MOVWF	LOOPCOUNT
CLOOPUM3232		
	RRF	BARGB1
	BTFSS	_C
	GOTO	CLUM3232NA
	MOVF	TEMPB3 , W
	ADDWF	ACCB3
	MOVF	TEMPB2 , W
	BTFSC	_C
	INCFSSZ	TEMPB2 , W
	ADDWF	ACCB2
	MOVF	TEMPB1 , W
	BTFSC	_C
	INCFSSZ	TEMPB1 , W
	ADDWF	ACCB1
	MOVF	TEMPB0 , W
	BTFSC	_C
	INCFSSZ	TEMPB0 , W
	ADDWF	ACCB0
CLUM3232NA		
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	RRF	ACCB5
	RRF	ACCB6
	DECFSSZ	LOOPCOUNT
	GOTO	CLOOPUM3232
	MOVLW	0x08
	MOVWF	LOOPCOUNT
DLOOPUM3232		
	RRF	BARGB0
	BTFSS	_C
	GOTO	DLUM3232NA
	MOVF	TEMPB3 , W
	ADDWF	ACCB3
	MOVF	TEMPB2 , W
	BTFSC	_C
	INCFSSZ	TEMPB2 , W
	ADDWF	ACCB2
	MOVF	TEMPB1 , W
	BTFSC	_C
	INCFSSZ	TEMPB1 , W
	ADDWF	ACCB1
	MOVF	TEMPB0 , W
	BTFSC	_C
	INCFSSZ	TEMPB0 , W
	ADDWF	ACCB0

```

DLUM3232NA
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
    RRF          ACCB4
    RRF          ACCB5
    RRF          ACCB6
    RRF          ACCB7
    DECFSZ      LOOPCOUNT
    GOTO        DLOOPUM3232
    endm

UMUL3131L    macro
;    Max Timing:    2+15+6*25+24+2+7*26+25+2+7*27+26+2+6*28+27+8 = 822 clks
;    Min Timing:    2+7*6+5+1+7*6+5+1+7*6+5+2+6*6+5+6 = 192 clks
;    PM: 39+24+2+25+2+26+2+27+8 = 155                DM: 17
    MOVLW      0x8
    MOVWF      LOOPCOUNT

LOOPUM3131A
    RRF          BARGB3
    BTFSC      _C
    GOTO        ALUM3131NAP
    DECFSZ      LOOPCOUNT
    GOTO        LOOPUM3131A
    MOVWF      LOOPCOUNT

LOOPUM3131B
    RRF          BARGB2
    BTFSC      _C
    GOTO        BLUM3131NAP
    DECFSZ      LOOPCOUNT
    GOTO        LOOPUM3131B
    MOVWF      LOOPCOUNT

LOOPUM3131C
    RRF          BARGB1
    BTFSC      _C
    GOTO        CLUM3131NAP
    DECFSZ      LOOPCOUNT
    GOTO        LOOPUM3131C
    MOVLW      0x7
    MOVWF      LOOPCOUNT

LOOPUM3131D
    RRF          BARGB0
    BTFSC      _C
    GOTO        DLUM3131NAP
    DECFSZ      LOOPCOUNT
    GOTO        LOOPUM3131D
    CLRF       AARGB0
    CLRF       AARGB1
    CLRF       AARGB2
    CLRF       AARGB3
    RETLW      0x00

ALUM3131NAP  BCF          _C
             GOTO        ALUM3131NA

BLUM3131NAP  BCF          _C
             GOTO        BLUM3131NA

CLUM3131NAP  BCF          _C
             GOTO        CLUM3131NA

DLUM3131NAP  BCF          _C
             GOTO        DLUM3131NA

ALOOPUM3131
    RRF          BARGB3
    BTFSS      _C

```

AN617

	GOTO	ALUM3131NA
	MOVF	TEMPB3 ,W
	ADDWF	ACCB3
	MOVF	TEMPB2 ,W
	BTFSC	_C
	INCFSZ	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C
	INCFSZ	TEMPB1 ,W
	ADDWF	ACCB1
	MOVF	TEMPB0 ,W
	BTFSC	_C
	INCFSZ	TEMPB0 ,W
	ADDWF	ACCB0
ALUM3131NA		
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPUM3131
	MOVLW	0x08
	MOVWF	LOOPCOUNT
BLOOPUM3131		
	RRF	BARGB2
	BTFSS	_C
	GOTO	BLUM3131NA
	MOVF	TEMPB3 ,W
	ADDWF	ACCB3
	MOVF	TEMPB2 ,W
	BTFSC	_C
	INCFSZ	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C
	INCFSZ	TEMPB1 ,W
	ADDWF	ACCB1
	MOVF	TEMPB0 ,W
	BTFSC	_C
	INCFSZ	TEMPB0 ,W
	ADDWF	ACCB0
BLUM3131NA		
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	RRF	ACCB5
	DECFSZ	LOOPCOUNT
	GOTO	BLOOPUM3131
	MOVLW	0x08
	MOVWF	LOOPCOUNT
CLOOPUM3131		
	RRF	BARGB1
	BTFSS	_C
	GOTO	CLUM3131NA
	MOVF	TEMPB3 ,W
	ADDWF	ACCB3
	MOVF	TEMPB2 ,W
	BTFSC	_C
	INCFSZ	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C

```

                INCFSZ      TEMPB1,W
                ADDWF       ACCB1
                MOVF        TEMPB0,W
                BTFSC       _C
                INCFSZ      TEMPB0,W
                ADDWF       ACCB0
CLUM3131NA
                RRF         ACCB0
                RRF         ACCB1
                RRF         ACCB2
                RRF         ACCB3
                RRF         ACCB4
                RRF         ACCB5
                RRF         ACCB6
                DECFSZ      LOOPCOUNT
                GOTO        CLOOPUM3131
                MOVLW       0x07
                MOVWF       LOOPCOUNT
DLOOPUM3131
                RRF         BARGB0
                BTFSS       _C
                GOTO        DLUM3131NA
                MOVF        TEMPB3,W
                ADDWF       ACCB3
                MOVF        TEMPB2,W
                BTFSC       _C
                INCFSZ      TEMPB2,W
                ADDWF       ACCB2
                MOVF        TEMPB1,W
                BTFSC       _C
                INCFSZ      TEMPB1,W
                ADDWF       ACCB1
                MOVF        TEMPB0,W
                BTFSC       _C
                INCFSZ      TEMPB0,W
                ADDWF       ACCB0
DLUM3131NA
                RRF         ACCB0
                RRF         ACCB1
                RRF         ACCB2
                RRF         ACCB3
                RRF         ACCB4
                RRF         ACCB5
                RRF         ACCB6
                RRF         ACCB7
                DECFSZ      LOOPCOUNT
                GOTO        DLOOPUM3131
                RRF         ACCB0
                RRF         ACCB1
                RRF         ACCB2
                RRF         ACCB3
                RRF         ACCB4
                RRF         ACCB5
                RRF         ACCB6
                RRF         ACCB7
                endm

SMUL3232      macro
;           Max Timing:      9+7*22+8*23+8*24+7*25+9 = 723 clks
;           Min Timing:      62+6 = 68 clks
;           PM: 68+6+7*22+8*23+8*24+7*25+9 = 788           DM: 16
                variable i
                i = 0

                while i < 8

```

AN617

```

    BTFSC          BARGB3,i
    GOTO          SM3232NA#v(i)
    i = i + 1
    endw
    i = 8

    while i < 16

    BTFSC          BARGB2,i-8
    GOTO          SM3232NA#v(i)
    i = i + 1
    endw
    i = 16

    while i < 24

    BTFSC          BARGB1,i-16
    GOTO          SM3232NA#v(i)
    i = i + 1
    endw
    i = 24

    while i < 31

    BTFSC          BARGB0,i-24
    GOTO          SM3232NA#v(i)
    i = i + 1
    endw
    CLRF          ACCB0          ; if we get here, BARG = 0
    CLRF          ACCB1
    CLRF          ACCB2
    CLRF          ACCB3
    RETEW
SM3232NA0      RLF          TEMPB0,W
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
    RRF          ACCB4
    i = 1
    while i < 8
SM3232A#v(i)  BTFSS          BARGB3,i
    GOTO          SM3232NA#v(i)
    MOVF          TEMPB3,W
    ADDWF        ACCB3
    MOVF          TEMPB2,W
    BTFSC        _C
    INCF        TEMPB2,W
    ADDWF        ACCB2
    MOVF          TEMPB1,W
    BTFSC        _C
    INCF        TEMPB1,W
    ADDWF        ACCB1
    MOVF          TEMPB0,W
    BTFSC        _C
    INCF        TEMPB0,W
    ADDWF        ACCB0
SM3232NA#v(i) RLF          TEMPB0,W
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
    RRF          ACCB4
    i = i + 1
    endw
    i = 8
```



```

        while    i < 16
SM3232A#v(i)  BTFSS      BARGB2,i-8
              GOTO      SM3232NA#v(i)
              MOVF      TEMPB3,W
              ADDWF     ACCB3
              MOVF      TEMPB2,W
              BTFSC     _C
              INCFSZ    TEMPB2,W
              ADDWF     ACCB2
              MOVF      TEMPB1,W
              BTFSC     _C
              INCFSZ    TEMPB1,W
              ADDWF     ACCB1
              MOVF      TEMPB0,W
              BTFSC     _C
              INCFSZ    TEMPB0,W
              ADDWF     ACCB0
SM3232NA#v(i) RLF        TEMPB0,W
              RRF        ACCB0
              RRF        ACCB1
              RRF        ACCB2
              RRF        ACCB3
              RRF        ACCB4
              RRF        ACCB5
              i = i + 1
              endw
              i = 16
              while    i < 24
SM3232A#v(i)  BTFSS      BARGB1,i-16
              GOTO      SM3232NA#v(i)
              MOVF      TEMPB3,W
              ADDWF     ACCB3
              MOVF      TEMPB2,W
              BTFSC     _C
              INCFSZ    TEMPB2,W
              ADDWF     ACCB2
              MOVF      TEMPB1,W
              BTFSC     _C
              INCFSZ    TEMPB1,W
              ADDWF     ACCB1
              MOVF      TEMPB0,W
              BTFSC     _C
              INCFSZ    TEMPB0,W
              ADDWF     ACCB0
SM3232NA#v(i) RLF        TEMPB0,W
              RRF        ACCB0
              RRF        ACCB1
              RRF        ACCB2
              RRF        ACCB3
              RRF        ACCB4
              RRF        ACCB5
              RRF        ACCB6
              i = i + 1
              endw
              i = 24
              while    i < 31
SM3232A#v(i)  BTFSS      BARGB0,i-24
              GOTO      SM3232NA#v(i)
              MOVF      TEMPB3,W
              ADDWF     ACCB3
              MOVF      TEMPB2,W
              BTFSC     _C
              INCFSZ    TEMPB2,W
              ADDWF     ACCB2
              MOVF      TEMPB1,W
              BTFSC     _C

```

AN617

```

                INCF SZ      TEMPB1,W
                ADDWF      ACCB1
                MOVF      TEMPB0,W
                BTFSC      _C
                INCF SZ      TEMPB0,W
                ADDWF      ACCB0
SM3232NA#v(i)  RLF      TEMPB0,W
                RRF      ACCB0
                RRF      ACCB1
                RRF      ACCB2
                RRF      ACCB3
                RRF      ACCB4
                RRF      ACCB5
                RRF      ACCB6
                RRF      ACCB7
                i = i + 1
                endw
                RLF      TEMPB0,W
                RRF      ACCB0
                RRF      ACCB1
                RRF      ACCB2
                RRF      ACCB3
                RRF      ACCB4
                RRF      ACCB5
                RRF      ACCB6
                RRF      ACCB7
                endm
UMUL3232      macro
;           Max Timing:      9+8*21+8*22+8*23+8*24 = 729 clks
;           Min Timing:      63+6 = 69 clks
;           PM: 69+6+8*21+8*22+8*23+8*24 = 795           DM: 16
                variable i
                i = 0
                BCF      _C           ; clear carry for first right shift

                while i < 8

                BTFSC      BARGB3,i
                GOTO      UM3232NA#v(i)
                i = i + 1
                endw
                i = 8

                while i < 16

                BTFSC      BARGB2,i-8
                GOTO      UM3232NA#v(i)
                i = i + 1
                endw
                i = 16

                while i < 24

                BTFSC      BARGB1,i-16
                GOTO      UM3232NA#v(i)
                i = i + 1
                endw
                i = 24

                while i < 32

                BTFSC      BARGB0,i-24
                GOTO      UM3232NA#v(i)
                i = i + 1
                endw
                CLRf      ACCB0           ; if we get here, BARG = 0

```

```

                CLRFB          ACCB1
                CLRFB          ACCB2
                CLRFB          ACCB3
                RETEW          0
UM3232NA0      RRF            ACCB0
                RRF            ACCB1
                RRF            ACCB2
                RRF            ACCB3
                RRF            ACCB4
                i = 1
                while i < 8
UM3232A#v(i)   BTFSS          BARGB3,i
                GOTO          UM3232NA#v(i)
                MOVF           TEMPB3,W
                ADDWF          ACCB3
                MOVF           TEMPB2,W
                BTFSC          _C
                INCFSZ        TEMPB2,W
                ADDWF          ACCB2
                MOVF           TEMPB1,W
                BTFSC          _C
                INCFSZ        TEMPB1,W
                ADDWF          ACCB1
                MOVF           TEMPB0,W
                BTFSC          _C
                INCFSZ        TEMPB0,W
                ADDWF          ACCB0
UM3232NA#v(i)  RRF            ACCB0
                RRF            ACCB1
                RRF            ACCB2
                RRF            ACCB3
                RRF            ACCB4
                i = i + 1
                endw
                i = 8
                while i < 16
UM3232A#v(i)   BTFSS          BARGB2,i-8
                GOTO          UM3232NA#v(i)
                MOVF           TEMPB3,W
                ADDWF          ACCB3
                MOVF           TEMPB2,W
                BTFSC          _C
                INCFSZ        TEMPB2,W
                ADDWF          ACCB2
                MOVF           TEMPB1,W
                BTFSC          _C
                INCFSZ        TEMPB1,W
                ADDWF          ACCB1
                MOVF           TEMPB0,W
                BTFSC          _C
                INCFSZ        TEMPB0,W
                ADDWF          ACCB0
UM3232NA#v(i)  RRF            ACCB0
                RRF            ACCB1
                RRF            ACCB2
                RRF            ACCB3
                RRF            ACCB4
                RRF            ACCB5
                i = i + 1
                endw
                i = 16
                while i < 24
UM3232A#v(i)   BTFSS          BARGB1,i-16
                GOTO          UM3232NA#v(i)
                MOVF           TEMPB3,W
                ADDWF          ACCB3

```

AN617

```

        MOVF          TEMPB2,W
        BTFSC        _C
        INCFSZ       TEMPB2,W
        ADDWF        ACCB2
        MOVF          TEMPB1,W
        BTFSC        _C
        INCFSZ       TEMPB1,W
        ADDWF        ACCB1
        MOVF          TEMPB0,W
        BTFSC        _C
        INCFSZ       TEMPB0,W
        ADDWF        ACCB0
UM3232NA#v(i)  RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               RRF          ACCB3
               RRF          ACCB4
               RRF          ACCB5
               RRF          ACCB6
               i = i + 1
               endw
               i = 24
               while    i < 32
UM3232A#v(i)  BTFSS        BARGB0,i-24
               GOTO      UM3232NA#v(i)
               MOVF          TEMPB3,W
               ADDWF        ACCB3
               MOVF          TEMPB2,W
               BTFSC        _C
               INCFSZ       TEMPB2,W
               ADDWF        ACCB2
               MOVF          TEMPB1,W
               BTFSC        _C
               INCFSZ       TEMPB1,W
               ADDWF        ACCB1
               MOVF          TEMPB0,W
               BTFSC        _C
               INCFSZ       TEMPB0,W
UM3232NA#v(i)  ADDWF        ACCB0
               RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               RRF          ACCB3
               RRF          ACCB4
               RRF          ACCB5
               RRF          ACCB6
               RRF          ACCB7
               i = i + 1
               endw
               endm
UMUL3131      macro
;           Max Timing:    9+7*21+8*22+8*23+7*24+9 = 693 clks
;           Min Timing:    62+6 = 68 clks
;           PM: 68+6+7*22+8*23+8*24+7*25+9 = 788           DM: 16
               variable i
               i = 0
               BCF          _C           ; clear carry for first right shift
               while i < 8

               BTFSC        BARGB3,i
               GOTO      UM3131NA#v(i)
               i = i + 1
               endw
               i = 8
               while i < 16
```

```

    BTFSC          BARGB2,i-8
    GOTO          UM3131NA#v(i)
    i = i + 1
    endw
    i = 16
    while i < 24

    BTFSC          BARGB1,i-16
    GOTO          UM3131NA#v(i)
    i = i + 1
    endw
    i = 24
    while i < 31

    BTFSC          BARGB0,i-24
    GOTO          UM3131NA#v(i)
    i = i + 1
    endw
    CLRF          ACCB0          ; if we get here, BARG = 0
    CLRF          ACCB1
    CLRF          ACCB2
    CLRF          ACCB3
    RETEW          0
UM3131NA0      RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               RRF          ACCB3
               RRF          ACCB4
    i = 1
    while i < 8
UM3131A#v(i)  BTFSS          BARGB3,i
               GOTO          UM3131NA#v(i)
               MOVF          TEMPB3,W
               ADDWF          ACCB3
               MOVF          TEMPB2,W
               BTFSC          _C
               INCF          TEMPB2,W
               ADDWF          ACCB2
               MOVF          TEMPB1,W
               BTFSC          _C
               INCF          TEMPB1,W
               ADDWF          ACCB1
               MOVF          TEMPB0,W
               BTFSC          _C
               INCF          TEMPB0,W
               ADDWF          ACCB0
UM3131NA#v(i) RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               RRF          ACCB3
               RRF          ACCB4
    i = i + 1
    endw
    i = 8
    while i < 16
UM3131A#v(i)  BTFSS          BARGB2,i-8
               GOTO          UM3131NA#v(i)
               MOVF          TEMPB3,W
               ADDWF          ACCB3
               MOVF          TEMPB2,W
               BTFSC          _C
               INCF          TEMPB2,W
               ADDWF          ACCB2
               MOVF          TEMPB1,W
               BTFSC          _C
               INCF          TEMPB1,W

```

AN617

```

                ADDWF      ACCB1
                MOVF      TEMPB0,W
                BTFSC     _C
                INCFSZ    TEMPB0,W
                ADDWF     ACCB0
UM3131NA#v(i)  RRF       ACCB0
                RRF       ACCB1
                RRF       ACCB2
                RRF       ACCB3
                RRF       ACCB4
                RRF       ACCB5
                i = i + 1
                endw
                i = 16
                while    i < 24
                BTFSS    BARGB1,i-16
                GOTO     UM3131NA#v(i)
UM3131A#v(i)  MOVF      TEMPB3,W
                ADDWF     ACCB3
                MOVF      TEMPB2,W
                BTFSC     _C
                INCFSZ    TEMPB2,W
                ADDWF     ACCB2
                MOVF      TEMPB1,W
                BTFSC     _C
                INCFSZ    TEMPB1,W
                ADDWF     ACCB1
                MOVF      TEMPB0,W
                BTFSC     _C
                INCFSZ    TEMPB0,W
                ADDWF     ACCB0
UM3131NA#v(i)  RRF       ACCB0
                RRF       ACCB1
                RRF       ACCB2
                RRF       ACCB3
                RRF       ACCB4
                RRF       ACCB5
                RRF       ACCB6
                i = i + 1
                endw
                i = 24
                while    i < 31
                BTFSS    BARGB0,i-24
                GOTO     UM3131NA#v(i)
UM3131A#v(i)  MOVF      TEMPB3,W
                ADDWF     ACCB3
                MOVF      TEMPB2,W
                BTFSC     _C
                INCFSZ    TEMPB2,W
                ADDWF     ACCB2
                MOVF      TEMPB1,W
                BTFSC     _C
                INCFSZ    TEMPB1,W
                ADDWF     ACCB1
                MOVF      TEMPB0,W
                BTFSC     _C
                INCFSZ    TEMPB0,W
                ADDWF     ACCB0
UM3131NA#v(i)  RRF       ACCB0
                RRF       ACCB1
                RRF       ACCB2
                RRF       ACCB3
                RRF       ACCB4
                RRF       ACCB5
                RRF       ACCB6
                RRF       ACCB7
```

```

        i = i + 1
    endw
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
    RRF          ACCB4
    RRF          ACCB5
    RRF          ACCB6
    RRF          ACCB7
    endm

;*****
;*****
;      32x32 Bit Signed Fixed Point Multiply 32x32 -> 64
;      Input:  32 bit signed fixed point multiplicand in AARGB0
;              32 bit signed fixed point multiplier in BARGB0
;      Use:    CALL    FXM3232S
;      Output: 64 bit signed fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing: 15+851+2 = 868 clks          B > 0
;                  36+851+2 = 889 clks          B < 0
;      Min Timing: 15+192 = 207 clks
;      PM: 36+152+1 = 189          DM: 17
FXM3232S    BTFSS          BARGB0,MSB
            GOTO          M3232SOK
            COMF          BARGB3          ; make multiplier BARG > 0
            INCF          BARGB3
            BTFSC         _Z
            DECF          BARGB2
            COMF          BARGB2
            BTFSC         _Z
            DECF          BARGB1
            COMF          BARGB1
            BTFSC         _Z
            DECF          BARGB0
            COMF          BARGB0
            COMF          AARGB3
            INCF          AARGB3
            BTFSC         _Z
            DECF          AARGB2
            COMF          AARGB2
            BTFSC         _Z
            DECF          AARGB1
            COMF          AARGB1
            BTFSC         _Z
            DECF          AARGB0
            COMF          AARGB0
M3232SOK    CLRF          ACCB4          ; clear partial product
            CLRF          ACCB5
            CLRF          ACCB7
            CLRF          ACCB6
            MOVF          AARGB0,W
            MOVWF         TEMPB0
            MOVF          AARGB1,W
            MOVWF         TEMPB1
            MOVF          AARGB2,W
            MOVWF         TEMPB2
            MOVF          AARGB3,W
            MOVWF         TEMPB3
            SMUL3232L
            RETLW         0x00
;*****
;*****
;      32x32 Bit Unsigned Fixed Point Multiply 32x32 -> 64
;      Input:  32 bit unsigned fixed point multiplicand in AARGB0

```

AN617

```
;          32 bit unsigned fixed point multiplier in BARGB0
;   Use:   CALL   FXM3232U
;   Output: 64 bit unsigned fixed point product in AARGB0
;   Result: AARG <-- AARG x BARG
;   Max Timing:    12+842+2 = 856 clks
;   Min Timing:    12+197 = 209 clks
;   PM: 12+155+1 = 168          DM: 17
FXM3232U
        CLRF      ACCB4          ; clear partial product
        CLRF      ACCB5
        CLRF      ACCB7
        CLRF      ACCB6
        MOVF      AARGB0,W
        MOVWF     TEMPB0
        MOVF      AARGB1,W
        MOVWF     TEMPB1
        MOVF      AARGB2,W
        MOVWF     TEMPB2
        MOVF      AARGB3,W
        MOVWF     TEMPB3
        UMUL3232L
        RETLW     0x00
;*****
;*****

;   31x31 Bit Unsigned Fixed Point Divide 31x31 -> 62
;   Input:  31 bit unsigned fixed point multiplicand in AARGB0
;           31 bit unsigned fixed point multiplier in BARGB0
;   Use:   CALL   FXM3131U
;   Output: 62 bit unsigned fixed point product in AARGB0
;   Result: AARG <-- AARG x BARG
;   Max Timing:    12+822+2 = 836 clks
;   Min Timing:    12+192 = 204 clks
;   PM: 12+155+1 = 168          DM: 17
FXM3131U
        CLRF      ACCB4          ; clear partial product
        CLRF      ACCB5
        CLRF      ACCB7
        CLRF      ACCB6
        MOVF      AARGB0,W
        MOVWF     TEMPB0
        MOVF      AARGB1,W
        MOVWF     TEMPB1
        MOVF      AARGB2,W
        MOVWF     TEMPB2
        MOVF      AARGB3,W
        MOVWF     TEMPB3
        UMUL3131L
        RETLW     0x00
;*****
;*****

        END
```

C.2 32x24 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

```
;   32x24 PIC16 FIXED POINT MULTIPLY ROUTINES          VERSION 1.2
;   Input:  fixed point arguments in AARG and BARG
;   Output: product AARGxBARG in AARG
;   All timings are worst case cycle counts
;   It is useful to note that the additional unsigned routines requiring a non-power of two
;   argument can be called in a signed multiply application where it is known that the
;   respective argument is nonnegative, thereby offering some improvement in
;   performance.
;
;   Routine      Clocks      Function
;   FXM3224S     652         32x24 -> 56 bit signed fixed point multiply
;   FXM3224U     630         32x24 -> 56 bit unsigned fixed point multiply
;   FXM3123U     610         31x23 -> 54 bit unsigned fixed point multiply
```



```

;      The above timings are based on the looped macros. If space permits,
;      approximately 80-97 clocks can be saved by using the unrolled macros.
;          list      r=dec,x=on,t=off
;          include <PIC16.INC>      ; general PIC16 definitions
;          include <MATH16.INC>    ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI      equ      0x1A      ; random number generator registers
RANDLO      equ      0x1B
DATA        equ      0x20      ; beginning of test data
;*****
;*****
;      Test suite for 32x24 bit fixed point multiply algorithms
MAIN        org          0x0005
            MOVLW        RAMSTART
            MOVWF        FSR
MEMLOOP     CLRF         INDF
            INCF         FSR
            MOVLW        RAMSTOP
            SUBWF        FSR,W
            BTFSS        _Z
            GOTO         MEMLOOP
            MOVLW        0x45          ; seed for random numbers
            MOVWF        RANDLO
            MOVLW        0x30
            MOVWF        RANDHI
            MOVLW        DATA
            MOVWF        FSR
            BCF          _RP0
            BCF          _RP1
            BCF          _IRP
            CALL         TFXM3224
SELF        GOTO         SELF
RANDOM16     RLF          RANDHI,W      ; random number generator
            XORWF        RANDHI,W
            MOVWF        TEMPB0
            SWAPF        RANDHI
            SWAPF        RANDLO,W
            MOVWF        TEMPB1
            RLF          TEMPB1,W
            RLF          TEMPB1
            MOVF         TEMPB1,W
            XORWF        RANDHI,W
            SWAPF        RANDHI
            ANDLW        0x01
            RLF          TEMPB0
            RLF          RANDLO
            XORWF        RANDLO
            RLF          RANDHI

            RETEW        0
;      Test suite for FXM3224
TFXM3224   CALL         RANDOM16
            MOVF         RANDHI,W
            MOVWF        BARGB0
;      BCF          BARGB0,MSB
;      MOVF         BARGB0,W
            MOVWF        INDF
            INCF         FSR
            MOVF         RANDLO,W
            MOVWF        BARGB1
            MOVWF        INDF
            INCF         FSR
            CALL         RANDOM16

```

AN617

```
MOVF          RANDHI ,W
MOVWF        BARGB2
MOVWF        INDF
INCF         FSR
CALL         RANDOM16
MOVF          RANDHI ,W
MOVWF        AARGB0
;
;          BCF          AARGB0 ,MSB
MOVF          AARGB0 ,W
MOVWF        INDF
INCF         FSR
MOVF          RANDLO ,W
MOVWF        AARGB1
MOVWF        INDF
INCF         FSR
CALL         RANDOM16
MOVF          RANDHI ,W
MOVWF        AARGB2
MOVWF        INDF
INCF         FSR
MOVF          RANDLO ,W
MOVWF        AARGB3
MOVWF        INDF
INCF         FSR
CALL         FXM3224S
MOVF          AARGB0 ,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB1 ,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB2 ,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB3 ,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB4 ,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB5 ,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB6 ,W
MOVWF        INDF
INCF         FSR
RETLW        0x00
;*****
;*****
;          32x24 Bit Multiplication Macros
SMUL3224L    macro
;          Max Timing:      2+13+6*26+25+2+7*27+26+2+6*28+27+8 = 618 clks
;          Min Timing:      2+7*6+5+1+7*6+5+2+6*6+5+6 = 146 clks
;          PM: 25+25+2+26+2+27+8 = 115          DM: 15
                MOVLW        0x8
                MOVWF        LOOPCOUNT
LOOPSM3224A
                RRF          BARGB2
                BTFSC        _C
                GOTO         ALSM3224NA
                DECFSZ       LOOPCOUNT
                GOTO         LOOPSM3224A
                MOVWF        LOOPCOUNT
LOOPSM3224B
                RRF          BARGB1
                BTFSC        _C
```

	GOTO	BLSM3224NA
	DECFSZ	LOOPCOUNT
	GOTO	LOOPSM3224B
	MOVLW	0x7
LOOPSM3224C	MOVWF	LOOPCOUNT
	RRF	BARGB0
	BTFSC	_C
	GOTO	CLSM3224NA
	DECFSZ	LOOPCOUNT
	GOTO	LOOPSM3224C
	CLRF	AARGB0
	CLRF	AARGB1
	CLRF	AARGB2
	CLRF	AARGB3
ALOOPSM3224	RETLW	0x00
	RRF	BARGB2
	BTFSS	_C
	GOTO	ALSM3224NA
	MOVF	TEMPB3 ,W
	ADDWF	ACCB3
	MOVF	TEMPB2 ,W
	BTFSC	_C
	INCFSSZ	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C
	INCFSSZ	TEMPB1 ,W
	ADDWF	ACCB1
	MOVF	TEMPB0 ,W
	BTFSC	_C
	INCFSSZ	TEMPB0 ,W
	ADDWF	ACCB0
ALSM3224NA	RLF	TEMPB0 ,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPSM3224
	MOVLW	0x8
BLOOPSM3224	MOVWF	LOOPCOUNT
	RRF	BARGB1
	BTFSS	_C
	GOTO	BLSM3224NA
	MOVF	TEMPB3 ,W
	ADDWF	ACCB3
	MOVF	TEMPB2 ,W
	BTFSC	_C
	INCFSSZ	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C
	INCFSSZ	TEMPB1 ,W
	ADDWF	ACCB1
	MOVF	TEMPB0 ,W
	BTFSC	_C
	INCFSSZ	TEMPB0 ,W
	ADDWF	ACCB0
BLSM3224NA	RLF	TEMPB0 ,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2

AN617

```

RRF          ACCB3
RRF          ACCB4
RRF          ACCB5
DECFSZ      LOOPCOUNT
GOTO        BLOOPSM3224
MOVLW      0x7
MOVWF      LOOPCOUNT

CLOOPSM3224
RRF          BARGB0
BTFSS      _C
GOTO        CLSM3224NA
MOVF       TEMPB3 ,W
ADDWF      ACCB3
MOVF       TEMPB2 ,W
BTFSC      _C
INCFSZ     TEMPB2 ,W
ADDWF      ACCB2
MOVF       TEMPB1 ,W
BTFSC      _C
INCFSZ     TEMPB1 ,W
ADDWF      ACCB1
MOVF       TEMPB0 ,W
BTFSC      _C
INCFSZ     TEMPB0 ,W
ADDWF      ACCB0

CLSM3224NA
RLF        TEMPB0 ,W
RRF        ACCB0
RRF        ACCB1
RRF        ACCB2
RRF        ACCB3
RRF        ACCB4
RRF        ACCB5
RRF        ACCB6
DECFSZ     LOOPCOUNT
GOTO        CLOOPSM3224
RLF        TEMPB0 ,W
RRF        ACCB0
RRF        ACCB1
RRF        ACCB2
RRF        ACCB3
RRF        ACCB4
RRF        ACCB5
RRF        ACCB6
endm

UMUL3224L
macro
;      Max Timing:      2+15+6*25+24+2+7*26+25+2+7*27+26 = 617 clks
;      Min Timing:      2+7*6+5+1+7*6+5+1+7*6+5+6 = 151 clks
;      PM: 31+24+2+25+2+26+2+27 = 139          DM: 15
MOVLW     0x08
MOVWF     LOOPCOUNT

LOOPUM3224A
RRF       BARGB2
BTFSC     _C
GOTO      ALUM3224NAP
DECFSZ    LOOPCOUNT
GOTO      LOOPUM3224A
MOVWF     LOOPCOUNT

LOOPUM3224B
RRF       BARGB1
BTFSC     _C
GOTO      BLUM3224NAP
DECFSZ    LOOPCOUNT
GOTO      LOOPUM3224B
MOVWF     LOOPCOUNT

LOOPUM3224C
RRF       BARGB0
```

	BTFSC	_C
	GOTO	CLUM3224NAP
	DECFSZ	LOOPCOUNT
	GOTO	LOOPUM3224C
	CLRF	AARGB0
	CLRF	AARGB1
	CLRF	AARGB2
	CLRF	AARGB3
	RETLW	0x00
ALUM3224NAP	BCF	_C
	GOTO	ALUM3224NA
BLUM3224NAP	BCF	_C
	GOTO	BLUM3224NA
CLUM3224NAP	BCF	_C
	GOTO	CLUM3224NA
ALOOPUM3224	RRF	BARGB2
	BTFSS	_C
	GOTO	ALUM3224NA
	MOVF	TEMPB3 , W
	ADDWF	ACCB3
	MOVF	TEMPB2 , W
	BTFSC	_C
	INCFSZ	TEMPB2 , W
	ADDWF	ACCB2
	MOVF	TEMPB1 , W
	BTFSC	_C
	INCFSZ	TEMPB1 , W
	ADDWF	ACCB1
	MOVF	TEMPB0 , W
	BTFSC	_C
	INCFSZ	TEMPB0 , W
	ADDWF	ACCB0
ALUM3224NA	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPUM3224
	MOVLW	0x08
	MOVWF	LOOPCOUNT
BLOOPUM3224	RRF	BARGB1
	BTFSS	_C
	GOTO	BLUM3224NA
	MOVF	TEMPB3 , W
	ADDWF	ACCB3
	MOVF	TEMPB2 , W
	BTFSC	_C
	INCFSZ	TEMPB2 , W
	ADDWF	ACCB2
	MOVF	TEMPB1 , W
	BTFSC	_C
	INCFSZ	TEMPB1 , W
	ADDWF	ACCB1
	MOVF	TEMPB0 , W
	BTFSC	_C
	INCFSZ	TEMPB0 , W
	ADDWF	ACCB0
BLUM3224NA	RRF	ACCB0

AN617

```

RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
RRF          ACCB5
DECFSZ      LOOPCOUNT
GOTO        BLOOPUM3224
MOVLW      0x08
MOVWF      LOOPCOUNT
CLOOPUM3224
RRF          BARGB0
BTFSS      _C
GOTO        CLUM3224NA
MOVF      TEMPB3,W
ADDWF      ACCB3
MOVF      TEMPB2,W
BTFSC      _C
INCF      TEMPB2,W
ADDWF      ACCB2
MOVF      TEMPB1,W
BTFSC      _C
INCF      TEMPB1,W
ADDWF      ACCB1
MOVF      TEMPB0,W
BTFSC      _C
INCF      TEMPB0,W
ADDWF      ACCB0
CLUM3224NA
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
RRF          ACCB5
RRF          ACCB6
DECFSZ      LOOPCOUNT
GOTO        CLOOPUM3224
endm
UMUL3123L
macro
;      Max Timing:      2+15+6*25+24+2+7*26+25+2+6*27+26+7 = 597 clks
;      Min Timing:      2+7*6+5+1+7*6+5+2+6*6+5+6 = 146 clks
;      PM: 31+24+2+25+2+26+7 = 117          DM: 15
MOVLW      0x8
MOVWF      LOOPCOUNT
LOOPUM3123A
RRF          BARGB2
BTFSC      _C
GOTO        ALUM3123NAP
DECFSZ      LOOPCOUNT
GOTO        LOOPUM3123A
MOVWF      LOOPCOUNT
LOOPUM3123B
RRF          BARGB1
BTFSC      _C
GOTO        BLUM3123NAP
DECFSZ      LOOPCOUNT
GOTO        LOOPUM3123B
MOVLW      0x7
MOVWF      LOOPCOUNT
LOOPUM3123C
RRF          BARGB0
BTFSC      _C
GOTO        CLUM3123NAP
DECFSZ      LOOPCOUNT
GOTO        LOOPUM3123C
CLRF      AARGB0
```

	CLRF	AARGB1
	CLRF	AARGB2
	CLRF	AARGB3
	RETLW	0x00
ALUM3123NAP	BCF	_C
	GOTO	ALUM3123NA
BLUM3123NAP	BCF	_C
	GOTO	BLUM3123NA
CLUM3123NAP	BCF	_C
	GOTO	CLUM3123NA
ALOOPUM3123	RRF	BARGB2
	BTFSS	_C
	GOTO	ALUM3123NA
	MOVF	TEMPB3 ,W
	ADDWF	ACCB3
	MOVF	TEMPB2 ,W
	BTFSC	_C
	INCFSZ	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C
	INCFSZ	TEMPB1 ,W
	ADDWF	ACCB1
	MOVF	TEMPB0 ,W
	BTFSC	_C
	INCFSZ	TEMPB0 ,W
	ADDWF	ACCB0
ALUM3123NA	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPUM3123
	MOVLW	0x08
	MOVWF	LOOPCOUNT
BLOOPUM3123	RRF	BARGB1
	BTFSS	_C
	GOTO	BLUM3123NA
	MOVF	TEMPB3 ,W
	ADDWF	ACCB3
	MOVF	TEMPB2 ,W
	BTFSC	_C
	INCFSZ	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C
	INCFSZ	TEMPB1 ,W
	ADDWF	ACCB1
	MOVF	TEMPB0 ,W
	BTFSC	_C
	INCFSZ	TEMPB0 ,W
	ADDWF	ACCB0
BLUM3123NA	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	RRF	ACCB5

AN617

```

                DECFSZ      LOOPCOUNT
                GOTO       BLOOPUM3123
                MOVLW     0x07
                MOVWF     LOOPCOUNT
CLOOPUM3123
                RRF       BARGB0
                BTFSS    _C
                GOTO     CLUM3123NA
                MOVF     TEMPB3,W
                ADDWF    ACCB3
                MOVF     TEMPB2,W
                BTFSC   _C
                INCF    TEMPB2,W
                ADDWF    ACCB2
                MOVF     TEMPB1,W
                BTFSC   _C
                INCF    TEMPB1,W
                ADDWF    ACCB1
                MOVF     TEMPB0,W
                BTFSC   _C
                INCF    TEMPB0,W
                ADDWF    ACCB0
CLUM3123NA
                RRF       ACCB0
                RRF       ACCB1
                RRF       ACCB2
                RRF       ACCB3
                RRF       ACCB4
                RRF       ACCB5
                RRF       ACCB6
                DECFSZ    LOOPCOUNT
                GOTO     CLOOPUM3123
                RRF       ACCB0
                RRF       ACCB1
                RRF       ACCB2
                RRF       ACCB3
                RRF       ACCB4
                RRF       ACCB5
                RRF       ACCB6
                endm

SMUL3224      macro
;           Max Timing:    9+7*22+8*23+7*24+8 = 523 clks
;           Min Timing:    40+6 = 46 clks
;           PM: 46+6+7*22+8*23+7*24+8 = 566           DM: 14
                variable i
                i = 0

                while i < 8

                BTFSC    BARGB2,i
                GOTO     SM3224NA#v(i)
                i = i + 1
                endw
                i = 8

                while i < 16

                BTFSC    BARGB1,i-8
                GOTO     SM3224NA#v(i)
                i = i + 1
                endw
                i = 16

                while i < 23
```



```

        BTFSC          BARGB0,i-16
        GOTO          SM3224NA#v(i)
        i = i + 1
    endw
    CLRF             ACCB0          ; if we get here, BARG = 0
    CLRF             ACCB1
    CLRF             ACCB2
    CLRF             ACCB3
    RETEW           0
SM3224NA0        RLF             TEMPB0,W
                RRF             ACCB0
                RRF             ACCB1
                RRF             ACCB2
                RRF             ACCB3
                RRF             ACCB4
        i = 1
        while      i < 8
SM3224A#v(i)    BTFSS          BARGB2,i
                GOTO          SM3224NA#v(i)
                MOVF          TEMPB3,W
                ADDWF         ACCB3
                MOVF          TEMPB2,W
                BTFSC         _C
                INCF          TEMPB2,W
                ADDWF         ACCB2
                MOVF          TEMPB1,W
                BTFSC         _C
                INCF          TEMPB1,W
                ADDWF         ACCB1
                MOVF          TEMPB0,W
                BTFSC         _C
                INCF          TEMPB0,W
                ADDWF         ACCB0
SM3224NA#v(i)  RLF             TEMPB0,W
                RRF             ACCB0
                RRF             ACCB1
                RRF             ACCB2
                RRF             ACCB3
                RRF             ACCB4
        i = i + 1
    endw
        i = 8
        while      i < 16
SM3224A#v(i)  BTFSS          BARGB1,i-8
                GOTO          SM3224NA#v(i)
                MOVF          TEMPB3,W
                ADDWF         ACCB3
                MOVF          TEMPB2,W
                BTFSC         _C
                INCF          TEMPB2,W
                ADDWF         ACCB2
                MOVF          TEMPB1,W
                BTFSC         _C
                INCF          TEMPB1,W
                ADDWF         ACCB1
                MOVF          TEMPB0,W
                BTFSC         _C
                INCF          TEMPB0,W
                ADDWF         ACCB0
SM3224NA#v(i) RLF             TEMPB0,W
                RRF             ACCB0
                RRF             ACCB1
                RRF             ACCB2
                RRF             ACCB3
                RRF             ACCB4
                RRF             ACCB5

```

AN617

```

        i = i + 1
    endw
    i = 16
    while i < 23
        BTFSS    BARGB0,i-16
        GOTO    SM3224NA#v(i)
SM3224A#v(i)  MOVF    TEMPB3,W
        ADDWF   ACCB3
        MOVF    TEMPB2,W
        BTFSC   _C
        INCFSZ  TEMPB2,W
        ADDWF   ACCB2
        MOVF    TEMPB1,W
        BTFSC   _C
        INCFSZ  TEMPB1,W
        ADDWF   ACCB1
        MOVF    TEMPB0,W
        BTFSC   _C
        INCFSZ  TEMPB0,W
        ADDWF   ACCB0
SM3224NA#v(i) RLF    TEMPB0,W
        RRF     ACCB0
        RRF     ACCB1
        RRF     ACCB2
        RRF     ACCB3
        RRF     ACCB4
        RRF     ACCB5
        RRF     ACCB6
        i = i + 1
    endw
    RLF     TEMPB0,W
    RRF     ACCB0
    RRF     ACCB1
    RRF     ACCB2
    RRF     ACCB3
    RRF     ACCB4
    RRF     ACCB5
    RRF     ACCB6
    endm
UMUL3224    macro
;           Max Timing:    9+8*21+8*22+8*23 = 537 clks
;           Min Timing:    41+6 = 47 clks
;           PM: 47+6+8*21+8*22+8*23 = 581           DM: 14
        variable i
        i = 0
        BCF     _C           ; clear carry for first right shift

        while i < 8

            BTFSC   BARGB2,i
            GOTO    UM3224NA#v(i)
            i = i + 1
        endw
        i = 8

        while i < 16

            BTFSC   BARGB1,i-8
            GOTO    UM3224NA#v(i)
            i = i + 1
        endw
        i = 16

        while i < 24

            BTFSC   BARGB0,i-16
```

```

        GOTO          UM3224NA#v(i)
        i = i + 1
    endw
    CLRF             ACCB0          ; if we get here, BARG = 0
    CLRF             ACCB1
    CLRF             ACCB2
    CLRF             ACCB3
    RETEW           0
UM3224NA0
    RRF              ACCB0
    RRF              ACCB1
    RRF              ACCB2
    RRF              ACCB3
    RRF              ACCB4
    i = 1
    while i < 8
    BTFSS            BARGB2,i
    GOTO             UM3224NA#v(i)
UM3224A#v(i)
    MOVF             TEMPB3,W
    ADDWF            ACCB3
    MOVF             TEMPB2,W
    BTFSC            _C
    INCF           INCFSZ            TEMPB2,W
    ADDWF            ACCB2
    MOVF             TEMPB1,W
    BTFSC            _C
    INCF           INCFSZ            TEMPB1,W
    ADDWF            ACCB1
    MOVF             TEMPB0,W
    BTFSC            _C
    INCF           INCFSZ            TEMPB0,W
    ADDWF            ACCB0
UM3224NA#v(i)
    RRF              ACCB0
    RRF              ACCB1
    RRF              ACCB2
    RRF              ACCB3
    RRF              ACCB4
    i = i + 1
    endw
    i = 8
    while i < 16
    BTFSS            BARGB1,i-8
    GOTO             UM3224NA#v(i)
UM3224A#v(i)
    MOVF             TEMPB3,W
    ADDWF            ACCB3
    MOVF             TEMPB2,W
    BTFSC            _C
    INCF           INCFSZ            TEMPB2,W
    ADDWF            ACCB2
    MOVF             TEMPB1,W
    BTFSC            _C
    INCF           INCFSZ            TEMPB1,W
    ADDWF            ACCB1
    MOVF             TEMPB0,W
    BTFSC            _C
    INCF           INCFSZ            TEMPB0,W
    ADDWF            ACCB0
UM3224NA#v(i)
    RRF              ACCB0
    RRF              ACCB1
    RRF              ACCB2
    RRF              ACCB3
    RRF              ACCB4
    RRF              ACCB5
    i = i + 1
    endw
    i = 16
    while i < 24

```

AN617

```

UM3224A#v(i)    BTFSS          BARGB0,i-16
                GOTO          UM3224NA#v(i)
                MOVF          TEMPB3,W
                ADDWF         ACCB3
                MOVF          TEMPB2,W
                BTFSC         _C
                INCFSZ        TEMPB2,W
                ADDWF         ACCB2
                MOVF          TEMPB1,W
                BTFSC         _C
                INCFSZ        TEMPB1,W
                ADDWF         ACCB1
                MOVF          TEMPB0,W
                BTFSC         _C
                INCFSZ        TEMPB0,W
                ADDWF         ACCB0
UM3224NA#v(i)   RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                RRF           ACCB4
                RRF           ACCB5
                RRF           ACCB6
                i = i + 1
                endw
                endm
UMUL3123        macro
;               Max Timing:    9+7*21+8*22+7*23+7 = 500 clks
;               Min Timing:    41+6 = 47 clks
;               PM: 47+5+7*22+8*23+7*24+7 = 565           DM: 14
                variable i
                i = 0
                BCF          _C           ; clear carry for first right shift
                while i < 8

                BTFSC        BARGB2,i
                GOTO          UM3123NA#v(i)
                i = i + 1
                endw
                i = 8
                while i < 16

                BTFSC        BARGB1,i-8
                GOTO          UM3123NA#v(i)
                i = i + 1
                endw
                i = 16
                while i < 23

                BTFSC        BARGB0,i-16
                GOTO          UM3123NA#v(i)
                i = i + 1
                endw
                CLRF         ACCB0           ; if we get here, BARG = 0
                CLRF         ACCB1
                CLRF         ACCB2
                CLRF         ACCB3
                RETEW        0
UM3123NA0       RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                RRF           ACCB4
                i = 1
                while i < 8
                BTFSS        BARGB2,i
```

```

UM3123A#v(i)      GOTO          UM3123NA#v(i)
                  MOVF          TEMPB3,W
                  ADDWF         ACCB3
                  MOVF          TEMPB2,W
                  BTFSC         _C
                  INCFSZ        TEMPB2,W
                  ADDWF         ACCB2
                  MOVF          TEMPB1,W
                  BTFSC         _C
                  INCFSZ        TEMPB1,W
                  ADDWF         ACCB1
                  MOVF          TEMPB0,W
                  BTFSC         _C
                  INCFSZ        TEMPB0,W
                  ADDWF         ACCB0
UM3123NA#v(i)     RRF           ACCB0
                  RRF           ACCB1
                  RRF           ACCB2
                  RRF           ACCB3
                  RRF           ACCB4
                  i = i + 1
                  endw
                  i = 8
                  while      i < 16
UM3123A#v(i)     BTFSS          BARGB1,i-8
                  GOTO          UM3123NA#v(i)
                  MOVF          TEMPB3,W
                  ADDWF         ACCB3
                  MOVF          TEMPB2,W
                  BTFSC         _C
                  INCFSZ        TEMPB2,W
                  ADDWF         ACCB2
                  MOVF          TEMPB1,W
                  BTFSC         _C
                  INCFSZ        TEMPB1,W
                  ADDWF         ACCB1
                  MOVF          TEMPB0,W
                  BTFSC         _C
                  INCFSZ        TEMPB0,W
                  ADDWF         ACCB0
UM3123NA#v(i)     RRF           ACCB0
                  RRF           ACCB1
                  RRF           ACCB2
                  RRF           ACCB3
                  RRF           ACCB4
                  RRF           ACCB5
                  i = i + 1
                  endw
                  i = 16
                  while      i < 23
UM3123A#v(i)     BTFSS          BARGB0,i-16
                  GOTO          UM3123NA#v(i)
                  MOVF          TEMPB3,W
                  ADDWF         ACCB3
                  MOVF          TEMPB2,W
                  BTFSC         _C
                  INCFSZ        TEMPB2,W
                  ADDWF         ACCB2
                  MOVF          TEMPB1,W
                  BTFSC         _C
                  INCFSZ        TEMPB1,W
                  ADDWF         ACCB1
                  MOVF          TEMPB0,W
                  BTFSC         _C
                  INCFSZ        TEMPB0,W
                  ADDWF         ACCB0

```

AN617

```
UM3123NA#v(i)   RRF           ACCB0
                 RRF           ACCB1
                 RRF           ACCB2
                 RRF           ACCB3
                 RRF           ACCB4
                 RRF           ACCB5
                 RRF           ACCB6
                 i = i + 1
                 endw
                 RRF           ACCB0
                 RRF           ACCB1
                 RRF           ACCB2
                 RRF           ACCB3
                 RRF           ACCB4
                 RRF           ACCB5
                 RRF           ACCB6
                 endm

;*****
;*****

;       32x24 Bit Signed Fixed Point Multiply 32x24 -> 56
;       Input:  32 bit signed fixed point multiplicand in AARGB0, AARGB1,
;               AARGB2, AARGB3
;       24 bit signed fixed point multiplier in BARGB0, BARGB1,
;               BARGB2
;       Use:    CALL    FXM3224S
;       Output: 56 bit signed fixed point product in AARGB0
;       Result: AARG <-- AARG x BARG
;       Max Timing:  14+618+2 = 634 clks           B > 0
;                   32+618+2 = 652 clks           B < 0
;       Min Timing:  14+146 = 160 clks
;       PM: 36+115+1 = 152           DM: 15
FXM3224S        BTFSS        BARGB0,MSB
                GOTO        M3224SOK
                COMF        BARGB2           ; make multiplier BARG > 0
                INCF        BARGB2
                BTFSC        _Z
                DECF        BARGB1
                COMF        BARGB1
                BTFSC        _Z
                DECF        BARGB0
                COMF        BARGB0
                COMF        AARGB3
                INCF        AARGB3
                BTFSC        _Z
                DECF        AARGB2
                COMF        AARGB2
                BTFSC        _Z
                DECF        AARGB1
                COMF        AARGB1
                BTFSC        _Z
                DECF        AARGB0
                COMF        AARGB0
M3224SOK        CLRF        ACCB4           ; clear partial product
                CLRF        ACCB5
                CLRF        ACCB6
                MOVF        AARGB0,W
                MOVWF       TEMPB0
                MOVF        AARGB1,W
                MOVWF       TEMPB1
                MOVF        AARGB2,W
                MOVWF       TEMPB2
                MOVF        AARGB3,W
                MOVWF       TEMPB3
                SMUL3224L
```

```

                RETLW          0x00
;*****
;*****
;
;   32x24 Bit Unsigned Fixed Point Multiply 32x24 -> 56
;   Input:  32 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;           AARGB2, AARGB3
;           24 bit unsigned fixed point multiplier in BARGB0, BARGB1,
;           BARGB2
;   Use:    CALL    FXM3224U
;   Output: 56 bit unsigned fixed point product in AARGB0
;   Result: AARG <-- AARG x BARG
;   Max Timing:    11+617+2 = 630 clks
;   Min Timing:    11+151 = 162 clks
;   PM: 11+139+1 = 151                DM: 15
FXM3224U
                CLRF          ACCB4          ; clear partial product
                CLRF          ACCB5
                CLRF          ACCB6
                MOVF          AARGB0,W
                MOVWF         TEMPB0
                MOVF          AARGB1,W
                MOVWF         TEMPB1
                MOVF          AARGB2,W
                MOVWF         TEMPB2
                MOVF          AARGB3,W
                MOVWF         TEMPB3
                UMUL3224L
                RETLW          0x00
;*****
;*****
;
;   31x23 Bit Unsigned Fixed Point Divide 31x23 -> 54
;   Input:  31 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;           AARGB2, AARGB3
;           23 bit unsigned fixed point multiplier in BARGB0, BARGB1,
;           BARGB2
;   Use:    CALL    FXM3123U
;   Output: 54 bit unsigned fixed point product in AARGB0
;   Result: AARG <-- AARG x BARG
;   Max Timing:    11+597+2 = 610 clks
;   Min Timing:    11+146 = 157 clks
;   PM: 11+117+1 = 129                DM: 15
FXM3123U
                CLRF          ACCB4          ; clear partial product
                CLRF          ACCB5
                CLRF          ACCB6
                MOVF          AARGB0,W
                MOVWF         TEMPB0
                MOVF          AARGB1,W
                MOVWF         TEMPB1
                MOVF          AARGB2,W
                MOVWF         TEMPB2
                MOVF          AARGB3,W
                MOVWF         TEMPB3
                UMUL3123L
                RETLW          0x00
;*****
;*****
                END

```

C.3 32x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

```

;   32x16 PIC16 FIXED POINT MULTIPLY ROUTINES          VERSION 1.2
;   Input:  fixed point arguments in AARG and BARG
;   Output: product AARGxBARG in AARG
;   All timings are worst case cycle counts

```

AN617

```
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed multiply application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;      Routine           Clocks      Function
;      FXM3216S         423          32x16 -> 48 bit signed fixed point multiply
;      FXM3216U         412          32x16 -> 48 bit unsigned fixed point multiply
;      FXM3115U         392          31x15 -> 46 bit unsigned fixed point multiply
;      The above timings are based on the looped macros. If space permits,
;      approximately 65-88 clocks can be saved by using the unrolled macros.
;      list      r=dec,x=on,t=off
;      include <PIC16.INC>      ; general PIC16 definitions
;      include <MATH16.INC>     ; PIC16 math library definitions
;*****
;      Test suite storage
RANDHI      equ      0x1A      ; random number generator registers
RANDLO      equ      0x1B
TESTCOUNT  equ      0x20      ; counter
DATA        equ      0x21      ; beginning of test data
;*****
;      Test suite for 32x16 bit fixed point multiply algorithms
MAIN        org          0x0005
MAIN        MOVLW       RAMSTART
MAIN        MOVWF       FSR
MEMLOOP     CLRF        INDF
MEMLOOP     INCF        FSR
MEMLOOP     MOVLW       RAMSTOP
MEMLOOP     SUBWF       FSR,W
MEMLOOP     BTFSS      _Z
MEMLOOP     GOTO       MEMLOOP
MEMLOOP     MOVLW       0x45          ; seed for random numbers
MEMLOOP     MOVWF       RANDLO
MEMLOOP     MOVLW       0x30
MEMLOOP     MOVWF       RANDHI
MEMLOOP     MOVLW       DATA
MEMLOOP     MOVWF       FSR
MEMLOOP     BCF        _RP0
MEMLOOP     BCF        _RP1
MEMLOOP     BCF        _IRP
MEMLOOP     CALL       TFXM3216
SELF        GOTO       SELF
RANDOM16     RLF        RANDHI,W      ; random number generator
RANDOM16     XORWF      RANDHI,W
RANDOM16     MOVWF     TEMPB0
RANDOM16     SWAPF     RANDHI
RANDOM16     SWAPF     RANDLO,W
RANDOM16     MOVWF     TEMPB1
RANDOM16     RLF      TEMPB1,W
RANDOM16     RLF      TEMPB1
RANDOM16     MOVF     TEMPB1,W
RANDOM16     XORWF   RANDHI,W
RANDOM16     SWAPF   RANDHI
RANDOM16     ANDLW   0x01
RANDOM16     RLF    TEMPB0
RANDOM16     RLF    RANDLO
RANDOM16     XORWF  RANDLO
RANDOM16     RLF   RANDHI
RANDOM16     RETEW  0
;      Test suite for FXM3216
TFXM3216    MOVLW   1
TFXM3216    MOVWF  TESTCOUNT
M3216LOOP  CALL    RANDOM16
```



```

MOVF          RANDHI ,W
MOVWF         BARGB0
BCF           BARGB0 ,MSB
MOVF          BARGB0 ,W
MOVWF         INDF
INCF          FSR
MOVF          RANDLO ,W
MOVWF         BARGB1
MOVWF         INDF
INCF          FSR
CALL          RANDOM16
MOVF          RANDHI ,W
MOVWF         AARGB0
BCF           AARGB0 ,MSB
MOVF          AARGB0 ,W
MOVWF         INDF
INCF          FSR
MOVF          RANDLO ,W
MOVWF         AARGB1
MOVWF         INDF
INCF          FSR
CALL          RANDOM16
MOVF          RANDHI ,W
MOVWF         AARGB2
MOVWF         INDF
INCF          FSR
MOVF          RANDLO ,W
MOVWF         AARGB3
MOVWF         INDF
INCF          FSR
CALL          FXM3115U
MOVF          AARGB0 ,W
MOVWF         INDF
INCF          FSR
MOVF          AARGB1 ,W
MOVWF         INDF
INCF          FSR
MOVF          AARGB2 ,W
MOVWF         INDF
INCF          FSR
MOVF          AARGB3 ,W
MOVWF         INDF
INCF          FSR
MOVF          AARGB4 ,W
MOVWF         INDF
INCF          FSR
MOVF          AARGB5 ,W
MOVWF         INDF
INCF          FSR
DECFSZ       TESTCOUNT
GOTO         M3216LOOP
RETLW        0x00
;*****
;*****
;      32x16 Bit Multiplication Macros
SMUL3216L    macro
;      Max Timing:      2+13+6*26+25+2+6*27+26+7 = 393 clks
;      Min Timing:      2+7*6+5+2+6*6+5+6 = 98 clks
;      PM: 19+60 = 79          DM: 11
                MOVLW          0x8
                MOVWF          LOOPCOUNT
LOOPSM3216A  RRF              BARGB1
                BTFSC          _C
                GOTO          ALSM3216NA
                DECFSZ         LOOPCOUNT

```

AN617

	GOTO	LOOPSM3216A
	MOVLW	0x7
	MOVWF	LOOPCOUNT
LOOPSM3216B		
	RRF	BARGB0
	BTFSC	_C
	GOTO	BLSM3216NA
	DECFSZ	LOOPCOUNT
	GOTO	LOOPSM3216B
	CLRF	AARGB0
	CLRF	AARGB1
	CLRF	AARGB2
	CLRF	AARGB3
	RETLW	0x00
ALOOPSM3216		
	RRF	BARGB1
	BTFSS	_C
	GOTO	ALSM3216NA
	MOVF	TEMPB3,W
	ADDWF	ACCB3
	MOVF	TEMPB2,W
	BTFSC	_C
	INCFBSZ	TEMPB2,W
	ADDWF	ACCB2
	MOVF	TEMPB1,W
	BTFSC	_C
	INCFBSZ	TEMPB1,W
	ADDWF	ACCB1
	MOVF	TEMPB0,W
	BTFSC	_C
	INCFBSZ	TEMPB0,W
	ADDWF	ACCB0
ALSM3216NA	RLF	TEMPB0,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPSM3216
	MOVLW	0x7
	MOVWF	LOOPCOUNT
BLOOPSM3216		
	RRF	BARGB0
	BTFSS	_C
	GOTO	BLSM3216NA
	MOVF	TEMPB3,W
	ADDWF	ACCB3
	MOVF	TEMPB2,W
	BTFSC	_C
	INCFBSZ	TEMPB2,W
	ADDWF	ACCB2
	MOVF	TEMPB1,W
	BTFSC	_C
	INCFBSZ	TEMPB1,W
	ADDWF	ACCB1
	MOVF	TEMPB0,W
	BTFSC	_C
	INCFBSZ	TEMPB0,W
	ADDWF	ACCB0
BLSM3216NA	RLF	TEMPB0,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4

```

RRF          ACCB5
DECFSZ       LOOPCOUNT
GOTO         BLOOFSM3216
RLF          TEMPB0,W
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
RRF          ACCB5
endm
UMUL3216L    macro
;           Max Timing:    2+15+6*25+24+2+7*26+25 = 400 clks
;           Min Timing:    2+7*6+5+1+7*6+5+6 = 103 clks
;           PM: 73         DM: 11
                MOVLW      0x08
                MOVWF      LOOPCOUNT
LOOPUM3216A
                RRF          BARGB1
                BTFSC       _C
                GOTO         ALUM3216NAP
                DECFSZ      LOOPCOUNT
                GOTO         LOOPUM3216A
                MOVWF      LOOPCOUNT
LOOPUM3216B
                RRF          BARGB0
                BTFSC       _C
                GOTO         BLUM3216NAP
                DECFSZ      LOOPCOUNT
                GOTO         LOOPUM3216B
                CLRF        AARGB0
                CLRF        AARGB1
                CLRF        AARGB2
                CLRF        AARGB3
                RETLW       0x00
BLUM3216NAP
                BCF          _C
                GOTO         BLUM3216NA
ALUM3216NAP
                BCF          _C
                GOTO         ALUM3216NA
ALOOPUM3216
                RRF          BARGB1
                BTFSS       _C
                GOTO         ALUM3216NA
                MOVF        TEMPB3,W
                ADDWF       ACCB3
                MOVF        TEMPB2,W
                BTFSC       _C
                INCF       TEMPB2,W
                ADDWF       ACCB2
                MOVF        TEMPB1,W
                BTFSC       _C
                INCF       TEMPB1,W
                ADDWF       ACCB1
                MOVF        TEMPB0,W
                BTFSC       _C
                INCF       TEMPB0,W
                ADDWF       ACCB0
ALUM3216NA
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                RRF          ACCB4
                DECFSZ      LOOPCOUNT

```

AN617

```
GOTO                ALOOPUM3216
MOVLW              0x08
MOVWF              LOOPCOUNT
BLOOPUM3216
RRF                BARGB0
BTFSS              _C
GOTO                BLUM3216NA
MOVF               TEMPB3,W
ADDWF              ACCB3
MOVF               TEMPB2,W
BTFSC              _C
INCFBSZ           TEMPB2,W
ADDWF              ACCB2
MOVF               TEMPB1,W
BTFSC              _C
INCFBSZ           TEMPB1,W
ADDWF              ACCB1
MOVF               TEMPB0,W
BTFSC              _C
INCFBSZ           TEMPB0,W
ADDWF              ACCB0
BLUM3216NA
RRF                ACCB0
RRF                ACCB1
RRF                ACCB2
RRF                ACCB3
RRF                ACCB4
RRF                ACCB5
DECFSZ            LOOPCOUNT
GOTO                BLOOPUM3216
endm
UMUL3115L          macro
;      Max Timing:      2+15+6*25+24+2+6*26+25+6 = 380 clks
;      Min Timing:      2+7*6+5+2+6*6+5+6 = 96 clks
;      PM: 80           DM: 11
MOVLW              0x8
MOVWF              LOOPCOUNT
LOOPUM3115A
RRF                BARGB1
BTFSC              _C
GOTO                ALUM3115NAP
DECFSZ            LOOPCOUNT
GOTO                LOOPUM3115A
MOVLW              0x7
MOVWF              LOOPCOUNT
LOOPUM3115B
RRF                BARGB0
BTFSC              _C
GOTO                BLUM3115NAP
DECFSZ            LOOPCOUNT
GOTO                LOOPUM3115B
CLRF              AARGB0
CLRF              AARGB1
CLRF              AARGB2
CLRF              AARGB3
RETLW             0x00
BLUM3115NAP
BCF                _C
GOTO                BLUM3115NA
ALUM3115NAP
BCF                _C
GOTO                ALUM3115NA
ALOOPUM3115
RRF                BARGB1
BTFSS              _C
GOTO                ALUM3115NA
```

```

MOVF          TEMPB3,W
ADDWF         ACCB3
MOVF          TEMPB2,W
BTFSC        _C
INCFSZ       TEMPB2,W
ADDWF         ACCB2
MOVF          TEMPB1,W
BTFSC        _C
INCFSZ       TEMPB1,W
ADDWF         ACCB1
MOVF          TEMPB0,W
BTFSC        _C
INCFSZ       TEMPB0,W
ADDWF         ACCB0
ALUM3115NA
RRF           ACCB0
RRF           ACCB1
RRF           ACCB2
RRF           ACCB3
RRF           ACCB4
DECFSZ       LOOPCOUNT
GOTO         ALOOPUM3115
MOVLW        0x07
MOVWF        LOOPCOUNT
BLOOPUM3115
RRF           BARGB0
BTFSS        _C
GOTO         BLUM3115NA
MOVF          TEMPB3,W
ADDWF         ACCB3
MOVF          TEMPB2,W
BTFSC        _C
INCFSZ       TEMPB2,W
ADDWF         ACCB2
MOVF          TEMPB1,W
BTFSC        _C
INCFSZ       TEMPB1,W
ADDWF         ACCB1
MOVF          TEMPB0,W
BTFSC        _C
INCFSZ       TEMPB0,W
ADDWF         ACCB0
BLUM3115NA
RRF           ACCB0
RRF           ACCB1
RRF           ACCB2
RRF           ACCB3
RRF           ACCB4
RRF           ACCB5
DECFSZ       LOOPCOUNT
GOTO         BLOOPUM3115
RRF           ACCB0
RRF           ACCB1
RRF           ACCB2
RRF           ACCB3
RRF           ACCB4
RRF           ACCB5
endm

SMUL3216      macro
;           Max Timing:      5+8+7*20+7*21+5 = 305 clks
;           Min Timing:      5+24+21+7 = 57 clks
;           PM: 5+24+21+6+5+7*20+7*21+5 = 353          DM: 10
              variable i
              i = 0
              BTFSC          AARGB0,MSB

```

AN617

```

COMF          ACCB4
MOVF          ACCB4,W
MOVWF        ACCB5
RLF          ACCB0,W

while i < 8

BTFSC        BARGB1,i
GOTO        SM3216NA#v(i)
BCF          ACCB4,7-i
i = i + 1
endw
i = 8

while i < 15

BTFSC        BARGB0,i-8
GOTO        SM3216NA#v(i)
BCF          ACCB5,15-i
i = i + 1
endw
CLRF        ACCB0          ; if we get here, BARG = 0
CLRF        ACCB1
CLRF        ACCB2
CLRF        ACCB3
CLRF        ACCB5
RETEW        0

SM3216NA0
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
i = 1
while i < 8
SM3216A#v(i) BTFSS        BARGB1,i
GOTO        SM3216NA#v(i)
MOVF        TEMPB3,W
ADDWF        ACCB3
MOVF        TEMPB2,W
BTFSC        _C
INCF        TEMPB2,W
ADDWF        ACCB2
MOVF        TEMPB1,W
BTFSC        _C
INCF        TEMPB1,W
ADDWF        ACCB1
MOVF        TEMPB0,W
BTFSC        _C
INCF        TEMPB0,W
ADDWF        ACCB0

SM3216NA#v(i)
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
i = i + 1
endw
i = 8
while i < 15
SM3216A#v(i) BTFSS        BARGB0,i-8
GOTO        SM3216NA#v(i)
MOVF        TEMPB3,W
ADDWF        ACCB3
MOVF        TEMPB2,W
```

```

        BTFSC          _C
        INCFSZ        TEMPB2,W
        ADDWF         ACCB2
        MOVF          TEMPB1,W
        BTFSC          _C
        INCFSZ        TEMPB1,W
        ADDWF         ACCB1
        MOVF          TEMPB0,W
        BTFSC          _C
        INCFSZ        TEMPB0,W
        ADDWF         ACCB0
SM3216NA#v(i)
        RRF           ACCB0
        RRF           ACCB1
        RRF           ACCB2
        RRF           ACCB3
        RRF           ACCB4
        RRF           ACCB5
        i = i + 1
        endw
        RRF           ACCB0
        RRF           ACCB1
        RRF           ACCB2
        RRF           ACCB3
        RRF           ACCB4
        RRF           ACCB5
        endm
UMUL3216
macro
;      Max Timing:      1+8+7*21+8*22 = 332 clks
;      Min Timing:      1+2*8+2*8+6 = 39 clks
;      PM: 1+2*8+2*8+6+7*21+8*22 = 362          DM: 10
        variable i
        i = 0
        BCF           _C          ; clear carry for first right shift

        while i < 8

        BTFSC        BARGB1,i
        GOTO         UM3216NA#v(i)
        i = i + 1
        endw
        i = 8

        while i < 16

        BTFSC        BARGB0,i-8
        GOTO         UM3216NA#v(i)
        i = i + 1
        endw
        CLRF         ACCB0          ; if we get here, BARG = 0
        CLRF         ACCB1
        CLRF         ACCB2
        CLRF         ACCB3
        RETEW        0
UM3216NA0
        RRF           ACCB0
        RRF           ACCB1
        RRF           ACCB2
        RRF           ACCB3
        RRF           ACCB4
        i = 1
        while i < 8
        BTFSS        BARGB1,i
        GOTO         UM3216NA#v(i)
UM3216A#v(i)
        MOVF         TEMPB3,W
        ADDWF         ACCB3
        MOVF         TEMPB2,W

```

AN617

```

    BTFSC          _C
    INCFSZ         TEMPB2,W
    ADDWF          ACCB2
    MOVF           TEMPB1,W
    BTFSC          _C
    INCFSZ         TEMPB1,W
    ADDWF          ACCB1
    MOVF           TEMPB0,W
    BTFSC          _C
    INCFSZ         TEMPB0,W
    ADDWF          ACCB0
UM3216NA#v(i)    RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                RRF           ACCB4
                i = i + 1
                endw
                i = 8
                while i < 16
    BTFSS          BARGB0,i-8
UM3216A#v(i)    GOTO          UM3216NA#v(i)
                MOVF           TEMPB3,W
                ADDWF          ACCB3
                MOVF           TEMPB2,W
                BTFSC          _C
                INCFSZ         TEMPB2,W
                ADDWF          ACCB2
                MOVF           TEMPB1,W
                BTFSC          _C
                INCFSZ         TEMPB1,W
                ADDWF          ACCB1
                MOVF           TEMPB0,W
                BTFSC          _C
                INCFSZ         TEMPB0,W
UM3216NA#v(i)    ADDWF          ACCB0
                RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                RRF           ACCB4
                RRF           ACCB5
                i = i + 1
                endw
                endm
UMUL3115        macro
;           Max Timing:      9+7*21+7*22+6 = 316 clks
;           Min Timing:      1+30+6 = 37 clks
;           PM: 1+30+10+7*21+7*22+6 = 348           DM: 10
                variable i
                i = 0
                BCF           _C           ; clear carry for first right shift
                while i < 8

                BTFSC          BARGB1,i
                GOTO          UM3115NA#v(i)
                i = i + 1
                endw
                i = 8
                while i < 15

                BTFSC          BARGB0,i-8
                GOTO          UM3115NA#v(i)
                i = i + 1
                endw
                CLRf          ACCB0           ; if we get here, BARG = 0

```



```

        CLRF          ACCB1
        CLRF          ACCB2
        CLRF          ACCB3
        RETEW         0
UM3115NA0  RRF          ACCB0
           RRF          ACCB1
           RRF          ACCB2
           RRF          ACCB3
           RRF          ACCB4
           i = 1
           while     i < 8
UM3115A#v(i)  BTFSS         BARGB1,i
           GOTO        UM3115NA#v(i)
           MOVF         TEMPB3,W
           ADDWF        ACCB3
           MOVF         TEMPB2,W
           BTFSC        _C
           INCF        TEMPB2,W
           ADDWF        ACCB2
           MOVF         TEMPB1,W
           BTFSC        _C
           INCF        TEMPB1,W
           ADDWF        ACCB1
           MOVF         TEMPB0,W
           BTFSC        _C
           INCF        TEMPB0,W
           ADDWF        ACCB0
UM3115NA#v(i)  RRF          ACCB0
           RRF          ACCB1
           RRF          ACCB2
           RRF          ACCB3
           RRF          ACCB4
           i = i + 1
           endw
           i = 8
           while     i < 15
UM3115A#v(i)  BTFSS         BARGB0,i-8
           GOTO        UM3115NA#v(i)
           MOVF         TEMPB3,W
           ADDWF        ACCB3
           MOVF         TEMPB2,W
           BTFSC        _C
           INCF        TEMPB2,W
           ADDWF        ACCB2
           MOVF         TEMPB1,W
           BTFSC        _C
           INCF        TEMPB1,W
           ADDWF        ACCB1
           MOVF         TEMPB0,W
           BTFSC        _C
           INCF        TEMPB0,W
           ADDWF        ACCB0
UM3115NA#v(i)  RRF          ACCB0
           RRF          ACCB1
           RRF          ACCB2
           RRF          ACCB3
           RRF          ACCB4
           RRF          ACCB5
           i = i + 1
           endw
           RRF          ACCB0
           RRF          ACCB1
           RRF          ACCB2
           RRF          ACCB3
           RRF          ACCB4
           RRF          ACCB5
           endm

```

AN617

```
*****
*****
;
;      32x16 Bit Signed Fixed Point Multiply 32x16 -> 32
;      Input:  16 bit signed fixed point multiplicand in AARGB0
;              16 bit signed fixed point multiplier in BARGB0
;      Use:    CALL    FXM3216S
;      Output: 32 bit signed fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:  13+393+2 = 408 clks          B > 0
;                  28+393+2 = 423 clks          B < 0
;      Min Timing:  13+98 = 111 clks
;      PM: 18+79+1 = 98          DM: 9
FXM3216S      BTFSS      BARGB0,MSB
              GOTO      M3216SOK
              COMF      BARGB1          ; make multiplier BARG > 0
              INCF      BARGB1
              BTFSC     _Z
              DECF      BARGB0
              COMF      BARGB0
              COMF      AARGB3
              INCF      AARGB3
              BTFSC     _Z
              DECF      AARGB2
              COMF      AARGB2
              BTFSC     _Z
              DECF      AARGB1
              COMF      AARGB1
              BTFSC     _Z
              DECF      AARGB0
              COMF      AARGB0
M3216SOK     CLRF      ACCB4          ; clear partial product
              CLRF      ACCB5
              MOVF      AARGB0,W
              MOVWF     TEMPB0
              MOVF      AARGB1,W
              MOVWF     TEMPB1
              MOVF      AARGB2,W
              MOVWF     TEMPB2
              MOVF      AARGB3,W
              MOVWF     TEMPB3
              SMUL3216L
              RETLW     0x00
;*****
;*****
;
;      32x16 Bit Unsigned Fixed Point Multiply 32x16 -> 32
;      Input:  16 bit unsigned fixed point multiplicand in AARGB0
;              16 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL    FXM3216U
;      Output: 32 bit unsigned fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:  10+400+2 = 412 clks
;      Min Timing:  10+104 = 114 clks
;      PM: 10+73+1 = 84          DM: 9
FXM3216U     CLRF      ACCB4          ; clear partial product
              CLRF      ACCB5
              MOVF      AARGB0,W
              MOVWF     TEMPB0
              MOVF      AARGB1,W
              MOVWF     TEMPB1
              MOVF      AARGB2,W
              MOVWF     TEMPB2
```

```

        MOVF          AARGB3,W
        MOVWF        TEMPB3
        UMUL3216L
        RETLW        0x00
;*****
;*****
;
;   31x15 Bit Unsigned Fixed Point Divide 31x15 -> 30
;   Input:  15 bit unsigned fixed point multiplicand in AARGB0
;           15 bit unsigned fixed point multiplier in BARGB0
;   Use:    CALL    FXM3115U
;   Output: 30 bit unsigned fixed point product in AARGB0
;   Result: AARG <-- AARG x BARG
;   Max Timing: 10+380+2 = 392 clks
;   Min Timing: 10+96 = 106 clks
;   PM: 10+80+1 = 91          DM: 9
FXM3115U
        CLRF          ACCB4          ; clear partial product
        CLRF          ACCB5
        MOVF          AARGB0,W
        MOVWF        TEMPB0
        MOVF          AARGB1,W
        MOVWF        TEMPB1
        MOVF          AARGB2,W
        MOVWF        TEMPB2
        MOVF          AARGB3,W
        MOVWF        TEMPB3
        UMUL3115L
        RETLW        0x00
;*****
;*****
        END

```

C.4 24x24 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

```

;   24x24 PIC16 FIXED POINT MULTIPLY ROUTINES          VERSION 1.2
;   Input:  fixed point arguments in AARG and BARG
;   Output: product AARGxBARG in AARG
;   All timings are worst case cycle counts
;   It is useful to note that the additional unsigned routines requiring a non-power of two
;   argument can be called in a signed multiply application where it is known that the
;   respective argument is nonnegative, thereby offering some improvement in
;   performance.
;
;   Routine      Clocks      Function
;   FXM2424S     535         24x24 -> 48 bit signed fixed point multiply
;   FXM2424U     512         24x24 -> 48 bit unsigned fixed point multiply
;   FXM2323U     497         23x23 -> 46 bit unsigned fixed point multiply
;   The above timings are based on the looped macros. If space permits,
;   approximately 61-95 clocks can be saved by using the unrolled macros.
;
;   list      r=dec,x=on,t=off
;   include <PIC16.INC>      ; general PIC16 definitions
;   include <MATH16.INC>    ; PIC16 math library definitions
;*****
;*****
;   Test suite storage
RANDHI      equ      0x1A      ; random number generator registers
RANDLO      equ      0x1B
DATA        equ      0x20      ; beginning of test data
;*****
;*****
;   Test suite for 24x24 bit fixed point multiply algorithms
MAIN        org          0x0005
           MOVLW        RAMSTART
           MOVWF        FSR
MEMLOOP     CLRF          INDF
           INCF          FSR
           MOVLW        RAMSTOP

```

AN617

```

SUBWF      FSR,W
BTFSS     _Z
GOTO      MEMLOOP
MOVLW     0x45          ; seed for random numbers
MOVWF     RANDLO
MOVLW     0x30
MOVWF     RANDHI
MOVLW     DATA
MOVWF     FSR
BCF       _RP0
BCF       _RP1
BCF       _IRP
CALL      TFXM2424
SELF
RANDOM16   GOTO      SELF
          RLF       RANDHI,W          ; random number generator
          XORWF    RANDHI,W
          MOVWF    TEMPB0
          SWAPF   RANDHI
          SWAPF   RANDLO,W
          MOVWF    TEMPB1
          RLF     TEMPB1,W
          RLF     TEMPB1
          MOVF    TEMPB1,W
          XORWF   RANDHI,W
          SWAPF   RANDHI
          ANDLW   0x01
          RLF     TEMPB0
          RLF     RANDLO
          XORWF   RANDLO
          RLF     RANDHI

          RETEW    0
;          Test suite for FXM2424
TFXM2424
CALL      RANDOM16
MOVF     RANDHI,W
MOVWF    BARGB0
BCF      BARGB0,MSB
MOVF     BARGB0,W
MOVWF    INDF
INCF     FSR
MOVF     RANDLO,W
MOVWF    BARGB1
MOVWF    INDF
INCF     FSR
CALL     RANDOM16
MOVF     RANDHI,W
MOVWF    BARGB2
MOVWF    INDF
INCF     FSR
CALL     RANDOM16
MOVF     RANDHI,W
MOVWF    AARGB0
BCF      AARGB0,MSB
MOVF     AARGB0,W
MOVWF    INDF
INCF     FSR
MOVF     RANDLO,W
MOVWF    AARGB1
MOVWF    INDF
INCF     FSR
CALL     RANDOM16
MOVF     RANDHI,W
MOVWF    AARGB2
MOVWF    INDF
INCF     FSR
```

```

CALL          FXM2323U
MOVF          AARGB0,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB1,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB2,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB3,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB4,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB5,W
MOVWF        INDF
INCF         FSR
RETLW        0x00
;*****
;*****
;      24x24 Bit Multiplication Macros
SMUL2424L    macro
;      Max Timing:      2+12+6*21+20+2+7*22+21+2+6*23+22+7 = 506 clks
;      Min Timing:      2+7*6+5+1+7*6+5+2+6*6+5+5 = 145 clks
;      PM: 24+20+2+21+2+22+7 = 98          DM: 13
                MOVLW          0x8
                MOVWF          LOOPCOUNT
LOOPSM2424A
                RRF            BARGB2
                BTFSC          _C
                GOTO           ALSM2424NA
                DECFSZ         LOOPCOUNT
                GOTO           LOOPSM2424A
                MOVWF          LOOPCOUNT
LOOPSM2424B
                RRF            BARGB1
                BTFSC          _C
                GOTO           BLSM2424NA
                DECFSZ         LOOPCOUNT
                GOTO           LOOPSM2424B
                MOVLW          0x7
                MOVWF          LOOPCOUNT
LOOPSM2424C
                RRF            BARGB0
                BTFSC          _C
                GOTO           CLSM2424NA
                DECFSZ         LOOPCOUNT
                GOTO           LOOPSM2424C
                CLRF           AARGB0
                CLRF           AARGB1
                CLRF           AARGB2
                RETLW          0x00
ALOOPSM2424
                RRF            BARGB2
                BTFSS          _C
                GOTO           ALSM2424NA
                MOVF           TEMPB2,W
                ADDWF          ACCB2
                MOVF           TEMPB1,W
                BTFSC          _C
                INCFSZ         TEMPB1,W
                ADDWF          ACCB1
                MOVF           TEMPB0,W
                BTFSC          _C

```

AN617

	INCFSZ	TEMPB0 ,W
	ADDWF	ACCB0
ALSM2424NA	RLF	TEMPB0 ,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPSM2424
	MOVLW	0x8
	MOVWF	LOOPCOUNT
BLOOPSM2424	RRF	BARGB1
	BTFSS	_C
	GOTO	BLSM2424NA
	MOVF	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C
	INCFSZ	TEMPB1 ,W
	ADDWF	ACCB1
	MOVF	TEMPB0 ,W
	BTFSC	_C
	INCFSZ	TEMPB0 ,W
	ADDWF	ACCB0
BLSM2424NA	RLF	TEMPB0 ,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	BLOOPSM2424
	MOVLW	0x7
	MOVWF	LOOPCOUNT
CLOOPSM2424	RRF	BARGB0
	BTFSS	_C
	GOTO	CLSM2424NA
	MOVF	TEMPB2 ,W
	ADDWF	ACCB2
	MOVF	TEMPB1 ,W
	BTFSC	_C
	INCFSZ	TEMPB1 ,W
	ADDWF	ACCB1
	MOVF	TEMPB0 ,W
	BTFSC	_C
	INCFSZ	TEMPB0 ,W
	ADDWF	ACCB0
CLSM2424NA	RLF	TEMPB0 ,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	RRF	ACCB5
	DECFSZ	LOOPCOUNT
	GOTO	CLOOPSM2424
	RLF	TEMPB0 ,W
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	RRF	ACCB5
	endm	

```

UMUL2424L      macro
;      Max Timing:      2+14+6*20+19+2+7*21+20+2+7*22+21 = 501 clks
;      Min Timing:      2+7*6+5+1+7*6+5+1+7*6+5+5 = 150 clks
;      PM: 23+20+2+21+2+22 = 88                      DM: 13
                MOVLW      0x08
                MOVWF      LOOPCOUNT

LOOPUM2424A
                RRF         BARGB2
                BTFSC      _C
                GOTO       ALUM2424NAP
                DECFSZ     LOOPCOUNT
                GOTO       LOOPUM2424A
                MOVWF      LOOPCOUNT

LOOPUM2424B
                RRF         BARGB1
                BTFSC      _C
                GOTO       BLUM2424NAP
                DECFSZ     LOOPCOUNT
                GOTO       LOOPUM2424B
                MOVWF      LOOPCOUNT

LOOPUM2424C
                RRF         BARGB0
                BTFSC      _C
                GOTO       CLUM2424NAP
                DECFSZ     LOOPCOUNT
                GOTO       LOOPUM2424C
                CLRF       AARGB0
                CLRF       AARGB1
                CLRF       AARGB2
                RETLW      0x00

CLUM2424NAP
                BCF         _C
                GOTO       CLUM2424NA

BLUM2424NAP
                BCF         _C
                GOTO       BLUM2424NA

ALUM2424NAP
                BCF         _C
                GOTO       ALUM2424NA

ALOOPUM2424
                RRF         BARGB2
                BTFSS      _C
                GOTO       ALUM2424NA
                MOVF       TEMPB2,W
                ADDWF      ACCB2
                MOVF       TEMPB1,W
                BTFSC      _C
                INCFSZ     TEMPB1,W
                ADDWF      ACCB1
                MOVF       TEMPB0,W
                BTFSC      _C
                INCFSZ     TEMPB0,W
                ADDWF      ACCB0

ALUM2424NA
                RRF         ACCB0
                RRF         ACCB1
                RRF         ACCB2
                RRF         ACCB3
                DECFSZ     LOOPCOUNT
                GOTO       ALOOPUM2424
                MOVLW      0x08
                MOVWF      LOOPCOUNT

BLOOPUM2424
                RRF         BARGB1
                BTFSS      _C
                GOTO       BLUM2424NA

```

AN617

```
MOVF          TEMPB2,W
ADDWF        ACCB2
MOVF          TEMPB1,W
BTFSC        _C
INCFSZ       TEMPB1,W
ADDWF        ACCB1
MOVF          TEMPB0,W
BTFSC        _C
INCFSZ       TEMPB0,W
ADDWF        ACCB0

BLUM2424NA
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
DECFSZ       LOOPCOUNT
GOTO         BLOOPUM2424
MOVLW        0x08
MOVWF        LOOPCOUNT

CLOOPUM2424
RRF          BARGB0
BTFSS        _C
GOTO         CLUM2424NA
MOVF          TEMPB2,W
ADDWF        ACCB2
MOVF          TEMPB1,W
BTFSC        _C
INCFSZ       TEMPB1,W
ADDWF        ACCB1
MOVF          TEMPB0,W
BTFSC        _C
INCFSZ       TEMPB0,W
ADDWF        ACCB0

CLUM2424NA
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
RRF          ACCB5
DECFSZ       LOOPCOUNT
GOTO         CLOOPUM2424
endm

UMUL2323L
macro
;      Max Timing:      2+15+6*20+19+2+7*21+20+2+6*22+21+6 = 486 clks
;      Min Timing:      2+7*6+5+1+7*6+5+2+6*6+5+5 = 145 clks
;      PM: 24+20+2+21+2+22+6 = 97          DM: 13
MOVLW        0x8
MOVWF        LOOPCOUNT

LOOPUM2323A
RRF          BARGB2
BTFSC        _C
GOTO         ALUM2323NAP
DECFSZ       LOOPCOUNT
GOTO         LOOPUM2323A
MOVWF        LOOPCOUNT

LOOPUM2323B
RRF          BARGB1
BTFSC        _C
GOTO         BLUM2323NAP
DECFSZ       LOOPCOUNT
GOTO         LOOPUM2323B
MOVLW        0x7
MOVWF        LOOPCOUNT

LOOPUM2323C
```


	RRF	BARGB0
	BTFSC	_C
	GOTO	CLUM2323NAP
	DECFSZ	LOOPCOUNT
	GOTO	LOOPUM2323C
	CLRF	AARGB0
	CLRF	AARGB1
	CLRF	AARGB2
	RETLW	0x00
CLUM2323NAP		
	BCF	_C
	GOTO	CLUM2323NA
BLUM2323NAP		
	BCF	_C
	GOTO	BLUM2323NA
ALUM2323NAP		
	BCF	_C
	GOTO	ALUM2323NA
ALOOPUM2323		
	RRF	BARGB2
	BTFSS	_C
	GOTO	ALUM2323NA
	MOVF	TEMPB2 , W
	ADDWF	ACCB2
	MOVF	TEMPB1 , W
	BTFSC	_C
	INCFSZ	TEMPB1 , W
	ADDWF	ACCB1
	MOVF	TEMPB0 , W
	BTFSC	_C
	INCFSZ	TEMPB0 , W
	ADDWF	ACCB0
ALUM2323NA		
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	DECFSZ	LOOPCOUNT
	GOTO	ALOOPUM2323
	MOVLW	0x08
	MOVWF	LOOPCOUNT
BLOOPUM2323		
	RRF	BARGB1
	BTFSS	_C
	GOTO	BLUM2323NA
	MOVF	TEMPB2 , W
	ADDWF	ACCB2
	MOVF	TEMPB1 , W
	BTFSC	_C
	INCFSZ	TEMPB1 , W
	ADDWF	ACCB1
	MOVF	TEMPB0 , W
	BTFSC	_C
	INCFSZ	TEMPB0 , W
	ADDWF	ACCB0
BLUM2323NA		
	RRF	ACCB0
	RRF	ACCB1
	RRF	ACCB2
	RRF	ACCB3
	RRF	ACCB4
	DECFSZ	LOOPCOUNT
	GOTO	BLOOPUM2323
	MOVLW	0x07
	MOVWF	LOOPCOUNT
CLOOPUM2323		

AN617

```

                RRF          BARGB0
                BTFSS       _C
                GOTO        CLUM2323NA
                MOVF        TEMPB2,W
                ADDWF       ACCB2
                MOVF        TEMPB1,W
                BTFSC       _C
                INCFSZ     TEMPB1,W
                ADDWF       ACCB1
                MOVF        TEMPB0,W
                BTFSC       _C
                INCFSZ     TEMPB0,W
                ADDWF       ACCB0
CLUM2323NA
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                RRF          ACCB4
                RRF          ACCB5
                DECFSZ     LOOPCOUNT
                GOTO        CLOOPUM2323
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                RRF          ACCB4
                RRF          ACCB5
                endm

SMUL2424      macro
;           Max Timing:      8+7*17+8*18+7*19+7 = 411 clks
;           Min Timing:      46+5 = 51 clks
;           PM: 51+4+7*17+8*18+7*19+7 = 466           DM: 12
                variable i
                i = 0

                while i < 8

                BTFSC       BARGB2,i
                GOTO        SM2424NA#v(i)
                i = i + 1
                endw
                i = 8

                while i < 16

                BTFSC       BARGB1,i-8
                GOTO        SM2424NA#v(i)
                i = i + 1
                endw
                i = 16

                while i < 23

                BTFSC       BARGB0,i-16
                GOTO        SM2424NA#v(i)
                i = i + 1
                endw
                CLRF        ACCB0           ; if we get here, BARG = 0
                CLRF        ACCB1
                CLRF        ACCB2
                RETEW       0
SM2424NA0    RLF          TEMPB0,W
                RRF          ACCB0
                RRF          ACCB1
```

```

RRF          ACCB2
RRF          ACCB3
i = 1
while      i < 8
SM2424A#v(i) BTFSS          BARGB2,i
GOTO          SM2424NA#v(i)
MOVF          TEMPB2,W
ADDWF        ACCB2
MOVF          TEMPB1,W
BTFSC        _C
INCFSZ       TEMPB1,W
ADDWF        ACCB1
MOVF          TEMPB0,W
BTFSC        _C
INCFSZ       TEMPB0,W
ADDWF        ACCB0
SM2424NA#v(i) RLF          TEMPB0,W
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
i = i + 1
endw
i = 8
while      i < 16
SM2424A#v(i) BTFSS          BARGB1,i-8
GOTO          SM2424NA#v(i)
MOVF          TEMPB2,W
ADDWF        ACCB2
MOVF          TEMPB1,W
BTFSC        _C
INCFSZ       TEMPB1,W
ADDWF        ACCB1
MOVF          TEMPB0,W
BTFSC        _C
INCFSZ       TEMPB0,W
ADDWF        ACCB0
SM2424NA#v(i) RLF          TEMPB0,W
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
i = i + 1
endw
i = 16
while      i < 23
SM2424A#v(i) BTFSS          BARGB0,i-16
GOTO          SM2424NA#v(i)
MOVF          TEMPB2,W
ADDWF        ACCB2
MOVF          TEMPB1,W
BTFSC        _C
INCFSZ       TEMPB1,W
ADDWF        ACCB1
MOVF          TEMPB0,W
BTFSC        _C
INCFSZ       TEMPB0,W
ADDWF        ACCB0
SM2424NA#v(i) RLF          TEMPB0,W
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
RRF          ACCB5

```

AN617

```
        i = i + 1
    endw
    RLF          TEMPB0,W
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
    RRF          ACCB4
    RRF          ACCB5
    endm
UMUL2424    macro
;    Max Timing:      8+8*17+8*18+8*19 = 440 clks
;    Min Timing:      49+5 = 54 clks
;    PM: 54+4+8*17+8*18+8*19 = 490          DM: 12
    variable i
    i = 0
    BCF          _C          ; clear carry for first right shift

    while i < 8

        BTFSC          BARGB2,i
        GOTO          UM2424NA#v(i)
        i = i + 1
    endw
    i = 8

    while i < 16

        BTFSC          BARGB1,i-8
        GOTO          UM2424NA#v(i)
        i = i + 1
    endw
    i = 16

    while i < 24

        BTFSC          BARGB0,i-16
        GOTO          UM2424NA#v(i)
        i = i + 1
    endw
    CLRF          ACCB0          ; if we get here, BARG = 0
    CLRF          ACCB1
    CLRF          ACCB2
    RETEW          0
UM2424NA0    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
    i = 1
    while i < 8
        BTFSS          BARGB2,i
        GOTO          UM2424NA#v(i)
UM2424A#v(i)  MOVF          TEMPB2,W
    ADDWF          ACCB2
    MOVF          TEMPB1,W
    BTFSC          _C
    INCF          TEMPB1,W
    ADDWF          ACCB1
    MOVF          TEMPB0,W
    BTFSC          _C
    INCF          TEMPB0,W
    ADDWF          ACCB0
UM2424NA#v(i)  RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
```

```

        i = i + 1
    endw
    i = 8
    while i < 16
        BTFSS          BARGB1,i-8
        GOTO          UM2424NA#v(i)
UM2424A#v(i)    MOVF          TEMPB2,W
        ADDWF         ACCB2
        MOVF          TEMPB1,W
        BTFSC         _C
        INCFSZ        TEMPB1,W
        ADDWF         ACCB1
        MOVF          TEMPB0,W
        BTFSC         _C
        INCFSZ        TEMPB0,W
        ADDWF         ACCB0
UM2424NA#v(i)  RRF          ACCB0
        RRF          ACCB1
        RRF          ACCB2
        RRF          ACCB3
        RRF          ACCB4
        i = i + 1
    endw
    i = 16
    while i < 24
        BTFSS          BARGB0,i-16
        GOTO          UM2424NA#v(i)
UM2424A#v(i)    MOVF          TEMPB2,W
        ADDWF         ACCB2
        MOVF          TEMPB1,W
        BTFSC         _C
        INCFSZ        TEMPB1,W
        ADDWF         ACCB1
        MOVF          TEMPB0,W
        BTFSC         _C
        INCFSZ        TEMPB0,W
        ADDWF         ACCB0
UM2424NA#v(i)  RRF          ACCB0
        RRF          ACCB1
        RRF          ACCB2
        RRF          ACCB3
        RRF          ACCB4
        RRF          ACCB5
        i = i + 1
    endw
endm
UMUL2323      macro
;      Max Timing:      8+7*17+8*18+7*19+7 = 411 clks
;      Min Timing:      46+5 = 51 clks
;      PM: 51+4+7*17+8*18+7*19+7 = 466          DM: 12
        variable i
        i = 0
        BCF          _C          ; clear carry for first right shift
        while i < 8

            BTFSC          BARGB2,i
            GOTO          UM2323NA#v(i)
            i = i + 1
        endw
        i = 8
        while i < 16

            BTFSC          BARGB1,i-8
            GOTO          UM2323NA#v(i)
            i = i + 1
        endw

```

AN617

```

        i = 16
        while i < 23

            BTFSC          BARGB0,i-16
            GOTO          UM2323NA#v(i)
            i = i + 1
        endw
        CLRF             ACCB0          ; if we get here, BARG = 0
        CLRF             ACCB1
        CLRF             ACCB2
        CLRF             ACCB3
UM2323NA0    RETEW          0
            RRF             ACCB0
            RRF             ACCB1
            RRF             ACCB2
            RRF             ACCB3
            i = 1
            while i < 8
UM2323A#v(i)    BTFSS          BARGB2,i
                GOTO          UM2323NA#v(i)
                MOVF          TEMPB2,W
                ADDWF          ACCB2
                MOVF          TEMPB1,W
                BTFSC          _C
                INCF          TEMPB1,W
                ADDWF          ACCB1
                MOVF          TEMPB0,W
                BTFSC          _C
                INCF          TEMPB0,W
UM2323NA#v(i)    ADDWF          ACCB0
                RRF             ACCB0
                RRF             ACCB1
                RRF             ACCB2
                RRF             ACCB3
                i = i + 1
            endw
            i = 8
            while i < 16
UM2323A#v(i)    BTFSS          BARGB1,i-8
                GOTO          UM2323NA#v(i)
                MOVF          TEMPB2,W
                ADDWF          ACCB2
                MOVF          TEMPB1,W
                BTFSC          _C
                INCF          TEMPB1,W
                ADDWF          ACCB1
                MOVF          TEMPB0,W
                BTFSC          _C
                INCF          TEMPB0,W
UM2323NA#v(i)    ADDWF          ACCB0
                RRF             ACCB0
                RRF             ACCB1
                RRF             ACCB2
                RRF             ACCB3
                RRF             ACCB4
                i = i + 1
            endw
            i = 16
            while i < 23
UM2323A#v(i)    BTFSS          BARGB0,i-16
                GOTO          UM2323NA#v(i)
                MOVF          TEMPB2,W
                ADDWF          ACCB2
                MOVF          TEMPB1,W
                BTFSC          _C
                INCF          TEMPB1,W
```

```

                ADDWF      ACCB1
                MOVF      TEMPB0,W
                BTFSC     _C
                INCF      TEMPB0,W
UM2323NA#v(i)  ADDWF      ACCB0
                RRF       ACCB0
                RRF       ACCB1
                RRF       ACCB2
                RRF       ACCB3
                RRF       ACCB4
                RRF       ACCB5
                i = i + 1
                endw
                RRF       ACCB0
                RRF       ACCB1
                RRF       ACCB2
                RRF       ACCB3
                RRF       ACCB4
                RRF       ACCB5
                endm

;*****
;*****

;      24x24 Bit Signed Fixed Point Multiply 24x24 -> 48
;      Input:  24 bit signed fixed point multiplicand in AARGB0
;              24 bit signed fixed point multiplier in BARGB0
;      Use:    CALL    FXM2424S
;      Output: 48 bit signed fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing: 12+506+2 = 520 clks          B > 0
;                  27+506+2 = 535 clks          B < 0
;      Min Timing: 12+145 = 157 clks
;      PM: 27+98+1 = 126                      DM: 13
FXM2424S      BTFSS     BARGB0,MSB
                GOTO     M2424SOK
                COMF     BARGB2          ; make multiplier BARG > 0
                INCF     BARGB2
                BTFSC     _Z
                DECF     BARGB1
                COMF     BARGB1
                BTFSC     _Z
                DECF     BARGB0
                COMF     BARGB0
                COMF     AARGB2
                INCF     AARGB2
                BTFSC     _Z
                DECF     AARGB1
                COMF     AARGB1
                BTFSC     _Z
                DECF     AARGB0
                COMF     AARGB0
M2424SOK     CLRF      ACCB3          ; clear partial product
                CLRF     ACCB4
                CLRF     ACCB5
                MOVF     AARGB0,W
                MOVWF    TEMPB0
                MOVF     AARGB1,W
                MOVWF    TEMPB1
                MOVF     AARGB2,W
                MOVWF    TEMPB2
                SMUL2424L
                RETLW    0x00
;*****
;*****

```

AN617

```
;      24x24 Bit Unsigned Fixed Point Multiply 24x24 -> 48
;      Input:  24 bit unsigned fixed point multiplicand in AARGB0
;              24 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL    FXM2424U
;      Output: 48 bit unsigned fixed point product in AARGB0
;      Result: AARG <--  AARG x BARG
;      Max Timing:    9+501+2 = 512 clks
;      Min Timing:    9+150 = 159 clks
;      PM: 9+88+1 = 98          DM: 13
FXM2424U
        CLRF        ACCB3          ; clear partial product
        CLRF        ACCB4
        CLRF        ACCB5
        MOVF        AARGB0,W
        MOVWF       TEMPB0
        MOVF        AARGB1,W
        MOVWF       TEMPB1
        MOVF        AARGB2,W
        MOVWF       TEMPB2
        UMUL2424L
        RETLW       0x00
;*****
;*****
;      23x23 Bit Unsigned Fixed Point Divide 23x23 -> 46
;      Input:  23 bit unsigned fixed point multiplicand in AARGB0
;              23 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL    FXM2323U
;      Output: 46 bit unsigned fixed point product in AARGB0
;      Result: AARG <--  AARG x BARG
;      Max Timing:    9+486+2 = 497 clks
;      Min Timing:    9+145 = 154 clks
;      PM: 9+97+1 = 107        DM: 13
FXM2323U
        CLRF        ACCB3          ; clear partial product
        CLRF        ACCB4
        CLRF        ACCB5
        MOVF        AARGB0,W
        MOVWF       TEMPB0
        MOVF        AARGB1,W
        MOVWF       TEMPB1
        MOVF        AARGB2,W
        MOVWF       TEMPB2
        UMUL2323L
        RETLW       0x00
;*****
;*****
        END
```

C.5 24x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

```
;      24x16 PIC16 FIXED POINT MULTIPLY ROUTINES      VERSION 1.2
;      Input:  fixed point arguments in AARG and BARG
;      Output: product AARGxBARG in AARG
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed multiply application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;      Routine      Clocks      Function
;      FXM2416S     346          24x16 -> 40 bit signed fixed point multiply
;      FXM2416U     334          24x16 -> 40 bit unsigned fixed point multiply
;      FXM2315U     319          23x15 -> 38 bit unsigned fixed point multiply
;      The above timings are based on the looped macros. If space permits,
;      approximately 36-62 clocks can be saved by using the unrolled macros.
;      list    r=dec,x=on,t=off
;      include <PIC16.INC>      ; general PIC16 definitions
```



```

        include <MATH16.INC>      ; PIC16 math library definitions
;*****
;*****
;       Test suite storage
RANDHI      equ      0x1A      ; random number generator registers
RANDLO      equ      0x1B
DATA        equ      0x20      ; beginning of test data
;*****
;*****
;       Test suite for 24x16 bit fixed point multiply algorithms
                                org          0x0005
MAIN         MOVLW        RAMSTART
                                MOVWF       FSR
MEMLOOP      CLRF         INDF
                                INCF        FSR
                                MOVLW       RAMSTOP
                                SUBWF       FSR,W
                                BTFSS       _Z
                                GOTO        MEMLOOP
                                MOVLW       0x45                                ; seed for random numbers
                                MOVWF       RANDLO
                                MOVLW       0x30
                                MOVWF       RANDHI
                                MOVLW       DATA
                                MOVWF       FSR
                                BCF         _RP0
                                BCF         _RP1
                                BCF         _IRP
                                CALL        TFXM2416
SELF         GOTO        SELF
RANDOM16      RLF          RANDHI,W                                ; random number generator
                                XORWF       RANDHI,W
                                MOVWF       TEMPB0
                                SWAPF       RANDHI
                                SWAPF       RANDLO,W
                                MOVWF       TEMPB1
                                RLF         TEMPB1,W
                                RLF         TEMPB1
                                MOVF        TEMPB1,W
                                XORWF       RANDHI,W
                                SWAPF       RANDHI
                                ANDLW      0x01
                                RLF         TEMPB0
                                RLF         RANDLO
                                XORWF       RANDLO
                                RLF         RANDHI

                                RETEW       0
;       Test suite for FXM2416
TFXM2416     CALL        RANDOM16
                                MOVF        RANDHI,W
                                MOVWF       BARGB0
;                                BCF         BARGB0,MSB
;                                MOVF        BARGB0,W
                                MOVWF       INDF
                                INCF        FSR
                                MOVF        RANDLO,W
                                MOVWF       BARGB1
                                MOVWF       INDF
                                INCF        FSR
                                CALL        RANDOM16
                                MOVF        RANDHI,W
                                MOVWF       AARGB0
;                                BCF         AARGB0,MSB
;                                MOVF        AARGB0,W

```

AN617

```
MOVWF    INDF
INCF     FSR
MOVF    RANDLO,W
MOVWF   AARGB1
MOVWF   INDF
INCF    FSR
CALL    RANDOM16
MOVF    RANDHI,W
MOVWF   AARGB2
MOVWF   INDF
INCF    FSR
CALL    FXM2416S
MOVF    AARGB0,W
MOVWF   INDF
INCF    FSR
MOVF    AARGB1,W
MOVWF   INDF
INCF    FSR
MOVF    AARGB2,W
MOVWF   INDF
INCF    FSR
MOVF    AARGB3,W
MOVWF   INDF
INCF    FSR
MOVF    AARGB4,W
MOVWF   INDF
INCF    FSR
RETLW   0x00
;*****
;*****
;      24x16 Bit Multiplication Macros
SMUL2416L    macro
;      Max Timing:      2+12+6*21+20+2+6*22+21+6 = 321 clks
;      Min Timing:      2+7*6+5+2+6*6+5+5 = 97 clks
;      PM: 19+20+2+21+6 = 68          DM: 12
        MOVLW    0x8
        MOVWF   LOOPCOUNT
LOOPSM2416A
        RRF     BARGB1
        BTFSC  _C
        GOTO   ALSM2416NA
        DECFSZ LOOPCOUNT
        GOTO   LOOPSM2416A
        MOVLW  0x7
        MOVWF  LOOPCOUNT
LOOPSM2416B
        RRF     BARGB0
        BTFSC  _C
        GOTO   BLSM2416NA
        DECFSZ LOOPCOUNT
        GOTO   LOOPSM2416B
        CLRF   AARGB0
        CLRF   AARGB1
        CLRF   AARGB2
        RETLW  0x00
ALOOPSM2416
        RRF     BARGB1
        BTFSS  _C
        GOTO   ALSM2416NA
        MOVF   TEMPB2,W
        ADDWF  ACB2
        MOVF   TEMPB1,W
        BTFSC  _C
        INCF   TEMPB1,W
        ADDWF  ACB1
        MOVF   TEMPB0,W
```

```

                BTFSC          _C
                INCFSZ        TEMPB0,W
                ADDWF         ACCB0
ALSM2416NA     RLF           TEMPB0,W
                RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                DECFSZ        LOOPCOUNT
                GOTO          ALOOPSM2416
                MOVLW         0x7
                MOVWF         LOOPCOUNT
BLOOPSM2416
                RRF           BARGB0
                BTFSS         _C
                GOTO          BLSM2416NA
                MOVF          TEMPB2,W
                ADDWF         ACCB2
                MOVF          TEMPB1,W
                BTFSC         _C
                INCFSZ        TEMPB1,W
                ADDWF         ACCB1
                MOVF          TEMPB0,W
                BTFSC         _C
                INCFSZ        TEMPB0,W
                ADDWF         ACCB0
BLSM2416NA     RLF           TEMPB0,W
                RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                RRF           ACCB4
                DECFSZ        LOOPCOUNT
                GOTO          BLOOPSM2416
                RLF           TEMPB0,W
                RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                RRF           ACCB4
                endm
UMUL2416L     macro
;           Max Timing:      2+14+6*20+19+2+7*21+20 = 324 clks
;           Min Timing:      2+7*6+5+1+7*6+5+5 = 102 clks
;           PM: 18+20+2+21 = 61           DM: 12
                MOVLW         0x08
                MOVWF         LOOPCOUNT
LOOPUM2416A
                RRF           BARGB1
                BTFSS         _C
                GOTO          ALUM2416NAP
                DECFSZ        LOOPCOUNT
                GOTO          LOOPUM2416A
                MOVWF         LOOPCOUNT
LOOPUM2416B
                RRF           BARGB0
                BTFSS         _C
                GOTO          BLUM2416NAP
                DECFSZ        LOOPCOUNT
                GOTO          LOOPUM2416B
                CLRF          AARGB0
                CLRF          AARGB1
                CLRF          AARGB2
                RETLW         0x00
BLUM2416NAP
                BCF           _C

```

AN617

```
ALUM2416NAP      GOTO      BLUM2416NA
ALUM2416NAP      BCF        _C
ALUM2416NAP      GOTO      ALUM2416NA
ALOOPUM2416      RRF        BARGB1
ALOOPUM2416      BTFSS     _C
ALOOPUM2416      GOTO      ALUM2416NA
ALOOPUM2416      MOVF      TEMPB2,W
ALOOPUM2416      ADDWF     ACCB2
ALOOPUM2416      MOVF      TEMPB1,W
ALOOPUM2416      BTFSC     _C
ALOOPUM2416      INCFSZ   TEMPB1,W
ALOOPUM2416      ADDWF     ACCB1
ALOOPUM2416      MOVF      TEMPB0,W
ALOOPUM2416      BTFSC     _C
ALOOPUM2416      INCFSZ   TEMPB0,W
ALOOPUM2416      ADDWF     ACCB0
ALUM2416NA      RRF        ACCB0
ALUM2416NA      RRF        ACCB1
ALUM2416NA      RRF        ACCB2
ALUM2416NA      RRF        ACCB3
ALUM2416NA      DECFSZ   LOOPCOUNT
ALUM2416NA      GOTO      ALOOPUM2416
ALUM2416NA      MOVLW    0x08
ALUM2416NA      MOVWF    LOOPCOUNT
BLOOPUM2416      RRF        BARGB0
BLOOPUM2416      BTFSS     _C
BLOOPUM2416      GOTO      BLUM2416NA
BLOOPUM2416      MOVF      TEMPB2,W
BLOOPUM2416      ADDWF     ACCB2
BLOOPUM2416      MOVF      TEMPB1,W
BLOOPUM2416      BTFSC     _C
BLOOPUM2416      INCFSZ   TEMPB1,W
BLOOPUM2416      ADDWF     ACCB1
BLOOPUM2416      MOVF      TEMPB0,W
BLOOPUM2416      BTFSC     _C
BLOOPUM2416      INCFSZ   TEMPB0,W
BLOOPUM2416      ADDWF     ACCB0
BLUM2416NA      RRF        ACCB0
BLUM2416NA      RRF        ACCB1
BLUM2416NA      RRF        ACCB2
BLUM2416NA      RRF        ACCB3
BLUM2416NA      RRF        ACCB4
BLUM2416NA      DECFSZ   LOOPCOUNT
BLUM2416NA      GOTO      BLOOPUM2416
BLUM2416NA      endm
UMUL2315L      macro
;      Max Timing:      2+15+6*20+19+2+6*21+20+5 = 309 clks
;      Min Timing:      2+7*6+5+1+6*6+5+5 = 96 clks
;      PM: 19+20+2+21+5 = 67          DM: 12
UMUL2315L      MOVLW    0x8
UMUL2315L      MOVWF    LOOPCOUNT
LOOPUM2315A      RRF        BARGB1
LOOPUM2315A      BTFSC     _C
LOOPUM2315A      GOTO      ALUM2315NAP
LOOPUM2315A      DECFSZ   LOOPCOUNT
LOOPUM2315A      GOTO      LOOPUM2315A
LOOPUM2315A      MOVLW    0x7
LOOPUM2315A      MOVWF    LOOPCOUNT
LOOPUM2315B      RRF        BARGB0
```

```

                BTFSC          _C
                GOTO          BLUM2315NAP
                DECFSZ        LOOPCOUNT
                GOTO          LOOPUM2315B
                CLRF          AARGB0
                CLRF          AARGB1
                CLRF          AARGB2
                RETLW         0x00
BLUM2315NAP
                BCF           _C
                GOTO          BLUM2315NA
ALUM2315NAP
                BCF           _C
                GOTO          ALUM2315NA
ALOOPUM2315
                RRF           BARGB1
                BTFSS        _C
                GOTO          ALUM2315NA
                MOVF         TEMPB2,W
                ADDWF        ACCB2
                MOVF         TEMPB1,W
                BTFSC        _C
                INCFSZ       TEMPB1,W
                ADDWF        ACCB1
                MOVF         TEMPB0,W
                BTFSC        _C
                INCFSZ       TEMPB0,W
                ADDWF        ACCB0
ALUM2315NA
                RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                DECFSZ       LOOPCOUNT
                GOTO          ALOOPUM2315
                MOVLW        0x07
                MOVWF        LOOPCOUNT
BLOOPUM2315
                RRF           BARGB0
                BTFSS        _C
                GOTO          BLUM2315NA
                MOVF         TEMPB2,W
                ADDWF        ACCB2
                MOVF         TEMPB1,W
                BTFSC        _C
                INCFSZ       TEMPB1,W
                ADDWF        ACCB1
                MOVF         TEMPB0,W
                BTFSC        _C
                INCFSZ       TEMPB0,W
                ADDWF        ACCB0
BLUM2315NA
                RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                RRF           ACCB4
                DECFSZ       LOOPCOUNT
                GOTO          BLOOPUM2315
                RRF           ACCB0
                RRF           ACCB1
                RRF           ACCB2
                RRF           ACCB3
                RRF           ACCB4
                endm

```

AN617

```
SMUL2416      macro
;      Max Timing:      8+7*17+7*18+6 = 259 clks
;      Min Timing:      30+5 = 35 clks
;      PM: 30+4+7*17+7*18+6 = 285          DM: 11
      variable i
      i = 0

      while i < 8

      BTFSC      BARGB1,i
      GOTO      SM2416NA#v(i)
      i = i + 1
      endw
      i = 8

      while i < 15

      BTFSC      BARGB0,i-8
      GOTO      SM2416NA#v(i)
      i = i + 1
      endw
      CLRF      ACCB0          ; if we get here, BARG = 0
      CLRF      ACCB1
      CLRF      ACCB2
      RETEW      0
SM2416NA0     RLF      TEMPB0,W
      RRF      ACCB0
      RRF      ACCB1
      RRF      ACCB2
      RRF      ACCB3
      i = 1
      while i < 8
      BTFSS      BARGB1,i
      GOTO      SM2416NA#v(i)
SM2416A#v(i)  MOVF      TEMPB2,W
      ADDWF     ACCB2
      MOVF      TEMPB1,W
      BTFSC     _C
      INCFSSZ   TEMPB1,W
      ADDWF     ACCB1
      MOVF      TEMPB0,W
      BTFSC     _C
      INCFSSZ   TEMPB0,W
      ADDWF     ACCB0
SM2416NA#v(i) RLF      TEMPB0,W
      RRF      ACCB0
      RRF      ACCB1
      RRF      ACCB2
      RRF      ACCB3
      i = i + 1
      endw
      i = 8
      while i < 15
      BTFSS      BARGB0,i-8
      GOTO      SM2416NA#v(i)
SM2416A#v(i)  MOVF      TEMPB2,W
      ADDWF     ACCB2
      MOVF      TEMPB1,W
      BTFSC     _C
      INCFSSZ   TEMPB1,W
      ADDWF     ACCB1
      MOVF      TEMPB0,W
      BTFSC     _C
      INCFSSZ   TEMPB0,W
      ADDWF     ACCB0
SM2416NA#v(i) RLF      TEMPB0,W
```

```

RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
i = i + 1
endw
RLF          TEMPB0,W
RRF          ACCB0
RRF          ACCB1
RRF          ACCB2
RRF          ACCB3
RRF          ACCB4
endm
UMUL2416    macro
;          Max Timing:      8+8*17+8*18 = 288 clks
;          Min Timing:      33+5 = 38 clks
;          PM: 37+4+8*17+8*18 = 321          DM: 11
            variable i
            i = 0
            BCF          _C          ; clear carry for first right shift

            while i < 8

            BTFSC        BARGB1,i
            GOTO         UM2416NA#v(i)
            i = i + 1
            endw
            i = 8

            while i < 16

            BTFSC        BARGB0,i-8
            GOTO         UM2416NA#v(i)
            i = i + 1
            endw
            CLRF         ACCB0          ; if we get here, BARG = 0
            CLRF         ACCB1
            CLRF         ACCB2
            RETEW        0
UM2416NA0   RRF          ACCB0
            RRF          ACCB1
            RRF          ACCB2
            RRF          ACCB3
            i = 1
            while i < 8
UM2416A#v(i) BTFSS        BARGB1,i
            GOTO         UM2416NA#v(i)
            MOVF         TEMPB2,W
            ADDWF        ACCB2
            MOVF         TEMPB1,W
            BTFSC        _C
            INCF        TEMPB1,W
            ADDWF        ACCB1
            MOVF         TEMPB0,W
            BTFSC        _C
            INCF        TEMPB0,W
            ADDWF        ACCB0
UM2416NA#v(i) RRF          ACCB0
            RRF          ACCB1
            RRF          ACCB2
            RRF          ACCB3
            i = i + 1
            endw
            i = 8
            while i < 16

```

AN617

```

    BTFSS          BARGB0,i-8
    GOTO          UM2416NA#v(i)
UM2416A#v(i)    MOVF          TEMPB2,W
                ADDWF        ACCB2
                MOVF          TEMPB1,W
                BTFSC        _C
                INCFSZ       TEMPB1,W
                ADDWF        ACCB1
                MOVF          TEMPB0,W
                BTFSC        _C
                INCFSZ       TEMPB0,W
                ADDWF        ACCB0
UM2416NA#v(i)  RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                RRF          ACCB4
                i = i + 1
                endw
                endm
UMUL2315      macro
;           Max Timing:      8*7*17+7*18+6 = 259 clks
;           Min Timing:      31+5 = 36 clks
;           PM: 35+4*7*17+7*18+6 = 290           DM: 11
                variable i
                i = 0
                BCF          _C           ; clear carry for first right shift
                while i < 8

                BTFSC        BARGB1,i
                GOTO          UM2315NA#v(i)
                i = i + 1
                endw
                i = 8
                while i < 15

                BTFSC        BARGB0,i-8
                GOTO          UM2315NA#v(i)
                i = i + 1
                endw
                CLRF         ACCB0           ; if we get here, BARG = 0
                CLRF         ACCB1
                CLRF         ACCB2
                RETEW        0
UM2315NA0     RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                i = 1
                while i < 8
                BTFSS        BARGB1,i
                GOTO          UM2315NA#v(i)
UM2315A#v(i)  MOVF          TEMPB2,W
                ADDWF        ACCB2
                MOVF          TEMPB1,W
                BTFSC        _C
                INCFSZ       TEMPB1,W
                ADDWF        ACCB1
                MOVF          TEMPB0,W
                BTFSC        _C
                INCFSZ       TEMPB0,W
                ADDWF        ACCB0
UM2315NA#v(i) RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
```



```

        i = i + 1
    endw
    i = 8
    while i < 15
        BTFSS    BARGB0,i-8
        GOTO     UM2315NA#v(i)
UM2315A#v(i)  MOVF     TEMPB2,W
        ADDWF   ACCB2
        MOVF   TEMPB1,W
        BTFSC  _C
        INCFSZ TEMPB1,W
        ADDWF  ACCB1
        MOVF   TEMPB0,W
        BTFSC  _C
        INCFSZ TEMPB0,W
        ADDWF  ACCB0
UM2315NA#v(i) RRF     ACCB0
        RRF     ACCB1
        RRF     ACCB2
        RRF     ACCB3
        RRF     ACCB4
        i = i + 1
    endw
    RRF     ACCB0
    RRF     ACCB1
    RRF     ACCB2
    RRF     ACCB3
    RRF     ACCB4
    endm

;*****
;*****

;      24x16 Bit Signed Fixed Point Multiply 24x16 -> 40
;      Input:  24 bit signed fixed point multiplicand in AARGB0
;              16 bit signed fixed point multiplier in BARGB0
;      Use:    CALL    FXM2416S
;      Output: 40 bit signed fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:  11+321+2 = 334 clks           B > 0
;                  23+321+2 = 346 clks           B < 0
;      Min Timing:  11+97 = 108 clks
;      PM: 23+68+1 = 92           DM: 12
FXM2416S    BTFSS    BARGB0,MSB
            GOTO     M2416SOK
            COMF    BARGB1           ; make multiplier BARG > 0
            INCF    BARGB1
            BTFSC  _Z
            DECF    BARGB0
            COMF    BARGB0
            COMF    AARGB2
            INCF    AARGB2
            BTFSC  _Z
            DECF    AARGB1
            COMF    AARGB1
            BTFSC  _Z
            DECF    AARGB0
            COMF    AARGB0
M2416SOK    CLRF    ACCB3           ; clear partial product
            CLRF    ACCB4
            MOVF   AARGB0,W
            MOVWF  TEMPB0
            MOVF   AARGB1,W
            MOVWF  TEMPB1
            MOVF   AARGB2,W
            MOVWF  TEMPB2

```

AN617

```
SMUL2416L
RETLW      0x00
;*****
;*****
;
; 24x16 Bit Unsigned Fixed Point Multiply 24x16 -> 40
; Input: 24 bit unsigned fixed point multiplicand in AARGB0
;       16 bit unsigned fixed point multiplier in BARGB0
; Use:   CALL   FXM2416U
; Output: 40 bit unsigned fixed point product in AARGB0
; Result: AARG <-- AARG x BARG
; Max Timing: 8+324+2 = 334 clks
; Min Timing: 8+102 = 110 clks
; PM: 8+61+1 = 70          DM: 12
FXM2416U
    CLRF      ACCB3          ; clear partial product
    CLRF      ACCB4
    MOVF      AARGB0,W
    MOVWF     TEMPB0
    MOVF      AARGB1,W
    MOVWF     TEMPB1
    MOVF      AARGB2,W
    MOVWF     TEMPB2
    UMUL2416L
    RETLW     0x00
;*****
;*****
;
; 23x15 Bit Unsigned Fixed Point Divide 23x15 -> 38
; Input: 23 bit unsigned fixed point multiplicand in AARGB0
;       15 bit unsigned fixed point multiplier in BARGB0
; Use:   CALL   FXM2315U
; Output: 38 bit unsigned fixed point product in AARGB0
; Result: AARG <-- AARG x BARG
; Max Timing: 8+309+2 = 319 clks
; Min Timing: 8+96 = 104 clks
; PM: 8+67+1 = 76          DM: 12
FXM2315U
    CLRF      ACCB3          ; clear partial product
    CLRF      ACCB4
    MOVF      AARGB0,W
    MOVWF     TEMPB0
    MOVF      AARGB1,W
    MOVWF     TEMPB1
    MOVF      AARGB2,W
    MOVWF     TEMPB2
    UMUL2315L
    RETLW     0x00
;*****
;*****
END
```

C.6 16x16 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

```
; 16x16 PIC16 FIXED POINT MULTIPLY ROUTINES          VERSION 1.2
; Input: fixed point arguments in AARG and BARG
; Output: product AARGxBARG in AARG
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed multiply application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in
; performance.
; Routine      Clocks      Function
; FXM1616S     269         16x16 -> 32 bit signed fixed point multiply
; FXM1616U     256         16x16 -> 32 bit unsigned fixed point multiply
; FXM1515U     244         15x15 -> 30 bit unsigned fixed point multiply
; The above timings are based on the looped macros. If space permits,
```

```

;      approximately 64-73 clocks can be saved by using the unrolled macros.
;      list      r=dec,x=on,t=off
;      include <PIC16.INC>      ; general PIC16 definitions
;      include <MATH16.INC>     ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI      equ      0x1A      ; random number generator registers
RANDLO      equ      0x1B
TESTCOUNT  equ      0x20      ; counter
DATA        equ      0x21      ; beginning of test data
;*****
;*****
;      Test suite for 16x16 bit fixed point multiply algorithms
;      org      0x0005
MAIN        MOVLW      RAMSTART
;      MOVWF      FSR
MEMLOOP     CLRF       INDF
;      INCF       FSR
;      MOVLW      RAMSTOP
;      SUBWF      FSR,W
;      BTFSS      _Z
;      GOTO       MEMLOOP
;      MOVLW      0x45          ; seed for random numbers
;      MOVWF      RANDLO
;      MOVLW      0x30
;      MOVWF      RANDHI
;      MOVLW      DATA
;      MOVWF      FSR
;      BCF       _RP0
;      BCF       _RP1
;      BCF       _IRP
;      CALL      TFXM1616
SELF        GOTO       SELF
RANDOM16     RLF       RANDHI,W      ; random number generator
;      XORWF      RANDHI,W
;      MOVWF      TEMPB0
;      SWAPF      RANDHI
;      SWAPF      RANDLO,W
;      MOVWF      TEMPB1
;      RLF       TEMPB1,W
;      RLF       TEMPB1
;      MOVF      TEMPB1,W
;      XORWF      RANDHI,W
;      SWAPF      RANDHI
;      ANDLW     0x01
;      RLF       TEMPB0
;      RLF       RANDLO
;      XORWF      RANDLO
;      RLF       RANDHI
;      RETEW      0
;      Test suite for FXM1616
TFXM1616    MOVLW      1
;      MOVWF      TESTCOUNT
M1616LOOP   CALL      RANDOM16
;      MOVF      RANDHI,W
;      MOVWF      BARGB0
;      BCF       BARGB0,MSB
;      MOVF      BARGB0,W
;      MOVWF      INDF
;      INCF      FSR
;      MOVF      RANDLO,W
;      MOVWF      BARGB1
;      MOVWF      INDF

```

AN617

```

    INCF          FSR
    CALL         RANDOM16
    MOVF         RANDHI ,W
    MOVWF        AARGB0
    BCF         AARGB0 ,MSB
    MOVF         AARGB0 ,W
    MOVWF        INDF
    INCF          FSR
    MOVF         RANDLO ,W
    MOVWF        AARGB1
    MOVWF        INDF
    INCF          FSR
    CALL         FXM1515U
    MOVF         AARGB0 ,W
    MOVWF        INDF
    INCF          FSR
    MOVF         AARGB1 ,W
    MOVWF        INDF
    INCF          FSR
    MOVF         AARGB2 ,W
    MOVWF        INDF
    INCF          FSR
    MOVF         AARGB3 ,W
    MOVWF        INDF
    INCF          FSR
    DECFSZ       TESTCOUNT
    GOTO         M1616LOOP
    RETLW        0x00
;*****
;*****
;    16x16 Bit Multiplication Macros
SMUL1616L    macro
;    Max Timing:      2+11+6*16+15+2+6*17+16+5 = 249 clks
;    Min Timing:      2+7*6+5+2+6*6+5+4 = 96 clks
;    PM: 55          DM: 9
    MOVLW        0x8
    MOVWF        LOOPCOUNT
LOOPSM1616A
    RRF          BARGB1
    BTFSC        _C
    GOTO         ALSM1616NA
    DECFSZ       LOOPCOUNT
    GOTO         LOOPSM1616A
    MOVLW        0x7
    MOVWF        LOOPCOUNT
LOOPSM1616B
    RRF          BARGB0
    BTFSC        _C
    GOTO         BLSM1616NA
    DECFSZ       LOOPCOUNT
    GOTO         LOOPSM1616B
    CLRF         AARGB0
    CLRF         AARGB1
    RETLW        0x00
ALOOPSM1616
    RRF          BARGB1
    BTFSS        _C
    GOTO         ALSM1616NA
    MOVF         TEMPB1 ,W
    ADDWF        ACCB1
    MOVF         TEMPB0 ,W
    BTFSC        _C
    INCF        TEMPB0 ,W
    ADDWF        ACCB0
ALSM1616NA
    RLF          TEMPB0 ,W
    RRF          ACCB0

```

```

                RRF          ACCB1
                RRF          ACCB2
                DECFSZ       LOOPCOUNT
                GOTO         ALOOPSM1616
                MOVLW        0x7
                MOVWF        LOOPCOUNT
BLOOPSM1616
                RRF          BARGB0
                BTFSS        _C
                GOTO         BLSM1616NA
                MOVF         TEMPB1,W
                ADDWF        ACCB1
                MOVF         TEMPB0,W
                BTFSC        _C
                INCF        TEMPB0,W
                ADDWF        ACCB0
BLSM1616NA
                RLF          TEMPB0,W
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                DECFSZ       LOOPCOUNT
                GOTO         BLOOPSM1616
                RLF          TEMPB0,W
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                endm
UMUL1616L      macro
;      Max Timing:      2+13+6*15+14+2+7*16+15 = 248 clks
;      Min Timing:      2+7*6+5+1+7*6+5+4 = 101 clks
;      PM: 51          DM: 9
                MOVLW        0x08
                MOVWF        LOOPCOUNT
LOOPUM1616A
                RRF          BARGB1
                BTFSC        _C
                GOTO         ALUM1616NAP
                DECFSZ       LOOPCOUNT
                GOTO         LOOPUM1616A
                MOVWF        LOOPCOUNT
LOOPUM1616B
                RRF          BARGB0
                BTFSC        _C
                GOTO         BLUM1616NAP
                DECFSZ       LOOPCOUNT
                GOTO         LOOPUM1616B
                CLRF         AARGB0
                CLRF         AARGB1
                RETLW        0x00
BLUM1616NAP
                BCF          _C
                GOTO         BLUM1616NA
ALUM1616NAP
                BCF          _C
                GOTO         ALUM1616NA
ALOOPUM1616
                RRF          BARGB1
                BTFSS        _C
                GOTO         ALUM1616NA
                MOVF         TEMPB1,W
                ADDWF        ACCB1
                MOVF         TEMPB0,W
                BTFSC        _C
                INCF        TEMPB0,W

```

AN617

```
ALUM1616NA      ADDWF      ACCB0
                RRF      ACCB0
                RRF      ACCB1
                RRF      ACCB2
                DECFSZ   LOOPCOUNT
                GOTO     ALOOPUM1616
                MOVLW    0x08
                MOVWF    LOOPCOUNT

BLOOPUM1616     RRF      BARGB0
                BTFSS   _C
                GOTO     BLUM1616NA
                MOVF    TEMPB1,W
                ADDWF   ACCB1
                MOVF    TEMPB0,W
                BTFSC   _C
                INCF    TEMPB0,W
                ADDWF   ACCB0

BLUM1616NA     RRF      ACCB0
                RRF      ACCB1
                RRF      ACCB2
                RRF      ACCB3
                DECFSZ   LOOPCOUNT
                GOTO     BLOOPUM1616
                endm

UMUL1515L      macro
;      Max Timing:      2+13+6*15+14+2+6*16+15+4 = 236 clks
;      Min Timing:      2+7*6+5+2+6*6+5+4 = 97 clks
;      PM: 56           DM: 9
                MOVLW    0x8
                MOVWF    LOOPCOUNT

LOOPUM1515A     RRF      BARGB1
                BTFSC   _C
                GOTO     ALUM1515NAP
                DECFSZ   LOOPCOUNT
                GOTO     LOOPUM1515A
                MOVLW    0x7
                MOVWF    LOOPCOUNT

LOOPUM1515B     RRF      BARGB0
                BTFSC   _C
                GOTO     BLUM1515NAP
                DECFSZ   LOOPCOUNT
                GOTO     LOOPUM1515B
                CLRF    AARGB0
                CLRF    AARGB1
                RETLW    0x00

BLUM1515NAP     BCF      _C
                GOTO     BLUM1515NA

ALUM1515NAP     BCF      _C
                GOTO     ALUM1515NA

ALOOPUM1515     RRF      BARGB1
                BTFSS   _C
                GOTO     ALUM1515NA
                MOVF    TEMPB1,W
                ADDWF   ACCB1
                MOVF    TEMPB0,W
                BTFSC   _C
                INCF    TEMPB0,W
                ADDWF   ACCB0
```

```

ALUM1515NA
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    DECFSZ      LOOPCOUNT
    GOTO        ALOOPUM1515
    MOVLW      0x07
    MOVWF      LOOPCOUNT

BLOOPUM1515
    RRF          BARGB0
    BTFSS      _C
    GOTO        BLUM1515NA
    MOVF       TEMPB1,W
    ADDWF      ACCB1
    MOVF       TEMPB0,W
    BTFSC      _C
    INCFSZ     TEMPB0,W
    ADDWF      ACCB0

BLUM1515NA
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
    DECFSZ     LOOPCOUNT
    GOTO        BLOOPUM1515

    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    RRF          ACCB3
    endm

SMUL1616      macro
;      Max Timing:      5+6+7*11+7*12+4 = 176 clks
;      Min Timing:      5+24+21+5 = 55 clks
;      PM: 5+3*8+3*7+6+7*11+7*12+4 = 221          DM: 8
    variable i
    i = 0
    BTFSC      AARGB0,MSB
    COMF       ACCB2
    MOVF       ACCB2,W
    MOVWF     ACCB3
    RLF        ACCB0,W

    while i < 8

    BTFSC      BARGB1,i
    GOTO        SM1616NA#v(i)
    BCF        ACCB2,7-i
    i = i + 1
    endw
    i = 8

    while i < 15

    BTFSC      BARGB0,i-8
    GOTO        SM1616NA#v(i)
    BCF        ACCB2,15-i
    i = i + 1
    endw
    CLRF      ACCB0          ; if we get here, BARG = 0
    CLRF      ACCB1
    CLRF      ACCB3
    RETEW      0

SM1616NA0
    RRF          ACCB0

```

AN617

```

        RRF          ACCB1
        RRF          ACCB2
        i = 1
        while i < 8
        BTFSS        BARGB1,i
SM1616A#v(i)      GOTO          SM1616NA#v(i)
        MOVF        TEMPB1,W
        ADDWF       ACCB1
        MOVF        TEMPB0,W
        BTFSC       _C
        INCFSZ     TEMPB0,W
        ADDWF       ACCB0
SM1616NA#v(i)
        RRF          ACCB0
        RRF          ACCB1
        RRF          ACCB2
        i = i + 1
        endw
        i = 8
        while i < 15
        BTFSS        BARGB0,i-8
SM1616A#v(i)      GOTO          SM1616NA#v(i)
        MOVF        TEMPB1,W
        ADDWF       ACCB1
        MOVF        TEMPB0,W
        BTFSC       _C
        INCFSZ     TEMPB0,W
        ADDWF       ACCB0
SM1616NA#v(i)
        RRF          ACCB0
        RRF          ACCB1
        RRF          ACCB2
        RRF          ACCB3
        i = i + 1
        endw
        RRF          ACCB0
        RRF          ACCB1
        RRF          ACCB2
        RRF          ACCB3
        endm
UMUL1616 macro
;      Max Timing:    1+6+7*11+8*12 = 180 clks
;      Min Timing:    1+2*8+2*8+4 = 37 clks
;      PM: 1+2*8+2*8+4+7*11+8*12 = 210          DM: 8
        variable i
        i = 0
        BCF          _C          ; clear carry for first right shift

        while i < 8

        BTFSC       BARGB1,i
        GOTO        UM1616NA#v(i)
        i = i + 1
        endw
        i = 8

        while i < 16

        BTFSC       BARGB0,i-8
        GOTO        UM1616NA#v(i)
        i = i + 1
        endw
        CLRF        ACCB0          ; if we get here, BARG = 0
        CLRF        ACCB1
        RETEW       0
UM1616NA0      RRF          ACCB0
```



```

                RRF          ACCB1
                RRF          ACCB2
                i = 1
                while i < 8
                BTFSS        BARGB1,i
                GOTO          UM1616NA#v(i)
UM1616A#v(i)   MOVF          TEMPB1,W
                ADDWF        ACCB1
                MOVF          TEMPB0,W
                BTFSC        _C
                INCFSZ       TEMPB0,W
                ADDWF        ACCB0
UM1616NA#v(i) RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                i = i + 1
                endw
                i = 8
                while i < 16
                BTFSS        BARGB0,i-8
                GOTO          UM1616NA#v(i)
UM1616A#v(i)   MOVF          TEMPB1,W
                ADDWF        ACCB1
                MOVF          TEMPB0,W
                BTFSC        _C
                INCFSZ       TEMPB0,W
                ADDWF        ACCB0
UM1616NA#v(i) RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                RRF          ACCB3
                i = i + 1
                endw
                endm
UMUL1515      macro
;           Max Timing:      7+7*11+7*12+4 = 172 clks
;           Min Timing:      1+16+14+4 = 35 clks
;           PM: 1+2*8+2*7+6+7*11+7*12+4 = 202           DM: 8
                variable i
                i = 0
                BCF          _C           ; clear carry for first right shift
                while i < 8

                BTFSC        BARGB1,i
                GOTO          UM1515NA#v(i)
                i = i + 1
                endw
                i = 8
                while i < 15

                BTFSC        BARGB0,i-8
                GOTO          UM1515NA#v(i)
                i = i + 1
                endw
                CLRF          ACCB0           ; if we get here, BARG = 0
                CLRF          ACCB1
                RETEW         0
UM1515NA0     RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                i = 1
                while i < 8
                BTFSS        BARGB1,i
                GOTO          UM1515NA#v(i)
UM1515A#v(i)   MOVF          TEMPB1,W
                ADDWF        ACCB1

```

AN617

```

        MOVF          TEMPB0,W
        BTFSC        _C
        INCFSZ       TEMPB0,W
        ADDWF        ACCB0
UM1515NA#v(i)  RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               i = i + 1
        endw
               i = 8
        while      i < 15
        BTFSS       BARGB0,i-8
        GOTO        UM1515NA#v(i)
UM1515A#v(i)  MOVF          TEMPB1,W
               ADDWF        ACCB1
               MOVF          TEMPB0,W
               BTFSC        _C
               INCFSZ       TEMPB0,W
               ADDWF        ACCB0
UM1515NA#v(i)  RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               RRF          ACCB3
               i = i + 1
        endw
               RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               RRF          ACCB3
        endm

;*****
;*****

;      16x16 Bit Signed Fixed Point Multiply 16x16 -> 32
;      Input:  16 bit signed fixed point multiplicand in AARGB0
;              16 bit signed fixed point multiplier in BARGB0
;      Use:    CALL    FXM1616S
;      Output: 32 bit signed fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:  9+249+2 = 260 clks          B > 0
;                  18+249+2 = 269 clks          B < 0
;      Min Timing:  9+96 = 105 clks
;      PM: 18+55+1 = 74          DM: 9
FXM1616S      BTFSS       BARGB0,MSB
               GOTO        M1616SOK
               COMF        BARGB1          ; make multiplier BARG > 0
               INCF        BARGB1
               BTFSC        _Z
               DECF        BARGB0
               COMF        BARGB0
               COMF        AARGB1
               INCF        AARGB1
               BTFSC        _Z
               DECF        AARGB0
               COMF        AARGB0
M1616SOK     CLRF        ACCB2          ; clear partial product
               CLRF        ACCB3
               MOVF        AARGB0,W
               MOVWF       TEMPB0
               MOVF        AARGB1,W
               MOVWF       TEMPB1
               SMUL1616L
               RETLW       0x00
;*****
;*****
```

```

;      16x16 Bit Unsigned Fixed Point Multiply 16x16 -> 32
;      Input:  16 bit unsigned fixed point multiplicand in AARGB0
;              16 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL    FXM1616U
;      Output: 32 bit unsigned fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:    6+248+2 = 256 clks
;      Min Timing:    6+101 = 107 clks
;      PM: 6+51+1 = 58          DM: 9
FXM1616U
        CLRF        ACCB2          ; clear partial product
        CLRF        ACCB3
        MOVF        AARGB0,W
        MOVWF       TEMPB0
        MOVF        AARGB1,W
        MOVWF       TEMPB1
        UMUL1616L
        RETLW       0x00
;*****
;*****
;      15x15 Bit Unsigned Fixed Point Divide 15x15 -> 30
;      Input:  15 bit unsigned fixed point multiplicand in AARGB0
;              15 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL    FXM1515U
;      Output: 30 bit unsigned fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:    6+236+2 = 244 clks
;      Min Timing:    6+97 = 103 clks
;      PM: 6+56+1 = 63          DM: 9
FXM1515U
        CLRF        ACCB2          ; clear partial product
        CLRF        ACCB3
        MOVF        AARGB0,W
        MOVWF       TEMPB0
        MOVF        AARGB1,W
        MOVWF       TEMPB1
        UMUL1515L
        RETLW       0x00
;*****
;*****
        END

```

C.7 16x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

```

;      16x8 PIC16 FIXED POINT MULTIPLY ROUTINES          VERSION 1.2
;      Input:  fixed point arguments in AARG and BARG
;      Output: product AARGxBARG in AARG
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed multiply application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;
;      Routine      Clocks      Function
;      FXM1608S    128          16x08 -> 24 bit signed fixed point multiply
;      FXM1608U    126          16x08 -> 24 bit unsigned fixed point multiply
;      FXM1507U    114          15x07 -> 22 bit unsigned fixed point multiply
;      The above timings are based on the looped macros. If space permits,
;      approximately 24-35 clocks can be saved by using the unrolled macros.
;      list    r=dec,x=on,t=off
;      include <PIC16.INC>          ; general PIC16 definitions
;      include <MATH16.INC>        ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI    equ    0x1A          ; random number generator registers

```

AN617

```
RANDLO      equ      0x1B
TESTCOUNT  equ      0x20      ; counter
DATA        equ      0x21      ; beginning of test data
;*****
;*****
;          Test suite for 16x8 bit fixed point multiply algorithms
          org          0x0005
MAIN       MOV LW     RAMSTART
          MOV WF     FSR
MEMLOOP    CLR F     INDF
          INC F     FSR
          MOV LW     RAMSTOP
          SUB WF     FSR,W
          BTF SS     _Z
          GOTO      MEMLOOP
          MOV LW     0x45          ; seed for random numbers
          MOV WF     RANDLO
          MOV LW     0x30
          MOV WF     RANDHI
          MOV LW     DATA
          MOV WF     FSR
          BCF       _RP0
          BCF       _RP1
          BCF       _IRP
          CALL      TFXM1608
SELF       GOTO      SELF
RANDOM16    RLF       RANDHI,W          ; random number generator
          XOR WF     RANDHI,W
          MOV WF     TEMPB0
          SWAP F     RANDHI
          SWAP F     RANDLO,W
          MOV WF     TEMPB1
          RLF       TEMPB1,W
          RLF       TEMPB1
          MOV F     TEMPB1,W
          XOR WF     RANDHI,W
          SWAP F     RANDHI
          AND LW     0x01
          RLF       TEMPB0
          RLF       RANDLO
          XOR WF     RANDLO
          RLF       RANDHI

          RETEW      0
;          Test suite for FXM1608
TFXM1608   MOV LW     2
          MOV WF     TESTCOUNT
M1608LOOP  CALL      RANDOM16
          MOV F     RANDHI,W
          MOV WF     BARGB0
          BCF       BARGB0,MSB
          MOV F     BARGB0,W
          MOV WF     INDF
          INC F     FSR
          CALL      RANDOM16
          MOV F     RANDHI,W
          MOV WF     AARGB0
          BCF       AARGB0,MSB
          MOV F     AARGB0,W
          MOV WF     INDF
          INC F     FSR
          MOV F     RANDLO,W
          MOV WF     AARGB1
          MOV WF     INDF
          INC F     FSR
```

```

CALL          FXM1507U
MOVWF        AARGB0,W
MOVWF        INDF
INCF         FSR
MOVWF        AARGB1,W
MOVWF        INDF
INCF         FSR
MOVWF        AARGB2,W
MOVWF        INDF
INCF         FSR
DECFSZ       TESTCOUNT
GOTO         M1608LOOP
RETLW        0x00
;*****
;*****
;      16x08 Bit Multiplication Macros
SMUL1608L    macro
;      Max Timing:      2+11+5*16+15+4 = 112 clks
;      Min Timing:      2+6*6+5+4 = 47 clks
;      PM: 29          DM: 7
                MOVLW        0x07
                MOVWF        LOOPCOUNT
LOOPSM1608A
                RRF           BARGB0
                BTFSC        _C
                GOTO         LSM1608NA
                DECFSZ       LOOPCOUNT
                GOTO         LOOPSM1608A
                CLRF         AARGB0
                CLRF         AARGB1
                RETLW        0x00
LOOPSM1608
                RRF           BARGB0
                BTFSS        _C
                GOTO         LSM1608NA
                MOVF         TEMPB1,W
                ADDWF        ACCB1
                MOVF         TEMPB0,W
                BTFSC        _C
                INCFSZ       TEMPB0,W
                ADDWF        ACCB0
LSM1608NA
                RLF          TEMPB0,W
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                DECFSZ       LOOPCOUNT
                GOTO         LOOPSM1608
                RLF          TEMPB0,W
                RRF          ACCB0
                RRF          ACCB1
                RRF          ACCB2
                endm
UMUL1608L    macro
;      Max Timing:      2+13+6*15+14 = 119 clks
;      Min Timing:      2+7*6+5+4 = 54 clks
;      PM: 26          DM: 7
                MOVLW        0x08
                MOVWF        LOOPCOUNT
LOOPUM1608A
                RRF           BARGB0
                BTFSC        _C
                GOTO         LUM1608NAP
                DECFSZ       LOOPCOUNT
                GOTO         LOOPUM1608A
                CLRF         AARGB0
                CLRF         AARGB1

```

AN617

```
LUM1608NAP      RETLW      0x00
LUM1608NAP      BCF          _C
LUM1608NAP      GOTO        LUM1608NA
LOOPUM1608
    RRF          BARGB0
    BTFSS        _C
    GOTO        LUM1608NA
    MOVF         TEMPB1,W
    ADDWF        ACCB1
    MOVF         TEMPB0,W
    BTFSC        _C
    INCF        TEMPB0,W
    ADDWF        ACCB0
LUM1608NA
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    DECFSZ       LOOPCOUNT
    GOTO        LOOPUM1608
    endm
UMUL1507L      macro
;      Max Timing:      2+13+5*15+14+3 = 107 clks
;      Min Timing:      2+6*6+5+4 = 47 clks
;      PM: 29          DM: 7
    MOVLW       0x07
    MOVWF       LOOPCOUNT
LOOPUM1507A
    RRF          BARGB0
    BTFSC        _C
    GOTO        LUM1507NAP
    DECFSZ       LOOPCOUNT
    GOTO        LOOPUM1507A
    CLRF        AARGB0
    CLRF        AARGB1
    RETLW       0x00
LUM1507NAP
    BCF          _C
    GOTO        LUM1507NA
LOOPUM1507
    RRF          BARGB0
    BTFSS        _C
    GOTO        LUM1507NA
    MOVF         TEMPB1,W
    ADDWF        ACCB1
    MOVF         TEMPB0,W
    BTFSC        _C
    INCF        TEMPB0,W
    ADDWF        ACCB0
LUM1507NA
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    DECFSZ       LOOPCOUNT
    GOTO        LOOPUM1507
    RRF          ACCB0
    RRF          ACCB1
    RRF          ACCB2
    endm
SMUL1608      macro
;      Max Timing:      3+6+6*11+3 = 78 clks
;      Min Timing:      3+21+5 = 29 clks
;      PM: 3+3*7+7+6*11+3 = 100          DM: 6
    variable i
    i = 0
    BTFSC        AARGB0,MSB
    COMF         ACCB2
```

```

        RLF                ACCB0,W

        while i < 7

            BTFSC          BARGB0,i
            GOTO           SM1608NA#v(i)
            BCF            ACCB2,7-i
            i = i + 1
        endw
        CLRF              ACCB0                ; if we get here, BARG = 0
        CLRF              ACCB1
        CLRF              ACCB2
        RETEW            0

SM1608NA0
        RRF              ACCB0
        RRF              ACCB1
        RRF              ACCB2
        i = 1
        while i < 7
            BTFSS          BARGB0,i
            GOTO           SM1608NA#v(i)
SM1608A#v(i)
            MOVF           TEMPB1,W
            ADDWF          ACCB1
            MOVF           TEMPB0,W
            BTFSC          _C
            INCF          TEMPB0,W
            ADDWF          ACCB0

SM1608NA#v(i)
            RRF              ACCB0
            RRF              ACCB1
            RRF              ACCB2
            i = i + 1
        endw
        RRF              ACCB0
        RRF              ACCB1
        RRF              ACCB2
        endm

UMUL1608
macro
;      Max Timing:      1+6+7*11 = 84 clks
;      Min Timing:      1+2*8+4 = 21 clks
;      PM: 1+2*8+4+6*7 = 63          DM: 4
        variable i
        i = 0
        BCF              _C                ; clear carry for first right shift

        while i < 8

            BTFSC          BARGB0,i
            GOTO           UM1608NA#v(i)
            i = i + 1
        endw
        CLRF              ACCB0                ; if we get here, BARG = 0
        CLRF              ACCB1
        RETEW            0

UM1608NA0
        RRF              ACCB0
        RRF              ACCB1
        RRF              ACCB2
        i = 1
        while i < 8
            BTFSS          BARGB0,i
            GOTO           UM1608NA#v(i)
UM1608A#v(i)
            MOVF           TEMPB1,W
            ADDWF          ACCB1
            MOVF           TEMPB0,W
            BTFSC          _C
            INCF          TEMPB0,W
    
```

AN617

```

        ADDWF          ACCB0
UM1608NA#v(i)  RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               i = i + 1
               endw
               endm
UMUL1507
macro
;           Max Timing:      7+6*12+4 = 83 clks
;           Min Timing:     14+3 = 17 clks
;           PM: 2*7+7+6*12+4 = 97           DM: 6
               variable i
               i = 0
               BCF          _C              ; clear carry for first right shift
               while i < 7

               BTFSC          BARGB0,i
               GOTO          UM1507NA#v(i)
               i = i + 1
               endw
               CLRF          ACCB0          ; if we get here, BARG = 0
               CLRF          ACCB1
               RETEW          0
UM1507NA0     RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               i = 1
               while i < 7
UM1507A#v(i)  BTFSS          BARGB0,i
               GOTO          UM1507NA#v(i)
               MOVF          TEMPB1,W
               ADDWF          ACCB1
               MOVF          TEMPB0,W
               BTFSC          _C
               INCF          TEMPB0,W
               ADDWF          ACCB0
UM1507NA#v(i) RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               i = i + 1
               endw
               RRF          ACCB0
               RRF          ACCB1
               RRF          ACCB2
               endm

;*****
;*****

;           16x8 Bit Signed Fixed Point Multiply 16x8 -> 24
;           Input:  16 bit signed fixed point multiplicand in AARGB0
;                   8 bit signed fixed point multiplier in BARGB0
;           Use:    CALL    FXM1608S
;           Output: 24 bit signed fixed point product in AARGB0
;           Result: AARG <-- AARG x BARG
;           Max Timing:      8+112+2 = 122 clks           B > 0
;                   14+112+2 = 128 clks           B < 0
;           Min Timing:     8+47 = 55 clks
;           PM: 14+29+1 = 44           DM: 7
FXM1608S     BTFSS          BARGB0,MSB
               GOTO          M1608SOK
               COMF          BARGB0          ; make multiplier BARG > 0
               INCF          BARGB0
               COMF          AARGB1
               INCF          AARGB1
               BTFSC          _Z
```



```

                DECF          AARGB0
                COMF          AARGB0
M1608SOK       CLRF          ACCB2          ; clear partial product
                MOVF          AARGB0,W
                MOVWF         TEMPB0
                MOVF          AARGB1,W
                MOVWF         TEMPB1
                SMUL1608L
                RETLW         0x00
;*****
;*****
;
;      16x8 Bit Unsigned Fixed Point Multiply 16x8 -> 24
;      Input: 16 bit unsigned fixed point multiplicand in AARGB0
;              8 bit unsigned fixed point multiplier in BARGB0
;      Use:   CALL    FXM1608U
;      Output: 24 bit unsigned fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing: 5+119+2 = 126 clks
;      Min Timing: 5+54 = 59 clks
;      PM: 5+26+1 = 31          DM: 7
FXM1608U       CLRF          ACCB2          ; clear partial product
                MOVF          AARGB0,W
                MOVWF         TEMPB0
                MOVF          AARGB1,W
                MOVWF         TEMPB1
                UMUL1608L
                RETLW         0x00
;*****
;*****
;
;      15x7 Bit Unsigned Fixed Point Divide 15x7 -> 22
;      Input: 15 bit unsigned fixed point multiplicand in AARGB0
;              7 bit unsigned fixed point multiplier in BARGB0
;      Use:   CALL    FXM1507U
;      Output: 22 bit unsigned fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing: 5+107+2 = 114 clks
;      Min Timing: 5+47 = 52 clks
;      PM: 5+29+1 = 35          DM: 7
FXM1507U       CLRF          ACCB2          ; clear partial product
                MOVF          AARGB0,W
                MOVWF         TEMPB0
                MOVF          AARGB1,W
                MOVWF         TEMPB1
                UMUL1507
                RETLW         0x00
;*****
;*****
                END

```

C.8 8x8 PIC16C5X/PIC16CXX Fixed Point Multiply Routines

```

;      8x8 PIC16 FIXED POINT MULTIPLY ROUTINES VERSION 1.2
;      Input: fixed point arguments in AARG and BARG
;      Output: product AARGxBARG in AARG
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed multiply application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;
;      Routine          Clocks      Function
;      FXM0808S       82 08x08 -> 16 bit signed fixed point multiply
;      FXM0808U       73 08x08 -> 16 bit unsigned fixed point multiply
;      FXM0707U       67 07x07 -> 14 bit unsigned fixed point multiply
;      The above timings are based on the looped macros. If space permits,
;      approximately 29-35 clocks can be saved by using the unrolled macros.

```

AN617

```
list      r=dec,x=on,t=off
include <PIC16.INC>      ; general PIC16 definitions
include <MATH16.INC>    ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI    equ    0x1A    ; random number generator registers
RANDLO    equ    0x1B
TESTCOUNT equ    0x20  ; counter
DATA      equ    0x21    ; beginning of test data
;*****
;*****
;      Test suite for 8x8 bit fixed point multiply algorithms
org       0x0005
MAIN      MOVLW    RAMSTART
          MOVWF    FSR
MEMLOOP   CLRF     INDF
          INCF     FSR
          MOVLW    RAMSTOP
          SUBWF    FSR,W
          BTFSS   _Z
          GOTO    MEMLOOP
          MOVLW    0x45          ; seed for random numbers
          MOVWF    RANDLO
          MOVLW    0x30
          MOVWF    RANDHI
          MOVLW    DATA
          MOVWF    FSR
          BCF     _RP0
          BCF     _RP1
          BCF     _IRP
          CALL    TFXM0808
SELF      GOTO    SELF
RANDOM16   RLF     RANDHI,W      ; random number generator
          XORWF   RANDHI,W
          MOVWF   TEMPB0
          SWAPF   RANDHI
          SWAPF   RANDLO,W
          MOVWF   TEMPB1
          RLF     TEMPB1,W
          RLF     TEMPB1
          MOVF    TEMPB1,W
          XORWF   RANDHI,W
          SWAPF   RANDHI
          ANDLW   0x01
          RLF     TEMPB0
          RLF     RANDLO
          XORWF   RANDLO
          RLF     RANDHI

          RETEW    0
;      Test suite for FXM0808
TFXM0808  MOVLW    3
          MOVWF   TESTCOUNT
M0808LOOP CALL    RANDOM16
          MOVF    RANDHI,W
          MOVWF   BARGB0
          BCF     BARGB0,MSB
          MOVF    BARGB0,W
          MOVWF   INDF
          INCF    FSR
          CALL    RANDOM16
          MOVF    RANDHI,W
          MOVWF   AARGB0
          BCF     AARGB0,MSB
```

```

MOVF          AARGB0,W
MOVWF        INDF
INCF         FSR
CALL        FXM0707U
MOVF          AARGB0,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB1,W
MOVWF        INDF
INCF         FSR
DECFSZ       TESTCOUNT
GOTO        M0808LOOP
RETLW        0x00
;*****
;*****
;      08x08 Bit Multiplication Macros
SMUL0808L    macro
;      Max Timing:      7+10+5*9+8+3 = 73 clks
;      Min Timing:      7+6*6+5+3 = 51 clks
;      PM: 25          DM: 5
                MOVLW          0x07
                MOVWF          LOOPCOUNT
                CLRW
                BTFSC          AARGB0,MSB
                MOVLW          0xFF
                MOVWF          SIGN
                MOVF            AARGB0,W
LOOPSM0808A
                RRF            BARGB0
                BTFSC          _C
                GOTO          LSM0808NA
                DECFSZ        LOOPCOUNT
                GOTO          LOOPSM0808A
                CLRF          AARGB0
                RETLW         0x00
LOOPSM0808
                RRF            BARGB0
                BTFSC          _C
                ADDWF          ACCB0
LSM0808NA
                RLF            SIGN
                RRF            ACCB0
                RRF            ACCB1
                DECFSZ        LOOPCOUNT
                GOTO          LOOPSM0808
                RLF            SIGN
                RRF            ACCB0
                RRF            ACCB1
                endm
UMUL0808L    macro
;      Max Timing:      3+12+6*8+7 = 70 clks
;      Min Timing:      3+7*6+5+3 = 53 clks
;      PM: 19          DM: 4
                MOVLW          0x08
                MOVWF          LOOPCOUNT
                MOVF            AARGB0,W
LOOPUM0808A
                RRF            BARGB0
                BTFSC          _C
                GOTO          LUM0808NAP
                DECFSZ        LOOPCOUNT
                GOTO          LOOPUM0808A
                CLRF          AARGB0
                RETLW         0x00
LUM0808NAP
                BCF            _C
                GOTO          LUM0808NA

```

AN617

```
LOOPUM0808
    RRF          BARGB0
    BTFSC       _C
    ADDWF       ACCB0
LUM0808NA
    RRF          ACCB0
    RRF          ACCB1
    DECFSZ     LOOPCOUNT
    GOTO        LOOPUM0808
    endm

UMUL0707L
    macro
;      Max Timing:    3+12+5*8+7+2 = 64 clks
;      Min Timing:    3+6*6+5+3 = 47 clks
;      PM: 21         DM: 4
    MOVLW       0x07
    MOVWF       LOOPCOUNT
    MOVF        AARGB0,W

LOOPUM0707A
    RRF          BARGB0
    BTFSC       _C
    GOTO        LUM0707NAP
    DECFSZ     LOOPCOUNT
    GOTO        LOOPUM0707A
    CLRF       AARGB0
    RETLW      0x00

LUM0707NAP
    BCF         _C
    GOTO        LUM0707NA

LOOPUM0707
    RRF          BARGB0
    BTFSC       _C
    ADDWF       ACCB0
LUM0707NA
    RRF          ACCB0
    RRF          ACCB1
    DECFSZ     LOOPCOUNT
    GOTO        LOOPUM0707
    RRF          ACCB0
    RRF          ACCB1
    endm

SMUL0808
    macro
;      Max Timing:    5+6+6*5+3 = 44 clks
;      Min Timing:    5+14+3 = 22 clks
;      PM: 5+2*7+5+6*5+3 = 57         DM: 5
    variable i
    i = 0
    CLRW
    BTFSC       AARGB0,MSB
    MOVLW       0xFF
    MOVWF       SIGN
    MOVF        AARGB0,W

    while i < 7

    BTFSC       BARGB0,i
    GOTO        SM0808NA#v(i)
    i = i + 1
    endw
    CLRF       ACCB0           ; if we get here, BARG = 0
    RETEW      0

SM0808NA0
    RLF        SIGN
    RRF        ACCB0
    RRF        ACCB1
    i = 1
    while i < 7
    BTFSC       BARGB0,i
    ADDWF       ACCB0
```

```

SM0808NA#v(i)  RLF          SIGN
                RRF          ACCB0
                RRF          ACCB1
                i = i + 1
                endw
                RLF          SIGN
                RRF          ACCB0
                RRF          ACCB1
                endm
UMUL0808      macro
;      Max Timing:      2+5+7*4 = 35 clks
;      Min Timing:      2+16+3 = 21 clks
;      PM: 2+2*8+4+7*4 = 50          DM: 3
                variable i
                i = 0
                BCF          _C          ; clear carry for first right shift
                MOVF         AARGB0,W

                while i < 8

                BTFSC        BARGB0,i
                GOTO         UM0808NA#v(i)
                i = i + 1
                endw
                CLRF         ACCB0          ; if we get here, BARG = 0
                RETEW        0
UM0808NA0     RRF          ACCB0
                RRF          ACCB1
                i = 1
                while i < 8
                BTFSC        BARGB0,i
                ADDWF        ACCB0
UM0808NA#v(i) RRF          ACCB0
                RRF          ACCB1
                i = i + 1
                endw
                endm
UMUL0707      macro
;      Max Timing:      2+5+6*4+2 = 33 clks
;      Min Timing:      2+14+3 = 19 clks
;      PM: 2+2*7+4+6*4+2 = 46          DM: 3
                variable i
                i = 0
                BCF          _C          ; clear carry for first right shift
                MOVF         AARGB0,W
                while i < 7

                BTFSC        BARGB0,i
                GOTO         UM0707NA#v(i)
                i = i + 1
                endw
                CLRF         ACCB0          ; if we get here, BARG = 0
                RETEW        0
UM0707NA0     RRF          ACCB0
                RRF          ACCB1
                i = 1
                while i < 7
                BTFSC        BARGB0,i
                ADDWF        ACCB0
UM0707NA#v(i) RRF          ACCB0
                RRF          ACCB1
                i = i + 1
                endw
                RRF          ACCB0
                RRF          ACCB1
                endm

```

AN617

```
*****
*****
;
;      8x8 Bit Signed Fixed Point Multiply 8x8 -> 16
;      Input:  8 bit signed fixed point multiplicand in AARGB0
;              8 bit signed fixed point multiplier in BARGB0
;      Use:    CALL    FXM0808S
;      Output: 16 bit signed fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:  4+73+2 = 79 clks          B > 0
;                  7+73+2 = 82 clks          B < 0
;      Min Timing:  4+51 = 55 clks
;      PM: 7+25+1 = 33          DM: 5
FXM0808S      BTFSS      BARGB0,MSB
              GOTO      M0808SOK
              COMF      BARGB0          ; make multiplier BARG > 0
              INCF      BARGB0
              COMF      AARGB0
              INCF      AARGB0
M0808SOK     CLRF      ACCB1          ; clear partial product
              SMUL0808L
              RETLW     0x00
;*****
;*****
;
;      8x8 Bit Unsigned Fixed Point Multiply 8x8 -> 16
;      Input:  8 bit unsigned fixed point multiplicand in AARGB0
;              8 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL    FXM0808U
;      Output: 8 bit unsigned fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:  1+70+2 = 73 clks
;      Min Timing:  1+53 = 54 clks
;      PM: 1+19+1 = 21          DM: 4
FXM0808U     CLRF      ACCB1          ; clear partial product
              UMUL0808L
              RETLW     0x00
;*****
;*****
;
;      7x7 Bit Unsigned Fixed Point Divide 7x7 -> 14
;      Input:  7 bit unsigned fixed point multiplicand in AARGB0
;              7 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL    FXM0707U
;      Output: 14 bit unsigned fixed point product in AARGB0
;      Result: AARG <-- AARG x BARG
;      Max Timing:  1+64+2 = 67 clks
;      Min Timing:  1+47 = 48 clks
;      PM: 1+21+1 = 23          DM: 4
FXM0707U     CLRF      ACCB1          ; clear partial product
              UMUL0707L
              RETLW     0x00
;*****
;*****
;
;      END
```

NOTES:

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX D: PIC16C5X/PIC16CXX DIVIDE ROUTINES

Table of Contents for Appendix D

D.1	32/32 PIC16C5X/PIC16CXX Fixed Point Divide Routines	2-254
D.2	32/24 PIC16C5X/PIC16CXX Fixed Point Divide Routines	2-273
D.3	32/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines	2-288
D.4	24/24 PIC16C5X/PIC16CXX Fixed Point Divide Routines	2-301
D.5	24/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines	2-313
D.6	16/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines	2-323
D.7	16/8 PIC16C5X/PIC16CXX Fixed Point Divide Routines	2-331
D.8	8/8 PIC16C5X/PIC16CXX Fixed Point Divide Routines	2-340

D.1 32/32 PIC16C5X/PIC16CXX Fixed Point Divide Routines

```

; 32/32 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
; Input:  fixed point arguments in AARG and BARG
; Output: quotient AARG/BARG followed by remainder in REM
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed divide application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in performance.
;
; Routine   Clocks   Function
; FXD3232S  912      32 bit/32 bit -> 32.32 signed fixed point divide
; FXD3232U  1031     32 bit/32 bit -> 32.32 unsigned fixed point divide
; FXD3131U  869      31 bit/31 bit -> 31.31 unsigned fixed point divide
;
; list      r=dec,x=on,t=off
; include <PIC16.INC>      ; general PIC16 definitions
; include <MATH16.INC>     ; PIC16 math library definitions
;*****
;*****
; Test suite storage
RANDHI      equ      0x1E      ; random number senerator registers
RANDLO      equ      0x1F
DATA        equ      0x20
;*****
;*****
; Test suite for 32/32 bit fixed point divide algorithms
;
; org      0x0005
MAIN        MOVLW      RAMSTART
;
; MOVWF    FSR
MEMLOOP    CLRf        INDF
;
; INCF     FSR
;
; MOVLW    RAMSTOP
; SUBWF    FSR,W
; BTFSS    _Z
; GOTO     MEMLOOP
; MOVLW    0x45          ; seed for random numbers
; MOVWF    RANDLO
; MOVLW    0x30
; MOVWF    RANDHI
; MOVLW    DATA
; MOVWF    FSR
; BCF      _RP0
; BCF      _RP1
; BCF      _IRP
; CALL     RANDOM16
; MOVF     RANDHI,W
; MOVWF    BARGB0
;
; BCF      BARGB0,MSB
;
; MOVF     BARGB0,W
; MOVWF    INDF
; INCF     FSR
; MOVF     RANDLO,W
; MOVWF    BARGB1

```



```

MOVWF      INDF
INCF
CALL       RANDOM16
MOVF      RANDHI ,W
MOVWF     BARGB2
MOVWF     INDF
INCF
MOVF      RANDLO ,W
MOVWF     BARGB3
MOVWF     INDF
INCF      FSR
CALL     RANDOM16
MOVF     RANDHI ,W
MOVWF    AARGB0
;
;
BCF     AARGB0 ,MSB
MOVF    AARGB0 ,W
MOVWF   INDF
INCF    FSR
MOVF    RANDLO ,W
MOVWF   AARGB1
MOVWF   INDF
INCF    FSR
CALL    RANDOM16

MOVF    RANDHI ,W
MOVWF   AARGB2
MOVWF   INDF
INCF    FSR
MOVF    RANDLO ,W
MOVWF   AARGB3
MOVWF   INDF
INCF    FSR
CALL    FXD3232S
MOVF    AARGB0 ,W
MOVWF   INDF
INCF    FSR
MOVF    AARGB1 ,W
MOVWF   INDF
INCF    FSR
MOVF    AARGB2 ,W
MOVWF   INDF
INCF    FSR
MOVF    AARGB3 ,W
MOVWF   INDF
INCF    FSR
MOVF    REMB0 ,W
MOVWF   INDF
INCF    FSR
MOVF    REMB1 ,W
MOVWF   INDF
INCF    FSR
MOVF    REMB2 ,W
MOVWF   INDF
INCF    FSR
MOVF    REMB3 ,W
MOVWF   INDF
INCF    FSR
SELF
RANDOM16  GOTO    SELF
; random number generator
RLF     RANDHI ,W
XORWF  RANDHI ,W
MOVWF  TEMPB0
SWAPF  RANDHI
SWAPF  RANDLO ,W
MOVWF  TEMPB1
RLF    TEMPB1 ,W
RLF    TEMPB1

```

AN617

```
MOVF          TEMPB1 ,W
XORWF        RANDHI ,W
SWAPF        RANDHI
ANDLW        0x01
RLF          TEMPB0
RLF          RANDLO
XORWF        RANDLO
RLF          RANDHI

RETLW        0
;*****
;*****
;      32/32 Bit Division Macros
SDIV3232L    macro
;      Max Timing:      17+6*27+26+26+6*27+26+26+6*27+26+26+6*27+26+16 = 863 clks
;      Min Timing:      17+6*26+25+25+6*26+25+25+6*26+25+25+6*26+25+3 = 819 clks
;      PM: 17+7*38+16 = 299                                DM: 13
MOVF          BARGB3 ,W
SUBWF        REMB3
MOVF          BARGB2 ,W
BTFSS        _C
INCFSZ       BARGB2 ,W
SUBWF        REMB2
MOVF          BARGB1 ,W
BTFSS        _C
INCFSZ       BARGB1 ,W
SUBWF        REMB1
MOVF          BARGB0 ,W
BTFSS        _C
INCFSZ       BARGB0 ,W
SUBWF        REMB0
RLF          ACCB0
MOVLW        7
MOVWF        LOOPCOUNT
LOOPS3232A  RLF          ACCB0 ,W
RLF          REMB3
RLF          REMB2
RLF          REMB1
RLF          REMB0
MOVF          BARGB3 ,W
BTFSS        ACCB0 ,LSB
GOTO         SADD22LA
SUBWF        REMB3
MOVF          BARGB2 ,W
BTFSS        _C
INCFSZ       BARGB2 ,W
SUBWF        REMB2
MOVF          BARGB1 ,W
BTFSS        _C
INCFSZ       BARGB1 ,W
SUBWF        REMB1
MOVF          BARGB0 ,W
BTFSS        _C
INCFSZ       BARGB0 ,W
SUBWF        REMB0
GOTO         SOK22LA
SADD22LA   ADDWF        REMB3
MOVF          BARGB2 ,W
BTFSC        _C
INCFSZ       BARGB2 ,W
ADDWF        REMB2
MOVF          BARGB1 ,W
BTFSC        _C
INCFSZ       BARGB1 ,W
ADDWF        REMB1
MOVF          BARGB0 ,W
```

	BTFSZ	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
SOK22LA	RLF	ACCB0
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS3232A
	RLF	ACCB1 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTFSZ	ACCB0 , LSB
	GOTO	SADD22L8
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSZ	_C
	INCFSSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSZ	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSZ	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	SOK22L8
SADD22L8	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSZ	_C
	INCFSSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSZ	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSZ	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
SOK22L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS3232B	RLF	ACCB1 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTFSZ	ACCB1 , LSB
	GOTO	SADD22LB
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSZ	_C
	INCFSSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSZ	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSZ	_C
	INCFSSZ	BARGB0 , W

AN617

	SUBWF	REMB0
	GOTO	SOK22LB
SADD22LB	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK22LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS3232B
	RLF	ACCB2 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTFSS	ACCB1 , LSB
	GOTO	SADD22L16
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	SOK22L16
SADD22L16	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK22L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS3232C	RLF	ACCB2 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTFSS	ACCB2 , LSB
	GOTO	SADD22LC

	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTSS	_C
	INCFSSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
SADD22LC	GOTO	SOK22LC
	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTSS	_C
	INCFSSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTSS	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTSS	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
SOK22LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS3232C
	RLF	ACCB3 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTSS	ACCB2 , LSB
	GOTO	SADD22L24
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTSS	_C
	INCFSSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
SADD22L24	GOTO	SOK22L24
	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTSS	_C
	INCFSSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTSS	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTSS	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0

AN617

```

SOK22L24      RLF      ACCB3
               MOVLW    7
               MOVWF   LOOPCOUNT
LOOPS3232D    RLF      ACCB3,W
               RLF      REMB3
               RLF      REMB2
               RLF      REMB1
               RLF      REMB0
               MOVF    BARGB3,W
               BTFSS  ACCB3,LSB
               GOTO   SADD22LD
               SUBWF  REMB3
               MOVF    BARGB2,W
               BTFSS  _C
               INCFSZ BARGB2,W
               SUBWF  REMB2
               MOVF    BARGB1,W
               BTFSS  _C
               INCFSZ BARGB1,W
               SUBWF  REMB1
               MOVF    BARGB0,W
               BTFSS  _C
               INCFSZ BARGB0,W
               SUBWF  REMB0
SADD22LD      GOTO   SOK22LD
               ADDWF  REMB3
               MOVF    BARGB2,W
               BTFSC  _C
               INCFSZ BARGB2,W
               ADDWF  REMB2
               MOVF    BARGB1,W
               BTFSC  _C
               INCFSZ BARGB1,W
               ADDWF  REMB1
               MOVF    BARGB0,W
               BTFSC  _C
               INCFSZ BARGB0,W
               ADDWF  REMB0

SOK22LD       RLF      ACCB3
               DECFSZ LOOPCOUNT
               GOTO   LOOPS3232D
               BTFSC  ACCB3,LSB
               GOTO   SOK22L
               MOVF    BARGB3,W
               ADDWF  REMB3
               MOVF    BARGB2,W
               BTFSC  _C
               INCF   BARGB2,W
               ADDWF  REMB2
               MOVF    BARGB1,W
               BTFSC  _C
               INCF   BARGB1,W
               ADDWF  REMB1
               MOVF    BARGB0,W
               BTFSC  _C
               INCF   BARGB0,W
               ADDWF  REMB0

SOK22L        endm
UDIV3232L     macro
;      Max Timing:    24+6*32+31+31+6*32+31+31+6*32+31+31+6*32+31+16 = 1025 clks
;      Min Timing:    24+6*31+30+30+6*31+30+30+6*31+30+30+6*31+30+3 = 981 clks
;      PM: 359
;
               CLRFB    TEMP

```

	RLF	ACCB0 , W
	RLF	REMB3
	MOVF	BARGB3 , W
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTSS	_C
	INCFSSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTSS	_C
	MOVLW	1
	SUBWF	TEMP
	RLF	ACCB0
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3232A	RLF	ACCB0 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB3 , W
	BTSS	ACCB0 , LSB
	GOTO	UADD22LA
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTSS	_C
	INCFSSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK22LA
UADD22LA	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTSS	_C
	INCFSSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTSS	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTSS	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTSS	_C
	MOVLW	1

AN617

	ADDWF	TEMP
UOK22LA	RLF	ACCB0
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3232A
	RLF	ACCB1 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB3 , W
	BTFSS	ACCB0 , LSB
	GOTO	UADD22L8
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK22L8
UADD22L8	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK22L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3232B	RLF	ACCB1 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB3 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD22LB
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSZ	BARGB2 , W

	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
UADD22LB	GOTO	UOK22LB
	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK22LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3232B
	RLF	ACCB2 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB3 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD22L16
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
UADD22L16	GOTO	UOK22L16
	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSZ	BARGB2 , W

AN617

	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRWF	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK22L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3232C	RLF	ACCB2 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB3 , W
	BTFSS	ACCB2 , LSB
	GOTO	UADD22LC
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRWF	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK22LC
UADD22LC	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRWF	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK22LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3232C
	RLF	ACCB3 , W

	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB3 ,W
	BTFSS	ACCB2 ,LSB
	GOTO	UADD22L24
	SUBWF	REMB3
	MOVF	BARGB2 ,W
	BTFSS	_C
	INCFSZ	BARGB2 ,W
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSZ	BARGB0 ,W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
UADD22L24	GOTO	UOK22L24
	ADDWF	REMB3
	MOVF	BARGB2 ,W
	BTFSC	_C
	INCFSZ	BARGB2 ,W
	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSZ	BARGB0 ,W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK22L24	RLF	ACCB3
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3232D	RLF	ACCB3 ,W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB3 ,W
	BTFSS	ACCB3 ,LSB
	GOTO	UADD22LD
	SUBWF	REMB3
	MOVF	BARGB2 ,W
	BTFSS	_C
	INCFSZ	BARGB2 ,W
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W

AN617

```

    BTFSS          _C
    INCF SZ        BARGB0 , W
    SUBWF          REMB0
    CLRW
    BTFSS          _C
    MOVLW         1
    SUBWF          TEMP
    GOTO           UOK22LD
UADD22LD
    ADDWF          REMB3
    MOVF          BARGB2 , W
    BTFSC         _C
    INCF SZ        BARGB2 , W
    ADDWF          REMB2
    MOVF          BARGB1 , W
    BTFSC         _C
    INCF SZ        BARGB1 , W
    ADDWF          REMB1
    MOVF          BARGB0 , W
    BTFSC         _C
    INCF SZ        BARGB0 , W
    ADDWF          REMB0
    CLRW
    BTFSC         _C
    MOVLW         1
    ADDWF          TEMP

UOK22LD
    RLF           ACCB3
    DECF SZ       LOOPCOUNT
    GOTO           LOOPU3232D
    BTFSC         ACCB3 , LSB
    GOTO           UOK22L
    MOVF          BARGB3 , W
    ADDWF          REMB3
    MOVF          BARGB2 , W
    BTFSC         _C
    INCF          BARGB2 , W
    ADDWF          REMB2
    MOVF          BARGB1 , W
    BTFSC         _C
    INCF          BARGB1 , W
    ADDWF          REMB1
    MOVF          BARGB0 , W
    BTFSC         _C
    INCF          BARGB0 , W
    ADDWF          REMB0

UOK22L
    endm
UDIV3131L macro
;   Max Timing:   17+6*27+26+26+6*27+26+26+6*27+26+26+6*27+26+16 = 863 clks
;   Min Timing:   17+6*26+25+25+6*26+25+25+6*26+25+25+6*26+25+3 = 819 clks
;   PM: 17+7*38+16 = 299                                     DM: 13
    MOVF          BARGB3 , W
    SUBWF          REMB3
    MOVF          BARGB2 , W
    BTFSS         _C
    INCF SZ        BARGB2 , W
    SUBWF          REMB2
    MOVF          BARGB1 , W
    BTFSS         _C
    INCF SZ        BARGB1 , W
    SUBWF          REMB1
    MOVF          BARGB0 , W
    BTFSS         _C
    INCF SZ        BARGB0 , W
    SUBWF          REMB0
    RLF           ACCB0
```

	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3131A	RLF	ACCB0,W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3,W
	BTFSS	ACCB0,LSB
	GOTO	UADD11LA
	SUBWF	REMB3
	MOVF	BARGB2,W
	BTFSS	_C
	INCFSSZ	BARGB2,W
	SUBWF	REMB2
	MOVF	BARGB1,W
	BTFSS	_C
	INCFSSZ	BARGB1,W
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	UOK11LA
UADD11LA	ADDWF	REMB3
	MOVF	BARGB2,W
	BTFSC	_C
	INCFSSZ	BARGB2,W
	ADDWF	REMB2
	MOVF	BARGB1,W
	BTFSC	_C
	INCFSSZ	BARGB1,W
	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSSZ	BARGB0,W
	ADDWF	REMB0
UOK11LA	RLF	ACCB0
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3131A
	RLF	ACCB1,W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3,W
	BTFSS	ACCB0,LSB
	GOTO	UADD11L8
	SUBWF	REMB3
	MOVF	BARGB2,W
	BTFSS	_C
	INCFSSZ	BARGB2,W
	SUBWF	REMB2
	MOVF	BARGB1,W
	BTFSS	_C
	INCFSSZ	BARGB1,W
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	UOK11L8
UADD11L8	ADDWF	REMB3
	MOVF	BARGB2,W
	BTFSC	_C

AN617

	INCFSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK11L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3131B	RLF	ACCB1 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD11LB
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK11LB
UADD11LB	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK11LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3131B
	RLF	ACCB2 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD11L16
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSZ	BARGB2 , W
	SUBWF	REMB2

	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSSZ	BARGB0 ,W
	SUBWF	REMB0
UADD11L16	GOTO	UOK11L16
	ADDWF	REMB3
	MOVF	BARGB2 ,W
	BTFSC	_C
	INCFSSZ	BARGB2 ,W
	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSSZ	BARGB0 ,W
	ADDWF	REMB0
UOK11L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3131C	RLF	ACCB2 ,W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 ,W
	BTFSS	ACCB2 ,LSB
	GOTO	UADD11LC
	SUBWF	REMB3
	MOVF	BARGB2 ,W
	BTFSS	_C
	INCFSSZ	BARGB2 ,W
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSSZ	BARGB0 ,W
	SUBWF	REMB0
GOTO	UOK11LC	
UADD11LC	ADDWF	REMB3
	MOVF	BARGB2 ,W
	BTFSC	_C
	INCFSSZ	BARGB2 ,W
	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSSZ	BARGB0 ,W
	ADDWF	REMB0
UOK11LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3131C
	RLF	ACCB3 ,W

AN617

	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTFSS	ACCB2 , LSB
	GOTO	UADD11L24
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
UADD11L24	GOTO	UOK11L24
	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
UOK11L24	RLF	ACCB3
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3131D	RLF	ACCB3 , W
	RLF	REMB3
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB3 , W
	BTFSS	ACCB3 , LSB
	GOTO	UADD11LD
	SUBWF	REMB3
	MOVF	BARGB2 , W
	BTFSS	_C
	INCFSSZ	BARGB2 , W
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK11LD
UADD11LD	ADDWF	REMB3
	MOVF	BARGB2 , W
	BTFSC	_C
	INCFSSZ	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W


```

        BTFSC          _C
        INCFSZ        BARGB1,W
        ADDWF         REMB1
        MOVF          BARGB0,W
        BTFSC          _C
        INCFSZ        BARGB0,W
        ADDWF         REMB0

UOK11LD      RLF          ACCB3
             DECFSZ      LOOPCOUNT
             GOTO        LOOPU3131D
             BTFSC       ACCB3,LSB
             GOTO        UOK11L
             MOVF        BARGB3,W
             ADDWF       REMB3
             MOVF        BARGB2,W
             BTFSC       _C
             INCF        BARGB2,W
             ADDWF       REMB2
             MOVF        BARGB1,W
             BTFSC       _C
             INCF        BARGB1,W
             ADDWF       REMB1
             MOVF        BARGB0,W
             BTFSC       _C
             INCF        BARGB0,W
             ADDWF       REMB0

UOK11L
             endm
;*****
;*****
;       32/32 Bit Signed Fixed Point Divide 32/32 -> 32.32
;       Input:  32 bit fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               32 bit fixed point divisor in BARGB0, BARGB1, BARGB2, BARGB3
;       Use:    CALL    FXD3232S
;       Output: 32 bit fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;               32 bit fixed point remainder in REMB0, REMB1, REMB2, REMB3
;       Result: AARG, REM <-- AARG / BARG
;       Max Timing:  13+863+3 = 879 clks           A > 0, B > 0
;                   23+863+26 = 912 clks          A > 0, B < 0
;                   23+863+26 = 912 clks          A < 0, B > 0
;                   33+863+3 = 899 clks           A < 0, B < 0
;       Min Timing:  13+819+3 = 835 clks           A > 0, B > 0
;                   23+819+26 = 868 clks          A > 0, B < 0
;                   23+819+26 = 868 clks          A < 0, B > 0
;                   33+819+3 = 855 clks           A < 0, B < 0
;       PM: 33+299+25 = 357                       DM: 13
FXD3232S      MOVF          AARGB0,W
             XORWF        BARGB0,W
             MOVWF       SIGN
             BTFSS       BARGB0,MSB           ; if MSB set, negate BARG
             GOTO        CA3232S
             COMF        BARGB3
             INCF        BARGB3
             BTFSC       _Z
             DECF        BARGB2
             COMF        BARGB2
             BTFSC       _Z
             DECF        BARGB1
             COMF        BARGB1
             BTFSC       _Z
             DECF        BARGB0
             COMF        BARGB0
CA3232S      BTFSS       AARGB0,MSB           ; if MSB set, negate AARG
             GOTO        C3232S
             COMF        AARGB3

```

AN617

```

                INCF                AARGB3
                BTFSC               _Z
                DECF                AARGB2
                COMF                AARGB2
                BTFSC               _Z
                DECF                AARGB1
                COMF                AARGB1
                BTFSC               _Z
                DECF                AARGB0
                COMF                AARGB0
C3232S         CLRF                REMB0
                CLRF                REMB1
                CLRF                REMB2
                CLRF                REMB3
                SDIV3232L
                BTFSS               SIGN,MSB
                RETLW              0x00
                COMF                AARGB3
                INCF                AARGB3
                BTFSC               _Z
                DECF                AARGB2
                COMF                AARGB2
                BTFSC               _Z
                DECF                AARGB1
                COMF                AARGB1
                BTFSC               _Z
                DECF                AARGB0
                COMF                AARGB0
                COMF                REMB3
                INCF                REMB3
                BTFSC               _Z
                DECF                REMB2
                COMF                REMB2
                BTFSC               _Z
                DECF                REMB1
                COMF                REMB1
                BTFSC               _Z
                DECF                REMB0
                COMF                REMB0
                RETLW              0x00
;*****
;*****
;
;       32/32 Bit Unsigned Fixed Point Divide 32/32 -> 32.32
;       Input:  32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               32 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2, BARGB3
;       Use:    CALL    FXD3232U
;       Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;               32 bit unsigned fixed point remainder in REMB0, REMB1, REMB2, REMB3
;       Result: AARG, REM  <--  AARG / BARG
;       Max Timing:  4+1025+2 = 1031 clks
;       Max Timing:  4+981+2 = 987 clks
;       PM: 4+359+1 = 364          DM: 13
FXD3232U       CLRF                REMB0
                CLRF                REMB1
                CLRF                REMB2
                CLRF                REMB3
                UDIV3232L
                RETLW              0x00
;*****
;*****
;
;       31/31 Bit Unsigned Fixed Point Divide 31/31 -> 31.31
;       Input:  31 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               31 bit unsigned fixed point divisor in BARGB0, BARGB1, BARBB2, BARGB3
;       Use:    CALL    FXD3131U

```

```

;      Output: 31 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;      31 bit unsigned fixed point remainder in REMB0, REMB1, REMB2, REMB3
;
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      4+863+2 = 869 clks
;      Min Timing:      4+819+2 = 825 clks
;      PM: 4+299+1 = 304          DM: 13
FXD3131U      CLRF          REMB0
              CLRF          REMB1
              CLRF          REMB2
              CLRF          REMB3
              UDIV3131L
              RETLW         0x00
;*****
;*****
                END

```

D.2 32/24 PIC16C5X/PIC16CXX Fixed Point Divide Routines

```

;      32/24 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
;      Input:  fixed point arguments in AARG and BARG
;      Output: quotient AARG/BARG followed by remainder in REM
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed divide application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;      Routine  Clocks      Function
;      FXD3224S   742        32 bit/24 bit -> 32.24 signed fixed point divide
;      FXD3224U   867        32 bit/24 bit -> 32.24 unsigned fixed point divide
;      FXD3123U   705        31 bit/23 bit -> 31.23 unsigned fixed point divide
                list      r=dec,x=on,t=off
                include <PIC16.INC>      ; general PIC16 definitions
                include <MATH16.INC>     ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI      equ      0x1E      ; random number senerator registers
RANDLO      equ      0x1F
DATA        equ      0x20
;*****
;*****
;      Test suite for 32/24 bit fixed point divide algorithms
                org          0x0005
MAIN         MOVLW        RAMSTART
            MOVWF        FSR
MEMLOOP     CLRF          INDF
            INCF          FSR
            MOVLW        RAMSTOP
            SUBWF        FSR,W
            BTFSS        _Z
            GOTO         MEMLOOP
            MOVLW        0x45          ; seed for random numbers
            MOVWF        RANDLO
            MOVLW        0x30
            MOVWF        RANDHI
            MOVLW        DATA
            MOVWF        FSR
            BCF          _RP0
            BCF          _RP1
            BCF          _IRP
            CALL         RANDOM16
            MOVF         RANDHI,W
            MOVWF        BARGB0
;
            BCF          BARGB0,MSB
;
            MOVF         BARGB0,W
            MOVWF        INDF
            INCF          FSR

```

AN617

```
MOVF          RANDLO,W
MOVWF        BARGB1
MOVWF        INDF
INCF         FSR
CALL        RANDOM16
MOVF          RANDHI,W
MOVWF        BARGB2
MOVWF        INDF
INCF         FSR
CALL        RANDOM16
MOVF          RANDHI,W
MOVWF        AARGB0
;          BCF          AARGB0,MSB
;
MOVF          AARGB0,W
MOVWF        INDF
INCF         FSR
MOVF          RANDLO,W
MOVWF        AARGB1
MOVWF        INDF
INCF         FSR
CALL        RANDOM16

MOVF          RANDHI,W
MOVWF        AARGB2
MOVWF        INDF
INCF         FSR
MOVF          RANDLO,W
MOVWF        AARGB3
MOVWF        INDF
INCF         FSR
CALL        FXD3224S
MOVF          AARGB0,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB1,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB2,W
MOVWF        INDF
INCF         FSR
MOVF          AARGB3,W
MOVWF        INDF
INCF         FSR
MOVF          REMB0,W
MOVWF        INDF
INCF         FSR
MOVF          REMB1,W
MOVWF        INDF
INCF         FSR
MOVF          REMB2,W
MOVWF        INDF
INCF         FSR
SELF
RANDOM16     GOTO        SELF
;          RLF          RANDHI,W          ; random number generator
;          XORWF        RANDHI,W
;          MOVWF        TEMPB0
;          SWAPF        RANDHI
;          SWAPF        RANDLO,W
;          MOVWF        TEMPB1
;          RLF          TEMPB1,W
;          RLF          TEMPB1
;          MOVF          TEMPB1,W
;          XORWF        RANDHI,W
;          SWAPF        RANDHI
;          ANDLW        0x01
;          RLF          TEMPB0
```

```

                RLF          RANDLO
                XORWF       RANDLO
                RLF          RANDHI

                RETLW        0
;*****
;*****
;
;      32/24 Bit Division Macros
SDIV3224L      macro
;      Max Timing:      13+6*22+21+21+6*22+21+21+6*22+21+21+6*22+21+12 = 700 clks
;      Min Timing:      13+6*21+20+20+6*21+20+20+6*21+20+20+6*21+20+3 = 660 clks
;      PM: 11+3*58+43 = 228                                DM: 10
                MOVF          BARGB2,W
                SUBWF        REMB2
                MOVF          BARGB1,W
                BTFSS        _C
                INCFSZ       BARGB1,W
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCFSZ       BARGB0,W
                SUBWF        REMB0
                RLF          ACCB0
                MOVLW        7
                MOVWF        LOOPCOUNT
LOOPS3224A     RLF          ACCB0,W
                RLF          REMB2
                RLF          REMB1
                RLF          REMB0
                MOVF          BARGB2,W
                BTFSS        ACCB0,LSB
                GOTO         SADD24LA
                SUBWF        REMB2
                MOVF          BARGB1,W
                BTFSS        _C
                INCFSZ       BARGB1,W
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCFSZ       BARGB0,W
                SUBWF        REMB0
                GOTO         SOK24LA
SADD24LA      ADDWF          REMB2
                MOVF          BARGB1,W
                BTFSC        _C
                INCFSZ       BARGB1,W
                ADDWF        REMB1
                MOVF          BARGB0,W
                BTFSC        _C
                INCFSZ       BARGB0,W
                ADDWF        REMB0

SOK24LA       RLF          ACCB0
                DECFSZ       LOOPCOUNT
                GOTO         LOOPS3224A
                RLF          ACCB1,W
                RLF          REMB2
                RLF          REMB1
                RLF          REMB0
                MOVF          BARGB2,W
                BTFSS        ACCB0,LSB
                GOTO         SADD24LA8
                SUBWF        REMB2
                MOVF          BARGB1,W
                BTFSS        _C
                INCFSZ       BARGB1,W

```

AN617

	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
SADD24L8	GOTO	SOK24L8
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK24L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS3224B	RLF	ACCB1 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	SADD24LB
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
SADD24LB	GOTO	SOK24LB
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK24LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS3224B
	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	SADD24L16
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	SOK24L16

SADD24L16	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK24L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS3224C	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB2 , LSB
	GOTO	SADD24LC
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	SOK24LC
SADD24LC	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK24LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS3224C
	RLF	ACCB3 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB2 , LSB
	GOTO	SADD24L24
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	SOK24L24
SADD24L24	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W

AN617

```

                BTFSC          _C
                INCF SZ       BARGB0 , W
                ADDWF        REMB0

SOK24L24      RLF           ACCB3
                MOVLW        7
                MOVWF       LOOPCOUNT

LOOPS3224D    RLF           ACCB3 , W
                RLF         REMB2
                RLF         REMB1
                RLF         REMB0
                MOVF        BARGB2 , W
                BTFSS       ACCB3 , LSB
                GOTO        SADD24LD
                SUBWF       REMB2
                MOVF        BARGB1 , W
                BTFSS       _C
                INCF SZ     BARGB1 , W
                SUBWF       REMB1
                MOVF        BARGB0 , W
                BTFSS       _C
                INCF SZ     BARGB0 , W
                SUBWF       REMB0
                GOTO        SOK24LD

SADD24LD      ADDWF        REMB2
                MOVF        BARGB1 , W
                BTFSC       _C
                INCF SZ     BARGB1 , W
                ADDWF       REMB1
                MOVF        BARGB0 , W
                BTFSC       _C
                INCF SZ     BARGB0 , W
                ADDWF       REMB0

SOK24LD       RLF           ACCB3
                DECFSZ      LOOPCOUNT
                GOTO        LOOPS3224D
                BTFSC       ACCB3 , LSB
                GOTO        SOK24L
                MOVF        BARGB2 , W
                ADDWF       REMB2
                MOVF        BARGB1 , W
                BTFSC       _C
                INCF        BARGB1 , W
                ADDWF       REMB1
                MOVF        BARGB0 , W
                BTFSC       _C
                INCF        BARGB0 , W
                ADDWF       REMB0

SOK24L        endm

UDIV3224L     macro
;           Max Timing:      20+6*27+26+26+6*27+26+26+6*27+26+26+6*27+26+12 = 862 clks
;           Min Timing:      20+6*26+25+25+6*26+25+25+6*26+25+25+6*26+25+3 = 822 clks
;           PM: 18+3*75+40+12 = 295                                     DM: 11

                CLRF        TEMP
                RLF         ACCB0 , W
                RLF         REMB2
                MOVF        BARGB2 , W
                SUBWF       REMB2
                MOVF        BARGB1 , W
                BTFSS       _C
                INCF SZ     BARGB1 , W
                SUBWF       REMB1
                MOVF        BARGB0 , W
                BTFSS       _C

```


	INCFSSZ	BARGB0 ,W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	RLF	ACCB0
	MOVLW	7
LOOPU3224A	MOVWF	LOOPCOUNT
	RLF	ACCB0 ,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 ,W
	BTFSS	ACCB0 ,LSB
	GOTO	UADD24LA
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSSZ	BARGB0 ,W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK24LA
UADD24LA	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSSZ	BARGB0 ,W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK24LA	RLF	ACCB0
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3224A
	RLF	ACCB1 ,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 ,W
	BTFSS	ACCB0 ,LSB
	GOTO	UADD24L8
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSSZ	BARGB0 ,W
	SUBWF	REMB0
	CLRW	

AN617

	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK24L8
UADD24L8	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK24L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3224B	RLF	ACCB1 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD24LB
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK24LB
UADD24LB	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK24LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3224B
	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 , W

	BTFSS	ACCB1,LSB
	GOTO	UADD24L16
	SUBWF	REMB2
	MOVF	BARGB1,W
	BTFSS	_C
	INCFSZ	BARGB1,W
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSZ	BARGB0,W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK24L16
UADD24L16	ADDWF	REMB2
	MOVF	BARGB1,W
	BTFSC	_C
	INCFSZ	BARGB1,W
	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSZ	BARGB0,W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK24L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3224C	RLF	ACCB2,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2,W
	BTFSS	ACCB2,LSB
	GOTO	UADD24LC
	SUBWF	REMB2
	MOVF	BARGB1,W
	BTFSS	_C
	INCFSZ	BARGB1,W
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSZ	BARGB0,W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK24LC
UADD24LC	ADDWF	REMB2
	MOVF	BARGB1,W
	BTFSC	_C
	INCFSZ	BARGB1,W
	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSZ	BARGB0,W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C

AN617

	MOVLW	1
	ADDWF	TEMP
UOK24LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3224C
	RLF	ACCB3 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 , W
	BTFSS	ACCB2 , LSB
	GOTO	UADD24L24
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK24L24
UADD24L24	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK24L24	RLF	ACCB3
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3224D	RLF	ACCB3 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 , W
	BTFSS	ACCB3 , LSB
	GOTO	UADD24LD
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP

```

UADD24LD      GOTO      UOK24LD
              ADDWF     REMB2
              MOVF      BARGB1,W
              BTFSC    _C
              INCF     BARGB1,W
              ADDWF     REMB1
              MOVF      BARGB0,W
              BTFSC    _C
              INCF     BARGB0,W
              ADDWF     REMB0
              CLRW
              BTFSC    _C
              MOVLW    1
              ADDWF     TEMP

UOK24LD       RLF       ACCB3
              DECFSZ   LOOPCOUNT
              GOTO     LOOPU3224D
              BTFSC   ACCB3,LSB
              GOTO     UOK24L
              MOVF     BARGB2,W
              ADDWF    REMB2
              MOVF     BARGB1,W
              BTFSC   _C
              INCF    BARGB1,W
              ADDWF   REMB1
              MOVF     BARGB0,W
              BTFSC   _C
              INCF    BARGB0,W
              ADDWF   REMB0

UOK24L
              endm
UDIV3123L    macro
;           Max Timing:  13+6*22+21+21+6*22+21+21+6*22+21+21+6*22+21+12 = 700 clks
;           Min Timing:  13+6*21+20+20+6*21+20+20+6*21+20+20+6*21+20+3 = 660 clks
;           PM: 11+3*58+43 = 228                                     DM: 10
              MOVF     BARGB2,W
              SUBWF   REMB2
              MOVF     BARGB1,W
              BTFSS   _C
              INCF     BARGB1,W
              SUBWF   REMB1
              MOVF     BARGB0,W
              BTFSS   _C
              INCF     BARGB0,W
              SUBWF   REMB0
              RLF     ACCB0
              MOVLW   7
              MOVWF   LOOPCOUNT

LOOPU3123A   RLF     ACCB0,W
              RLF     REMB2
              RLF     REMB1
              RLF     REMB0
              MOVF     BARGB2,W
              BTFSS   ACCB0,LSB
              GOTO     UADD13LA
              SUBWF   REMB2
              MOVF     BARGB1,W
              BTFSS   _C
              INCF     BARGB1,W
              SUBWF   REMB1
              MOVF     BARGB0,W
              BTFSS   _C
              INCF     BARGB0,W
              SUBWF   REMB0
              GOTO     UOK13LA

```

AN617

UADD13LA	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK13LA	RLF	ACCB0
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3123A
	RLF	ACCB1 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB0 , LSB
	GOTO	UADD13L8
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK13L8
UADD13L8	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK13L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3123B	RLF	ACCB1 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD13LB
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK13LB
UADD13LB	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W

	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK13LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3123B
	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD13L16
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK13L16
UADD13L16	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK13L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3123C	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB2 , LSB
	GOTO	UADD13LC
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK13LC
UADD13LC	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK13LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT

AN617

	GOTO	LOOPU3123C
	RLF	ACCB3 ,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 ,W
	BTFSS	ACCB2 ,LSB
	GOTO	UADD13L24
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSZ	BARGB0 ,W
	SUBWF	REMB0
	GOTO	UOK13L24
UADD13L24	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSZ	BARGB0 ,W
	ADDWF	REMB0
UOK13L24	RLF	ACCB3
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3123D	RLF	ACCB3 ,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 ,W
	BTFSS	ACCB3 ,LSB
	GOTO	UADD13LD
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSZ	BARGB0 ,W
	SUBWF	REMB0
	GOTO	UOK13LD
UADD13LD	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSZ	BARGB0 ,W
	ADDWF	REMB0
UOK13LD	RLF	ACCB3
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3123D
	BTFSC	ACCB3 ,LSB
	GOTO	UOK13L
	MOVF	BARGB2 ,W
	ADDWF	REMB2
	MOVF	BARGB1 ,W


```

        BTFSC          _C
        INCF           BARGB1,W
        ADDWF          REMB1
        MOVF           BARGB0,W
        BTFSC          _C
        INCF           BARGB0,W
        ADDWF          REMB0

UOK13L
        endm
;*****
;*****
;
;   32/24 Bit Signed Fixed Point Divide 32/24 -> 32.24
;   Input:  32 bit fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;           24 bit fixed point divisor in BARGB0, BARGB1, BARGB2
;   Use:    CALL    FXD3224S
;   Output: 32 bit fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;           24 bit fixed point remainder in REMB0, REMB1, REMB2
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:      12+700+3 = 715 clks           A > 0, B > 0
;                   19+700+23 = 742 clks         A > 0, B < 0
;                   19+700+23 = 742 clks         A < 0, B > 0
;                   29+700+3 = 732 clks         A < 0, B < 0
;   Min Timing:      12+660+3 = 675 clks         A > 0, B > 0
;                   19+660+23 = 702 clks         A > 0, B < 0
;                   19+660+23 = 702 clks         A < 0, B > 0
;                   29+660+3 = 692 clks         A < 0, B < 0
;   PM: 29+228+23 = 280                          DM: 11
FXD3224S    MOVF           AARGB0,W
            XORWF          BARGB0,W
            MOVWF          SIGN
            BTFSS          BARGB0,MSB           ; if MSB set, negate BARG
            GOTO           CA3224S
            COMF           BARGB2
            INCF           BARGB2
            BTFSC          _Z
            DECF           BARGB1
            COMF           BARGB1
            BTFSC          _Z
            DECF           BARGB0
            COMF           BARGB0
CA3224S    BTFSS          AARGB0,MSB           ; if MSB set, negate AARG
            GOTO           C3224S
            COMF           AARGB3
            INCF           AARGB3
            BTFSC          _Z
            DECF           AARGB2
            COMF           AARGB2
            BTFSC          _Z
            DECF           AARGB1
            COMF           AARGB1
            BTFSC          _Z
            DECF           AARGB0
            COMF           AARGB0
C3224S    CLRf           REMB0
            CLRf           REMB1
            CLRf           REMB2
            SDIV3224L
            BTFSS          SIGN,MSB
            RETLW          0x00
            COMF           AARGB3
            INCF           AARGB3
            BTFSC          _Z
            DECF           AARGB2
            COMF           AARGB2
            BTFSC          _Z

```

AN617

```

    DECF          AARGB1
    COMF          AARGB1
    BTFSC        _Z
    DECF          AARGB0
    COMF          AARGB0
    COMF          REMB2
    INCF          REMB2
    BTFSC        _Z
    DECF          REMB1
    COMF          REMB1
    BTFSC        _Z
    DECF          REMB0
    COMF          REMB0
    RETLW        0x00
;*****
;*****
;
;   32/24 Bit Unsigned Fixed Point Divide 32/24 -> 32.24
;   Input:  32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;   Use:    CALL   FXD3224U
;   Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;           24 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:    3+862+2 = 867 clks
;   Min Timing:    3+822+2 = 827 clks
;   PM: 3+295+1 = 299          DM: 11
FXD3224U    CLRF          REMB0
            CLRF          REMB1
            CLRF          REMB2
            UDIV3224L
            RETLW        0x00
;*****
;*****
;
;   31/23 Bit Unsigned Fixed Point Divide 31/23 -> 31.23
;   Input:  31 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;   Use:    CALL   FXD3123U
;   Output: 31 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;           23 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:    3+700+2 = 705 clks
;   Min Timing:    3+660+2 = 665 clks
;   PM: 3+228+1 = 232          DM: 10
FXD3123U    CLRF          REMB0
            CLRF          REMB1
            CLRF          REMB2
            UDIV3123L
            RETLW        0x00
;*****
;*****
;
;   END

```

D.3 32/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines

```

;   32/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
;   Input:  fixed point arguments in AARG and BARG
;   Output: quotient AARG/BARG followed by remainder in REM
;   All timings are worst case cycle counts
;   It is useful to note that the additional unsigned routines requiring a non-power of two
;   argument can be called in a signed divide application where it is known that the
;   respective argument is nonnegative, thereby offering some improvement in
;   performance.
;
;   Routine          Clocks      Function
;   FXD3216S        578 32 bit/16 bit -> 32.16 signed fixed point divide
;   FXD3216U        703 32 bit/16 bit -> 32.16 unsigned fixed point divide

```

```

;      FXD3115U    541 31 bit/15 bit -> 31.15 unsigned fixed point divide
                list    r=dec,x=on,t=off
                include <PIC16.INC>      ; general PIC16 definitions
                include <MATH16.INC>     ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI         equ    0x1E      ; random number senerator registers
RANDLO         equ    0x1F
TDATA          equ    0x20
;*****
;*****
;      Test suite for 32/16 bit fixed point divide algorithms
                org      0x0005
MAIN           MOVLW    RAMSTART
                MOVWF   FSR
MEMLOOP       CLRF     INDF
                INCF    FSR
                MOVLW   RAMSTOP
                SUBWF   FSR,W
                BTFSS  _Z
                GOTO   MEMLOOP
                MOVLW   0x45                ; seed for random numbers
                MOVWF  RANDLO
                MOVLW   0x30
                MOVWF  RANDHI
                MOVLW   TDATA
                MOVWF  FSR
                BCF    _RP0
                BCF    _RP1
                BCF    _IRP
                CALL   RANDOM16
;
;      SWAPF    RANDHI
;      SWAPF    RANDLO
                MOVF   RANDHI,W
                MOVWF  BARGB0
;
;      BCF     BARGB0,MSB
;
;      MOVF    BARGB0,W
                MOVWF  INDF
                INCF   FSR
                MOVF   RANDLO,W
                MOVWF  BARGB1
                MOVWF  INDF
                INCF   FSR
                CALL   RANDOM16

;
;      SWAPF    RANDHI
;      SWAPF    RANDLO
                MOVF   RANDHI,W
                MOVWF  AARGB0
;
;      BCF     AARGB0,MSB
;
;      MOVF    AARGB0,W
                MOVWF  INDF
                INCF   FSR
                MOVF   RANDLO,W
                MOVWF  AARGB1
                MOVWF  INDF
                INCF   FSR
                CALL   RANDOM16

                MOVF   RANDHI,W
                MOVWF  AARGB2
                MOVWF  INDF
                INCF   FSR
                MOVF   RANDLO,W
                MOVWF  AARGB3

```

AN617

```

MOVWF      INDF
INCF       FSR
CALL      FXD3216S
MOVF      AARGB0,W
MOVWF     INDF
INCF      FSR
MOVF      AARGB1,W
MOVWF     INDF
INCF      FSR
MOVF      AARGB2,W
MOVWF     INDF
INCF      FSR
MOVF      AARGB3,W
MOVWF     INDF
INCF      FSR
MOVF      REMB0,W
MOVWF     INDF
INCF      FSR
MOVF      REMB1,W
MOVWF     INDF
INCF      FSR
SELF      GOTO      SELF
RANDOM16   RLF      RANDHI,W           ; random number generator
XORWF     RANDHI,W
MOVWF     TEMPB0
SWAPF    RANDHI
SWAPF    RANDLO,W
MOVWF     TEMPB1
RLF      TEMPB1,W
RLF      TEMPB1
MOVF     TEMPB1,W
XORWF    RANDHI,W
SWAPF    RANDHI
ANDLW    0x01
RLF      TEMPB0
RLF      RANDLO
XORWF    RANDLO
RLF      RANDHI

RETLW    0
;*****
;*****
;      32/16 Bit Division Macros
SDIV3216L macro
;      Max Timing:      9+6*17+16+16+6*17+16+16+6*17+16+16+6*17+16+8 = 537 clks
;      Min Timing:      9+6*16+15+15+6*16+15+15+6*16+15+15+6*16+15+3 = 501 clks
;      PM: 157          DM: 9

MOVWF    BARGB1,W
SUBWF    REMB1
MOVF     BARGB0,W
BTFSS   _C
INCFSZ  BARGB0,W
SUBWF    REMB0
RLF     ACCB0
MOVLW   7
MOVWF   LOOPCOUNT
LOOPS3216A RLF     ACCB0,W
RLF     REMB1
RLF     REMB0
MOVF    BARGB1,W
BTFSS  ACCB0,LSB
GOTO   SADD26LA
SUBWF  REMB1
MOVF   BARGB0,W
BTFSS  _C
INCFSZ BARGB0,W

```

	SUBWF	REMB0
	GOTO	SOK26LA
SADD26LA	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSZ	BARGB0,W
	ADDWF	REMB0
SOK26LA	RLF	ACCB0
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS3216A
	RLF	ACCB1,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB0,LSB
	GOTO	SADD26L8
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	SOK26L8
SADD26L8	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSZ	BARGB0,W
	ADDWF	REMB0
SOK26L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS3216B	RLF	ACCB1,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB1,LSB
	GOTO	SADD26LB
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	SOK26LB
SADD26LB	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSZ	BARGB0,W
	ADDWF	REMB0
SOK26LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS3216B
	RLF	ACCB2,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB1,LSB
	GOTO	SADD26L16
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	SOK26L16
SADD26L16	ADDWF	REMB1

AN617

	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK26L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS3216C	RLF	ACCB2 , W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1 , W
	BTFSS	ACCB2 , LSB
	GOTO	SADD26LC
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
GOTO		SOK26LC
SADD26LC	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK26LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS3216C
	RLF	ACCB3 , W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1 , W
	BTFSS	ACCB2 , LSB
	GOTO	SADD26L24
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
GOTO		SOK26L24
SADD26L24	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
SOK26L24	RLF	ACCB3
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS3216D	RLF	ACCB3 , W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1 , W
	BTFSS	ACCB3 , LSB
	GOTO	SADD26LD
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
GOTO		SOK26LD
SADD26LD	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W

```

                ADDWF          REMB0

SOK26LD        RLF            ACCB3
                DECFSZ        LOOPCOUNT
                GOTO          LOOPS3216D
                BTFSC        ACCB3,LSB
                GOTO          SOK26L
                MOVF          BARGB1,W
                ADDWF        REMB1
                MOVF          BARGB0,W
                BTFSC        _C
                INCF          BARGB0,W
                ADDWF        REMB0

SOK26L
                endm
UDIV3216L     macro
;           Max Timing:      16+6*22+21+21+6*22+21+21+6*22+21+21+6*22+21+8 = 699 clks
;           Min Timing:      16+6*21+20+20+6*21+20+20+6*21+20+20+6*21+20+3 = 663 clks
;           PM: 240                                DM: 9

                CLRF          TEMP
                RLF            ACCB0,W
                RLF            REMB1
                MOVF          BARGB1,W
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCFSZ       BARGB0,W
                SUBWF        REMB0
                CLRW
                BTFSS        _C
                MOVLW        1
                SUBWF        TEMP
                RLF            ACCB0
                MOVLW        7
                MOVWF        LOOPCOUNT
LOOPU3216A    RLF            ACCB0,W
                RLF            REMB1
                RLF            REMB0
                RLF            TEMP
                MOVF          BARGB1,W
                BTFSS        ACCB0,LSB
                GOTO          UADD26LA
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCFSZ       BARGB0,W
                SUBWF        REMB0
                CLRW
                BTFSS        _C
                MOVLW        1
                SUBWF        TEMP
                GOTO          UOK26LA
UADD26LA     ADDWF          REMB1
                MOVF          BARGB0,W
                BTFSC        _C
                INCFSZ       BARGB0,W
                ADDWF        REMB0
                CLRW
                BTFSC        _C
                MOVLW        1
                ADDWF        TEMP

UOK26LA      RLF            ACCB0
                DECFSZ        LOOPCOUNT
                GOTO          LOOPU3216A
                RLF            ACCB1,W

```

AN617

	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB1 , W
	BTFSS	ACCB0 , LSB
	GOTO	UADD26L8
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
UADD26L8	GOTO	UOK26L8
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK26L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3216B	RLF	ACCB1 , W
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB1 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD26LB
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
UADD26LB	GOTO	UOK26LB
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK26LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3216B
	RLF	ACCB2 , W
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB1 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD26L16

	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK26L16
UADD26L16	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK26L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3216C	RLF	ACCB2 , W
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB1 , W
	BTFSS	ACCB2 , LSB
	GOTO	UADD26LC
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK26LC
UADD26LC	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK26LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3216C
	RLF	ACCB3 , W
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB1 , W
	BTFSS	ACCB2 , LSB
	GOTO	UADD26L24
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	

AN617

```

                BTFSS          _C
                MOVLW         1
                SUBWF         TEMP
                GOTO          UOK26L24
UADD26L24      ADDWF          REMB1
                MOVF          BARGB0,W
                BTFSC         _C
                INCF          BARGB0,W
                ADDWF         REMB0
                CLRW
                BTFSC         _C
                MOVLW         1
                ADDWF         TEMP

UOK26L24       RLF            ACCB3
                MOVLW         7
                MOVWF         LOOPCOUNT
LOOPU3216D     RLF            ACCB3,W
                RLF            REMB1
                RLF            REMB0
                RLF            TEMP
                MOVF          BARGB1,W
                BTFSS         ACCB3,LSB
                GOTO          UADD26LD
                SUBWF         REMB1
                MOVF          BARGB0,W
                BTFSS         _C
                INCF          BARGB0,W
                SUBWF         REMB0
                CLRW
                BTFSS         _C
                MOVLW         1
                SUBWF         TEMP
                GOTO          UOK26LD
UADD26LD      ADDWF          REMB1
                MOVF          BARGB0,W
                BTFSC         _C
                INCF          BARGB0,W
                ADDWF         REMB0
                CLRW
                BTFSC         _C
                MOVLW         1
                ADDWF         TEMP

UOK26LD       RLF            ACCB3
                DECF          LOOPCOUNT
                GOTO          LOOPU3216D
                BTFSC         ACCB3,LSB
                GOTO          UOK26L
                MOVF          BARGB1,W
                ADDWF         REMB1
                MOVF          BARGB0,W
                BTFSC         _C
                INCF          BARGB0,W
                ADDWF         REMB0

UOK26L        endm
UDIV3115L     macro
;           Max Timing:      9+6*17+16+16+6*17+16+16+6*17+16+16+6*17+16+8 = 537 clks
;           Min Timing:      9+6*16+15+15+6*16+15+15+6*16+15+15+6*16+15+3 = 501 clks
;           PM: 157                                     DM: 9
                MOVF          BARGB1,W
                SUBWF         REMB1
                MOVF          BARGB0,W
                BTFSS         _C
                INCF          BARGB0,W
```

	SUBWF	REMB0
	RLF	ACCB0
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3115A	RLF	ACCB0,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB0,LSB
	GOTO	UADD15LA
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	UOK15LA
UADD15LA	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSSZ	BARGB0,W
	ADDWF	REMB0
UOK15LA	RLF	ACCB0
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3115A
	RLF	ACCB1,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB0,LSB
	GOTO	UADD15L8
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	UOK15L8
UADD15L8	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSSZ	BARGB0,W
	ADDWF	REMB0
UOK15L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3115B	RLF	ACCB1,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB1,LSB
	GOTO	UADD15LB
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	UOK15LB
UADD15LB	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSSZ	BARGB0,W
	ADDWF	REMB0
UOK15LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT

AN617

	GOTO	LOOPU3115B
	RLF	ACCB2,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB1,LSB
	GOTO	UADD15L16
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	UOK15L16
UADD15L16	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSZ	BARGB0,W
	ADDWF	REMB0
UOK15L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3115C	RLF	ACCB2,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB2,LSB
	GOTO	UADD15LC
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSZ	BARGB0,W
	SUBWF	REMB0
UADD15LC	GOTO	UOK15LC
	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSZ	BARGB0,W
	ADDWF	REMB0
UOK15LC	RLF	ACCB2
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU3115C
	RLF	ACCB3,W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1,W
	BTFSS	ACCB2,LSB
	GOTO	UADD15L24
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSZ	BARGB0,W
	SUBWF	REMB0
UADD15L24	GOTO	UOK15L24
	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSZ	BARGB0,W
	ADDWF	REMB0
UOK15L24	RLF	ACCB3
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU3115D	RLF	ACCB3,W
	RLF	REMB1

```

                RLF                REMB0
                MOVF               BARGB1,W
                BTFSS             ACCB3,LSB
                GOTO              UADD15LD
                SUBWF             REMB1
                MOVF               BARGB0,W
                BTFSS             _C
                INCFSZ            BARGB0,W
                SUBWF             REMB0
                GOTO              UOK15LD
UADD15LD      ADDWF               REMB1
                MOVF               BARGB0,W
                BTFSC             _C
                INCFSZ            BARGB0,W
                ADDWF             REMB0

UOK15LD      RLF                ACCB3
                DECFSZ            LOOPCOUNT
                GOTO              LOOPU3115D
                BTFSC             ACCB3,LSB
                GOTO              UOK15L
                MOVF               BARGB1,W
                ADDWF             REMB1
                MOVF               BARGB0,W
                BTFSC             _C
                INCF              BARGB0,W
                ADDWF             REMB0

UOK15L
                endm

;*****
;*****

;      32/16 Bit Signed Fixed Point Divide 32/16 -> 32.16
;      Input:  32 bit fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;              16 bit fixed point divisor in BARGB0, BARGB1
;      Use:    CALL    FXD3216S
;      Output: 32 bit fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;              16 bit fixed point remainder in REMB0, REMB1
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      11+537+3 = 551 clks           A > 0, B > 0
;                      15+537+20 = 572 clks          A > 0, B < 0
;                      21+537+20 = 578 clks          A < 0, B > 0
;                      25+537+3 = 565 clks           A < 0, B < 0
;      Min Timing:      11+501+3 = 515 clks           A > 0, B > 0
;                      15+501+20 = 536 clks          A > 0, B < 0
;                      21+501+20 = 542 clks          A < 0, B > 0
;                      25+501+3 = 529 clks           A < 0, B < 0
;      PM: 25+157+19 = 201           DM: 10
FXD3216S      MOVF               AARGB0,W
                XORWF             BARGB0,W
                MOVWF            SIGN
                BTFSS             BARGB0,MSB           ; if MSB set, negate BARG
                GOTO              CA3216S
                COMF              BARGB1
                INCF              BARGB1
                BTFSC             _Z
                DECF              BARGB0
                COMF              BARGB0
CA3216S      BTFSS             AARGB0,MSB           ; if MSB set, negate AARG
                GOTO              C3216S
                COMF              AARGB3
                INCF              AARGB3
                BTFSC             _Z
                DECF              AARGB2
                COMF              AARGB2

```

AN617

```

        BTFSC          _Z
        DECF           AARGB1
        COMF           AARGB1
        BTFSC          _Z
        DECF           AARGB0
        COMF           AARGB0
C3216S   CLRF          REMB0
        CLRF          REMB1
        SDIV3216L
        BTFSS          SIGN,MSB
        RETLW          0x00
        COMF           AARGB3
        INCF           AARGB3
        BTFSC          _Z
        DECF           AARGB2
        COMF           AARGB2
        BTFSC          _Z
        DECF           AARGB1
        COMF           AARGB1
        BTFSC          _Z
        DECF           AARGB0
        COMF           AARGB0
        COMF           REMB1
        INCF           REMB1
        BTFSC          _Z
        DECF           REMB0
        COMF           REMB0
        RETLW          0x00
;*****
;*****
;
;   32/16 Bit Unsigned Fixed Point Divide 32/16 -> 32.16
;   Input:  32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;           16 bit unsigned fixed point divisor in BARGB0, BARGB1
;   Use:    CALL    FXD3216U
;   Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;           16 bit unsigned fixed point remainder in REMB0, REMB1
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:  2+699+2 = 703 clks
;   Max Timing:  2+663+2 = 667 clks
;   PM: 2+240+1 = 243          DM: 9
FXD3216U   CLRF          REMB0
        CLRF          REMB1
        UDIV3216L
        RETLW          0x00
;*****
;*****
;
;   31/15 Bit Unsigned Fixed Point Divide 31/15 -> 31.15
;   Input:  31 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;           15 bit unsigned fixed point divisor in BARGB0, BARGB1
;   Use:    CALL    FXD3115U
;   Output: 31 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;           15 bit unsigned fixed point remainder in REMB0, REMB1
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:  2+537+2 = 541 clks
;   Min Timing:  2+501+2 = 505 clks
;   PM: 2+157+1 = 160          DM: 9
FXD3115U   CLRF          REMB0
        CLRF          REMB1
        UDIV3115L
        RETLW          0x00
;*****
;*****
        END
```

D.4 24/24 PIC16C5X/PIC16CXX Fixed Point Divide Routines

```

;      24/24 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
;      Input:  fixed point arguments in AARG and BARG
;      Output: quotient AARG/BARG followed by remainder in REM
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed divide application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;      Routine           Clocks      Function
;      FXD2424S          565         24 bit/24 bit -> 24.24 signed fixed point divide
;      FXD2424U          676         24 bit/24 bit -> 24.24 unsigned fixed point divide
;      FXD2323U          531         23 bit/23 bit -> 23.23 unsigned fixed point divide
;
;      list      r=dec,x=on,t=off
;      include  <PIC16.INC>      ; general PIC16 definitions
;      include  <MATH16.INC>    ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI      equ      0x1A      ; random number senerator registers
RANDLO      equ      0x1B
DATA        equ      0x20
;*****
;*****
;      Test suite for 24/24 bit fixed point divide algorithms
MAIN        org          0x0005
            MOVLW        RAMSTART
            MOVWF        FSR
MEMLOOP     CLRF        INDF
            INCF        FSR
            MOVLW        RAMSTOP
            SUBWF        FSR,W
            BTFSS        _Z
            GOTO        MEMLOOP
            MOVLW        0x45          ; seed for random numbers
            MOVWF        RANDLO
            MOVLW        0x30
            MOVWF        RANDHI
            MOVLW        DATA
            MOVWF        FSR
            BCF          _RP0
            BCF          _RP1
            BCF          _IRP
            CALL        RANDOM16
            MOVF         RANDHI,W
            MOVWF        BARGB0
;      BCF          BARGB0,MSB
;      MOVF         BARGB0,W
            MOVWF        INDF
            INCF        FSR
            MOVF         RANDLO,W
            MOVWF        BARGB1
            MOVWF        INDF
            INCF        FSR
            CALL        RANDOM16
            MOVF         RANDHI,W
            MOVWF        BARGB2
            MOVWF        INDF
            INCF        FSR
            CALL        RANDOM16
            MOVF         RANDHI,W
            MOVWF        AARGB0
;      BCF          AARGB0,MSB
;      MOVF         AARGB0,W
            MOVWF        INDF

```

AN617

```

        INCF          FSR
        MOVF          RANDLO,W
        MOVWF        AARGB1
        MOVWF        INDF
        INCF          FSR
        CALL         RANDOM16

        MOVF          RANDHI,W
        MOVWF        AARGB2
        MOVWF        INDF
        INCF          FSR
        CALL         FXD2424S
        MOVF          AARGB0,W
        MOVWF        INDF
        INCF          FSR
        MOVF          AARGB1,W
        MOVWF        INDF
        INCF          FSR
        MOVF          AARGB2,W
        MOVWF        INDF
        INCF          FSR
        MOVF          REMB0,W
        MOVWF        INDF
        INCF          FSR
        MOVF          REMB1,W
        MOVWF        INDF
        INCF          FSR
        MOVF          REMB2,W
        MOVWF        INDF
        INCF          FSR
        GOTO         SELF
SELF
RANDOM16
        RLF          RANDHI,W          ; random number generator
        XORWF        RANDHI,W
        MOVWF        TEMPB0
        SWAPF        RANDHI
        SWAPF        RANDLO,W
        MOVWF        TEMPB1
        RLF          TEMPB1,W
        RLF          TEMPB1
        MOVF          TEMPB1,W
        XORWF        RANDHI,W
        SWAPF        RANDHI
        ANDLW        0x01
        RLF          TEMPB0
        RLF          RANDLO
        XORWF        RANDLO
        RLF          RANDHI

        RETLW        0
;*****
;*****
;      24/24 Bit Division Macros
SDIV2424L      macro
;      Max Timing:    13+6*22+21+21+6*22+21+21+6*22+21+12 = 526 clks
;      Min Timing:    13+6*21+20+20+6*21+20+20+6*21+20+3 = 494 clks
;      PM: 11+3*51+31+12 = 207                                DM: 12
        MOVF          BARGB2,W
        SUBWF        REMB2
        MOVF          BARGB1,W
        BTFSS        _C
        INCFSZ       BARGB1,W
        SUBWF        REMB1
        MOVF          BARGB0,W
        BTFSS        _C
        INCFSZ       BARGB0,W
        SUBWF        REMB0

```


	RLF	ACCB0
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS2424A	RLF	ACCB0,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2,W
	BTFSS	ACCB0,LSB
	GOTO	SADD44LA
	SUBWF	REMB2
	MOVF	BARGB1,W
	BTFSS	_C
	INCFSSZ	BARGB1,W
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	SOK44LA
SADD44LA	ADDWF	REMB2
	MOVF	BARGB1,W
	BTFSC	_C
	INCFSSZ	BARGB1,W
	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSSZ	BARGB0,W
	ADDWF	REMB0
SOK44LA	RLF	ACCB0
	DECFSSZ	LOOPCOUNT
	GOTO	LOOPS2424A
	RLF	ACCB1,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2,W
	BTFSS	ACCB0,LSB
	GOTO	SADD44L8
	SUBWF	REMB2
	MOVF	BARGB1,W
	BTFSS	_C
	INCFSSZ	BARGB1,W
	SUBWF	REMB1
	MOVF	BARGB0,W
	BTFSS	_C
	INCFSSZ	BARGB0,W
	SUBWF	REMB0
	GOTO	SOK44L8
SADD44L8	ADDWF	REMB2
	MOVF	BARGB1,W
	BTFSC	_C
	INCFSSZ	BARGB1,W
	ADDWF	REMB1
	MOVF	BARGB0,W
	BTFSC	_C
	INCFSSZ	BARGB0,W
	ADDWF	REMB0
SOK44L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS2424B	RLF	ACCB1,W
	RLF	REMB2
	RLF	REMB1

AN617

	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	SADD44LB
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
SADD44LB	GOTO	SOK44LB
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
SOK44LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPS2424B
	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	SADD44L16
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
SADD44L16	GOTO	SOK44L16
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
SOK44L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPS2424C	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 , W
	BTFSS	ACCB2 , LSB
	GOTO	SADD44LC
	SUBWF	REMB2
	MOVF	BARGB1 , W

```

                BTFSS          _C
                INCFSZ         BARGB1,W
                SUBWF          REMB1
                MOVF           BARGB0,W
                BTFSS          _C
                INCFSZ         BARGB0,W
                SUBWF          REMB0
SADD44LC      GOTO           SOK44LC
                ADDWF          REMB2
                MOVF           BARGB1,W
                BTFSC          _C
                INCFSZ         BARGB1,W
                ADDWF          REMB1
                MOVF           BARGB0,W
                BTFSC          _C
                INCFSZ         BARGB0,W
                ADDWF          REMB0

SOK44LC      RLF             ACCB2
                DECFSZ        LOOPCOUNT
                GOTO          LOOPS2424C
                BTFSC         ACCB2,LSB
                GOTO          SOK44L
                MOVF          BARGB2,W
                ADDWF         REMB2
                MOVF          BARGB1,W
                BTFSC         _C
                INCF          BARGB1,W
                ADDWF         REMB1
                MOVF          BARGB0,W
                BTFSC         _C
                INCF          BARGB0,W
                ADDWF         REMB0

SOK44L
                endm
UDIV2424L    macro
;           Max Timing:      20+6*28+27+27+6*28+27+27+6*28+27+12 = 671 clks
;           Min Timing:      20+6*27+26+26+6*27+26+26+6*27+26+3 = 639 clks
;           PM: 18+2*76+40+12 = 222                                     DM: 13
                CLRF          TEMP
                RLF           ACCB0,W
                RLF           REMB2
                MOVF          BARGB2,W
                SUBWF         REMB2
                MOVF          BARGB1,W
                BTFSS         _C
                INCFSZ        BARGB1,W
                SUBWF         REMB1
                MOVF          BARGB0,W
                BTFSS         _C
                INCFSZ        BARGB0,W
                SUBWF         REMB0
                CLRW
                BTFSS         _C
                MOVLW         1
                SUBWF         TEMP
                RLF           ACCB0
                MOVLW         7
                MOVWF         LOOPCOUNT
LOOPU2424A   RLF           ACCB0,W
                RLF           REMB2
                RLF           REMB1
                RLF           REMB0
                RLF           TEMP
                MOVF          BARGB2,W
                BTFSS         ACCB0,LSB

```

AN617

	GOTO	UADD44LA
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSSZ	BARGB0 ,W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
UADD44LA	GOTO	UOK44LA
	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSSZ	BARGB0 ,W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK44LA	RLF	ACCB0
	DECFSSZ	LOOPCOUNT
	GOTO	LOOPU2424A
	RLF	ACCB1 ,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 ,W
	BTFSS	ACCB0 ,LSB
	GOTO	UADD44L8
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSSZ	BARGB0 ,W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
UADD44L8	GOTO	UOK44L8
	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSSZ	BARGB0 ,W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1

	ADDWF	TEMP
UOK44L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU2424B	RLF	ACCB1 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD44LB
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK44LB
UADD44LB	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	ADDWF	TEMP
UOK44LB	RLF	ACCB1
	DECFSSZ	LOOPCOUNT
	GOTO	LOOPU2424B
	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD44L16
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCFSSZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK44L16

AN617

UADD44L16	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCF SZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCF SZ	BARGB0 , W
	ADDWF	REMB0
	CLR W	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK44L16	RLF	ACCB2
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU2424C	RLF	ACCB2 , W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB2 , W
	BTFSS	ACCB2 , LSB
	GOTO	UADD44LC
	SUBWF	REMB2
	MOVF	BARGB1 , W
	BTFSS	_C
	INCF SZ	BARGB1 , W
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCF SZ	BARGB0 , W
	SUBWF	REMB0
	CLR W	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK44LC
UADD44LC	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCF SZ	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCF SZ	BARGB0 , W
	ADDWF	REMB0
	CLR W	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK44LC	RLF	ACCB2
	DECF SZ	LOOPCOUNT
	GOTO	LOOPU2424C
	BTFSC	ACCB2 , LSB
	GOTO	UOK44L
	MOVF	BARGB2 , W
	ADDWF	REMB2
	MOVF	BARGB1 , W
	BTFSC	_C
	INCF	BARGB1 , W
	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C

```

                INCF          BARGB0,W
                ADDWF         REMB0
UOK44L
                endm
UDIV2323L      macro
;      Max Timing:      13+6*22+21+21+6*22+21+21+6*22+21+12 = 526 clks
;      Min Timing:      13+6*21+20+20+6*21+20+20+6*21+20+3 = 494 clks
;      PM: 11+3*51+31+12 = 207                                DM: 12
                MOVF         BARGB2,W
                SUBWF        REMB2
                MOVF         BARGB1,W
                BTFSS        _C
                INCFSZ       BARGB1,W
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCFSZ       BARGB0,W
                SUBWF        REMB0
                RLF          ACCB0
                MOVLW        7
                MOVWF        LOOPCOUNT
LOOPU2323A     RLF          ACCB0,W
                RLF          REMB2
                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB2,W
                BTFSS        ACCB0,LSB
                GOTO         UADD33LA
                SUBWF        REMB2
                MOVF         BARGB1,W
                BTFSS        _C
                INCFSZ       BARGB1,W
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCFSZ       BARGB0,W
                SUBWF        REMB0
                GOTO         UOK33LA
UADD33LA      ADDWF         REMB2
                MOVF         BARGB1,W
                BTFSC        _C
                INCFSZ       BARGB1,W
                ADDWF        REMB1
                MOVF         BARGB0,W
                BTFSC        _C
                INCFSZ       BARGB0,W
                ADDWF        REMB0
UOK33LA      RLF          ACCB0
                DECFSZ       LOOPCOUNT
                GOTO         LOOPU2323A
                RLF          ACCB1,W
                RLF          REMB2
                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB2,W
                BTFSS        ACCB0,LSB
                GOTO         UADD33L8
                SUBWF        REMB2
                MOVF         BARGB1,W
                BTFSS        _C
                INCFSZ       BARGB1,W
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCFSZ       BARGB0,W

```

AN617

	SUBWF	REMB0
	GOTO	UOK33L8
UADD33L8	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSZ	BARGB0 ,W
	ADDWF	REMB0
UOK33L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU2323B	RLF	ACCB1 ,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 ,W
	BTFSS	ACCB1 ,LSB
	GOTO	UADD33LB
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSZ	BARGB0 ,W
	SUBWF	REMB0
	GOTO	UOK33LB
UADD33LB	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSZ	BARGB1 ,W
	ADDWF	REMB1
	MOVF	BARGB0 ,W
	BTFSC	_C
	INCFSZ	BARGB0 ,W
	ADDWF	REMB0
UOK33LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU2323B
	RLF	ACCB2 ,W
	RLF	REMB2
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB2 ,W
	BTFSS	ACCB1 ,LSB
	GOTO	UADD33L16
	SUBWF	REMB2
	MOVF	BARGB1 ,W
	BTFSS	_C
	INCFSZ	BARGB1 ,W
	SUBWF	REMB1
	MOVF	BARGB0 ,W
	BTFSS	_C
	INCFSZ	BARGB0 ,W
	SUBWF	REMB0
	GOTO	UOK33L16
UADD33L16	ADDWF	REMB2
	MOVF	BARGB1 ,W
	BTFSC	_C
	INCFSZ	BARGB1 ,W


```

                ADDWF      REMB1
                MOVF       BARGB0,W
                BTFSC     _C
                INCFSZ    BARGB0,W
                ADDWF     REMB0

UOK33L16      RLF        ACCB2
                MOVLW     7
                MOVWF     LOOPCOUNT
LOOPU2323C    RLF        ACCB2,W
                RLF       REMB2
                RLF       REMB1
                RLF       REMB0
                MOVF      BARGB2,W
                BTFSS     ACCB2,LSB
                GOTO      UADD33LC
                SUBWF     REMB2
                MOVF      BARGB1,W
                BTFSS     _C
                INCFSZ    BARGB1,W
                SUBWF     REMB1
                MOVF      BARGB0,W
                BTFSS     _C
                INCFSZ    BARGB0,W
                SUBWF     REMB0
                GOTO      UOK33LC
UADD33LC      ADDWF     REMB2
                MOVF      BARGB1,W
                BTFSC     _C
                INCFSZ    BARGB1,W
                ADDWF     REMB1
                MOVF      BARGB0,W
                BTFSC     _C
                INCFSZ    BARGB0,W
                ADDWF     REMB0

UOK33LC      RLF        ACCB2
                DECFSZ    LOOPCOUNT
                GOTO      LOOPU2323C
                BTFSC     ACCB2,LSB
                GOTO      UOK33L
                MOVF      BARGB2,W
                ADDWF     REMB2
                MOVF      BARGB1,W
                BTFSC     _C
                INCF      BARGB1,W
                ADDWF     REMB1
                MOVF      BARGB0,W
                BTFSC     _C
                INCF      BARGB0,W
                ADDWF     REMB0

UOK33L      endm
;*****
;*****
;
;   24/24 Bit Signed Fixed Point Divide 24/24 -> 24.24
;   Input:  24 bit fixed point dividend in AARGB0, AARGB1,AARGB2
;           24 bit fixed point divisor in BARGB0, BARGB1, BARGB2
;   Use:    CALL    FXD2424S
;   Output: 24 bit fixed point quotient in AARGB0, AARGB1,AARGB2
;           24 bit fixed point remainder in REMB0, REMB1, REMB2
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:      12+526+3 = 541 clks           A > 0, B > 0
;                   19+526+20 = 565 clks          A > 0, B < 0
;                   19+526+20 = 565 clks          A < 0, B > 0

```

AN617

```
;          26+526+3 = 555 clks          A < 0, B < 0
;          Min Timing: 12+494+3 = 509 clks      A > 0, B > 0
;          19+494+20 = 533 clks          A > 0, B < 0
;          19+494+20 = 533 clks          A < 0, B > 0
;          26+494+3 = 523 clks          A < 0, B < 0
;          PM: 26+207+20 = 253          DM: 12
FXD2424S   MOVF          AARGB0,W
           XORWF        BARGB0,W
           MOVWF        SIGN
           BTFSS        BARGB0,MSB          ; if MSB set, negate BARG
           GOTO         CA2424S
           COMF         BARGB2
           INCF         BARGB2
           BTFSC        _Z
           DECF         BARGB1
           COMF         BARGB1
           BTFSC        _Z
           DECF         BARGB0
           COMF         BARGB0
CA2424S    BTFSS        AARGB0,MSB          ; if MSB set, negate AARG
           GOTO         C2424S
           COMF         AARGB2
           INCF         AARGB2
           BTFSC        _Z
           DECF         AARGB1
           COMF         AARGB1
           BTFSC        _Z
           DECF         AARGB0
           COMF         AARGB0
C2424S     CLRF         REMB0
           CLRF         REMB1
           CLRF         REMB2
           SDIV2424L
           BTFSS        SIGN,MSB
           RETLW        0x00
           COMF         AARGB2
           INCF         AARGB2
           BTFSC        _Z
           DECF         AARGB1
           COMF         AARGB1
           BTFSC        _Z
           DECF         AARGB0
           COMF         AARGB0
           COMF         REMB2
           INCF         REMB2
           BTFSC        _Z
           DECF         REMB1
           COMF         REMB1
           BTFSC        _Z
           DECF         REMB0
           COMF         REMB0
           RETLW        0x00
;*****
;*****
;          24/24 Bit Unsigned Fixed Point Divide 24/24 -> 24.24
;          Input:  24 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2
;          24 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
;          Use:    CALL    FXD2424U
;          Output: 24 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2
;          24 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;          Result: AARG, REM  <--  AARG / BARG
;          Max Timing:  3+671+2 = 676 clks
;          Max Timing:  3+639+2 = 644 clks
;          PM: 3+222+1 = 226          DM: 13
FXD2424U   CLRF         REMB0
```

```

                CLRF          REMB1
                CLRF          REMB2
                UDIV2424L
                RETLW         0x00
;*****
;*****
;
;      23/23 Bit Unsigned Fixed Point Divide 23/23 -> 23.23
;      Input:  23 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2
;              23 bit unsigned fixed point divisor in BARGB0, BARGB1, BARBB2
;      Use:    CALL    FXD2323U
;      Output: 23 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2
;              23 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:  3+526+2 = 531 clks
;      Min Timing:  3+494+2 = 499 clks
;      PM: 3+207+1 = 211          DM: 12
FXD2323U        CLRF          REMB0
                CLRF          REMB1
                CLRF          REMB2
                UDIV2323L
                RETLW         0x00
;*****
;*****
                END

```

D.5 24/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines

```

;      24/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
;      Input:  fixed point arguments in AARG and BARG
;      Output: quotient AARG/BARG followed by remainder in REM
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed divide application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;
;      Routine      Clocks      Function
;      FXD2416S     438         24 bit/16 bit -> 24.16 signed fixed point divide
;      FXD2416U     529         24 bit/16 bit -> 24.16 unsigned fixed point divide
;      FXD2315U     407         23 bit/15 bit -> 23.15 unsigned fixed point divide
;
;      list      r=dec,x=on,t=off
;      include <PIC16.INC>      ; general PIC16 definitions
;      include <MATH16.INC>     ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI        equ      0x1E      ; random number senerator registers
RANDLO        equ      0x1F
DATA          equ      0x20
;*****
;*****
;      Test suite for 24/16 bit fixed point divide algorithms
                org          0x0005
MAIN          MOVLW         RAMSTART
                MOVWF        FSR
MEMLOOP       CLRF          INDF
                INCF         FSR
                MOVLW         RAMSTOP
                SUBWF        FSR,W
                BTFSS        _Z
                GOTO         MEMLOOP
                MOVLW         0x45          ; seed for random numbers
                MOVWF        RANDLO
                MOVLW         0x30
                MOVWF        RANDHI
                MOVLW         DATA
                MOVWF        FSR

```

AN617

```
BCF          _RP0
BCF          _RP1
BCF          _IRP
CALL         RANDOM16
MOVF        RANDHI , W
MOVWF       BARGB0
BCF         BARGB0 , MSB
MOVF        BARGB0 , W
MOVWF       INDF
INCF        FSR
MOVF        RANDLO , W
MOVWF       BARGB1
MOVWF       INDF
INCF        FSR
CALL         RANDOM16
MOVF        RANDHI , W
MOVWF       AARGB0
BCF         AARGB0 , MSB
MOVF        AARGB0 , W
MOVWF       INDF
INCF        FSR
MOVF        RANDLO , W
MOVWF       AARGB1
MOVWF       INDF
INCF        FSR
CALL         RANDOM16

MOVF        RANDHI , W
MOVWF       AARGB2
MOVWF       INDF
INCF        FSR
CALL         FXD2315U
MOVF        AARGB0 , W
MOVWF       INDF
INCF        FSR
MOVF        AARGB1 , W
MOVWF       INDF
INCF        FSR
MOVF        AARGB2 , W
MOVWF       INDF
INCF        FSR
MOVF        REMB0 , W
MOVWF       INDF
INCF        FSR
MOVF        REMB1 , W
MOVWF       INDF
INCF        FSR
SELF        GOTO     SELF
RANDOM16     RLF         RANDHI , W          ; random number generator
XORWF       RANDHI , W
MOVWF       TEMPB0
SWAPF      RANDHI
SWAPF      RANDLO , W
MOVWF       TEMPB1
RLF         TEMPB1 , W
RLF         TEMPB1
MOVF        TEMPB1 , W
XORWF       RANDHI , W
SWAPF      RANDHI
ANDLW      0x01
RLF         TEMPB0
RLF         RANDLO
XORWF       RANDLO
RLF         RANDHI

RETLW      0
```

```

;*****
;*****
;      24/16 Bit Division Macros
SDIV2416L      macro
;      Max Timing:      9+6*17+16+16+6*17+16+16+6*17+16+8 = 403 clks
;      Min Timing:      9+6*16+15+15+6*16+15+15+6*16+15+3 = 375 clks
;      PM: 7+2*40+22+8 = 117                                DM: 7

                MOVF          BARGB1,W
                SUBWF         REMB1
                MOVF          BARGB0,W
                BTFSS         _C
                INCF          BARGB0,W
                SUBWF         REMB0
                RLF           ACCB0
                MOVLW         7
                MOVWF         LOOPCOUNT
LOOPS2416A     RLF           ACCB0,W
                RLF           REMB1
                RLF           REMB0
                MOVF          BARGB1,W
                BTFSS         ACCB0,LSB
                GOTO          SADD46LA
                SUBWF         REMB1
                MOVF          BARGB0,W
                BTFSS         _C
                INCF          BARGB0,W
                SUBWF         REMB0
                GOTO          SOK46LA
SADD46LA      ADDWF          REMB1
                MOVF          BARGB0,W
                BTFSC         _C
                INCF          BARGB0,W
                ADDWF         REMB0

SOK46LA      RLF           ACCB0
                DECFSZ        LOOPCOUNT
                GOTO          LOOPS2416A
                RLF          ACCB1,W
                RLF          REMB1
                RLF          REMB0
                MOVF          BARGB1,W
                BTFSS         ACCB0,LSB
                GOTO          SADD46L8
                SUBWF         REMB1
                MOVF          BARGB0,W
                BTFSS         _C
                INCF          BARGB0,W
                SUBWF         REMB0
                GOTO          SOK46L8
SADD46L8     ADDWF          REMB1
                MOVF          BARGB0,W
                BTFSC         _C
                INCF          BARGB0,W
                ADDWF         REMB0

SOK46L8      RLF           ACCB1
                MOVLW         7
                MOVWF         LOOPCOUNT
LOOPS2416B   RLF           ACCB1,W
                RLF           REMB1
                RLF           REMB0
                MOVF          BARGB1,W
                BTFSS         ACCB1,LSB
                GOTO          SADD46LB
                SUBWF         REMB1
                MOVF          BARGB0,W

```

AN617

```

                BTFSS          _C
                INCFSZ        BARGB0 , W
                SUBWF         REMB0
                GOTO          SOK46LB
SADD46LB      ADDWF          REMB1
                MOVF          BARGB0 , W
                BTFSC         _C
                INCFSZ        BARGB0 , W
                ADDWF         REMB0

SOK46LB      RLF             ACCB1
                DECFSZ        LOOPCOUNT
                GOTO          LOOPS2416B
                RLF             ACCB2 , W
                RLF             REMB1
                RLF             REMB0
                MOVF          BARGB1 , W
                BTFSS        ACCB1 , LSB
                GOTO          SADD46L16
                SUBWF         REMB1
                MOVF          BARGB0 , W
                BTFSS         _C
                INCFSZ        BARGB0 , W
                SUBWF         REMB0
                GOTO          SOK46L16
SADD46L16    ADDWF          REMB1
                MOVF          BARGB0 , W
                BTFSC         _C
                INCFSZ        BARGB0 , W
                ADDWF         REMB0

SOK46L16     RLF             ACCB2
                MOVLW         7
                MOVWF         LOOPCOUNT
LOOPS2416C   RLF             ACCB2 , W
                RLF             REMB1
                RLF             REMB0
                MOVF          BARGB1 , W
                BTFSS        ACCB2 , LSB
                GOTO          SADD46LC
                SUBWF         REMB1
                MOVF          BARGB0 , W
                BTFSS         _C
                INCFSZ        BARGB0 , W
                SUBWF         REMB0
                GOTO          SOK46LC
SADD46LC     ADDWF          REMB1
                MOVF          BARGB0 , W
                BTFSC         _C
                INCFSZ        BARGB0 , W
                ADDWF         REMB0

SOK46LC      RLF             ACCB2
                DECFSZ        LOOPCOUNT
                GOTO          LOOPS2416C
                BTFSC        ACCB2 , LSB
                GOTO          SOK46L
                MOVF          BARGB1 , W
                ADDWF         REMB1
                MOVF          BARGB0 , W
                BTFSC         _C
                INCF          BARGB0 , W
                ADDWF         REMB0

SOK46L
                endm
UDIV2416L   macro
```

```

;      Max Timing:      16+6*22+21+21+6*22+21+21+6*22+21+8 = 525 clks
;      Min Timing:      16+6*21+20+20+6*21+20+20+6*21+20+3 = 497 clks
;      PM: 14+31+27+31+27+31+8 = 169                               DM: 8
      CLRF          TEMP
      RLF          ACCB0,W
      RLF          REMB1
      MOVF         BARGB1,W
      SUBWF        REMB1
      MOVF         BARGB0,W
      BTFSS        _C
      INCFSZ       BARGB0,W
      SUBWF        REMB0
      CLRW
      BTFSS        _C
      MOVLW        1
      SUBWF        TEMP
      RLF          ACCB0
      MOVLW        7
      MOVWF        LOOPCOUNT
LOOPU2416A  RLF          ACCB0,W
           RLF          REMB1
           RLF          REMB0
           RLF          TEMP
           MOVF         BARGB1,W
           BTFSS        ACCB0,LSB
           GOTO         UADD46LA
           SUBWF        REMB1
           MOVF         BARGB0,W
           BTFSS        _C
           INCFSZ       BARGB0,W
           SUBWF        REMB0
           CLRW
           BTFSS        _C
           MOVLW        1
           SUBWF        TEMP
           GOTO         UOK46LA
UADD46LA   ADDWF        REMB1
           MOVF         BARGB0,W
           BTFSC        _C
           INCFSZ       BARGB0,W
           ADDWF        REMB0
           CLRW
           BTFSC        _C
           MOVLW        1
           ADDWF        TEMP
           GOTO         UOK46LA
UOK46LA   RLF          ACCB0
           DECFSZ       LOOPCOUNT
           GOTO         LOOPU2416A
           RLF          ACCB1,W
           RLF          REMB1
           RLF          REMB0
           RLF          TEMP
           MOVF         BARGB1,W
           BTFSS        ACCB0,LSB
           GOTO         UADD46L8
           SUBWF        REMB1
           MOVF         BARGB0,W
           BTFSS        _C
           INCFSZ       BARGB0,W
           SUBWF        REMB0
           CLRW
           BTFSS        _C
           MOVLW        1
           SUBWF        TEMP
           GOTO         UOK46L8

```

AN617

UADD46L8	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK46L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU2416B	RLF	ACCB1 , W
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB1 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD46LB
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK46LB
UADD46LB	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	
	BTFSC	_C
	MOVLW	1
	ADDWF	TEMP
UOK46LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU2416B
	RLF	ACCB2 , W
	RLF	REMB1
	RLF	REMB0
	RLF	TEMP
	MOVF	BARGB1 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD46L16
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	CLRW	
	BTFSS	_C
	MOVLW	1
	SUBWF	TEMP
	GOTO	UOK46L16
UADD46L16	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
	CLRW	


```

                BTFSC      _C
                MOVLW     1
                ADDWF     TEMP

UOK46L16      RLF        ACCB2
                MOVLW     7
                MOVWF     LOOPCOUNT
LOOPU2416C    RLF        ACCB2,W
                RLF        REMB1
                RLF        REMB0
                RLF        TEMP
                MOVF      BARGB1,W
                BTFSS     ACCB2,LSB
                GOTO      UADD46LC
                SUBWF     REMB1
                MOVF      BARGB0,W
                BTFSS     _C
                INCF      BARGB0,W
                SUBWF     REMB0
                CLRW
                BTFSS     _C
                MOVLW     1
                SUBWF     TEMP
                GOTO      UOK46LC
UADD46LC      ADDWF     REMB1
                MOVF      BARGB0,W
                BTFSC     _C
                INCF      BARGB0,W
                ADDWF     REMB0
                CLRW
                BTFSC     _C
                MOVLW     1
                ADDWF     TEMP

UOK46LC      RLF        ACCB2
                DECFSZ    LOOPCOUNT
                GOTO      LOOPU2416C
                BTFSC     ACCB2,LSB
                GOTO      UOK46L
                MOVF      BARGB1,W
                ADDWF     REMB1
                MOVF      BARGB0,W
                BTFSC     _C
                INCF      BARGB0,W
                ADDWF     REMB0

UOK46L      endm
UDIV2315L    macro
;           Max Timing:    9+6*17+16+16+6*17+16+16+6*17+16+8 = 403 clks
;           Min Timing:    9+6*16+15+15+6*16+15+15+6*16+15+3 = 375 clks
;           PM: 7+2*40+22+8 = 117                                     DM: 7
                MOVF      BARGB1,W
                SUBWF     REMB1
                MOVF      BARGB0,W
                BTFSS     _C
                INCF      BARGB0,W
                SUBWF     REMB0
                RLF        ACCB0
                MOVLW     7
                MOVWF     LOOPCOUNT
LOOPU2315A    RLF        ACCB0,W
                RLF        REMB1
                RLF        REMB0
                MOVF      BARGB1,W
                BTFSS     ACCB0,LSB
                GOTO      UADD35LA

```

AN617

	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK35LA
UADD35LA	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK35LA	RLF	ACCB0
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU2315A
	RLF	ACCB1 , W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1 , W
	BTFSS	ACCB0 , LSB
	GOTO	UADD35L8
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK35L8
UADD35L8	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK35L8	RLF	ACCB1
	MOVLW	7
	MOVWF	LOOPCOUNT
LOOPU2315B	RLF	ACCB1 , W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD35LB
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C
	INCFSZ	BARGB0 , W
	SUBWF	REMB0
	GOTO	UOK35LB
UADD35LB	ADDWF	REMB1
	MOVF	BARGB0 , W
	BTFSC	_C
	INCFSZ	BARGB0 , W
	ADDWF	REMB0
UOK35LB	RLF	ACCB1
	DECFSZ	LOOPCOUNT
	GOTO	LOOPU2315B
	RLF	ACCB2 , W
	RLF	REMB1
	RLF	REMB0
	MOVF	BARGB1 , W
	BTFSS	ACCB1 , LSB
	GOTO	UADD35L16
	SUBWF	REMB1
	MOVF	BARGB0 , W
	BTFSS	_C

```

                INCFSZ      BARGB0,W
                SUBWF      REMB0
UADD35L16      GOTO      UOK35L16
                ADDWF      REMB1
                MOVF      BARGB0,W
                BTFSC     _C
                INCFSZ     BARGB0,W
                ADDWF      REMB0

UOK35L16      RLF      ACCB2
                MOVLW     7
                MOVWF     LOOPCOUNT
LOOPU2315C    RLF      ACCB2,W
                RLF      REMB1
                RLF      REMB0
                MOVF      BARGB1,W
                BTFSS     ACCB2,LSB
                GOTO      UADD35LC
                SUBWF     REMB1
                MOVF      BARGB0,W
                BTFSS     _C
                INCFSZ     BARGB0,W
                SUBWF     REMB0
                GOTO      UOK35LC
UADD35LC      ADDWF      REMB1
                MOVF      BARGB0,W
                BTFSC     _C
                INCFSZ     BARGB0,W
                ADDWF      REMB0

UOK35LC      RLF      ACCB2
                DECFSZ    LOOPCOUNT
                GOTO      LOOPU2315C
                BTFSC     ACCB2,LSB
                GOTO      UOK35L
                MOVF      BARGB1,W
                ADDWF     REMB1
                MOVF      BARGB0,W
                BTFSC     _C
                INCF      BARGB0,W
                ADDWF     REMB0

UOK35L
                endm
;*****
;*****
;
;      24/16 Bit Signed Fixed Point Divide 24/16 -> 24.16
;      Input:  24 bit fixed point dividend in AARGB0, AARGB1,AARGB2
;              16 bit fixed point divisor in BARGB0, BARGB1
;      Use:    CALL    FXD2416S
;      Output: 24 bit fixed point quotient in AARGB0, AARGB1,AARGB2
;              16 bit fixed point remainder in REMB0, REMB1
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:      11+403+3 = 417 clks          A > 0, B > 0
;                      15+403+17 = 435 clks         A > 0, B < 0
;                      18+403+17 = 438 clks         A < 0, B > 0
;                      22+403+3 = 428 clks          A < 0, B < 0
;      Min Timing:      11+375+3 = 389 clks         A > 0, B > 0
;                      15+375+17 = 407 clks         A > 0, B < 0
;                      18+375+17 = 410 clks         A < 0, B > 0
;                      22+375+3 = 400 clks          A < 0, B < 0
;      PM: 22+117+16 = 140                          DM: 8
FXD2416S      MOVF      AARGB0,W
                XORWF      BARGB0,W
                MOVWF     SIGN
                BTFSS     BARGB0,MSB          ; if MSB set, negate BARG

```

AN617

```
GOTO          CA2416S
COMF          BARGB1
INCF          BARGB1
BTFSC        _Z
DECF         BARGB0
COMF         BARGB0
CA2416S      BTFSS        AARGB0,MSB          ; if MSB set, negate AARG
GOTO          C2416S
COMF         AARGB2
INCF         AARGB2
BTFSC        _Z
DECF         AARGB1
COMF         AARGB1
BTFSC        _Z
DECF         AARGB0
COMF         AARGB0
C2416S      CLRF          REMB0
CLRF          REMB1
SDIV2416L
BTFSS        SIGN,MSB
RETLW        0x00
COMF         AARGB2
INCF         AARGB2
BTFSC        _Z
DECF         AARGB1
COMF         AARGB1
BTFSC        _Z
DECF         AARGB0
COMF         AARGB0
COMF         REMB1
INCF         REMB1
BTFSC        _Z
DECF         REMB0
COMF         REMB0
RETLW        0x00
;*****
;*****
;
; 24/16 Bit Unsigned Fixed Point Divide 24/16 -> 24.16
; Input:  24 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2
;         16 bit unsigned fixed point divisor in BARGB0, BARGB1
; Use:    CALL    FXD2416U
; Output: 24 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2
;         16 bit unsigned fixed point remainder in REMB0, REMB1
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 2+525+2 = 529 clks
; Max Timing: 2+497+2 = 501 clks
; PM: 2+169+1 = 172          DM: 8
FXD2416U     CLRF          REMB0
CLRF          REMB1
UDIV2416L
RETLW        0x00
;*****
;*****
;
; 23/15 Bit Unsigned Fixed Point Divide 23/15 -> 23.15
; Input:  23 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2
;         15 bit unsigned fixed point divisor in BARGB0, BARGB1
; Use:    CALL    FXD2315U
; Output: 23 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2
;         15 bit unsigned fixed point remainder in REMB0, REMB1
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 2+403+2 = 407 clks
; Min Timing: 2+375+2 = 379 clks
; PM: 2+117+1 = 120          DM: 7
FXD2315U     CLRF          REMB0
```

```

        CLRF          REMB1
        UDIV2315L
        RETLW        0x00
;*****
;*****
        END

```

D.6 16/16 PIC16C5X/PIC16CXX Fixed Point Divide Routines

```

;      16/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.7
;      Input:  fixed point arguments in AARG and BARG
;      Output: quotient AARG/BARG followed by remainder in REM
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed divide application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;      Routine      Clocks      Function
;      FXD1616S    319 16 bit/16 bit -> 16.16 signed fixed point divide
;      FXD1616U    373 16 bit/16 bit -> 16.16 unsigned fixed point divide
;      FXD1515U    294 15 bit/15 bit -> 15.15 unsigned fixed point divide
;      The above timings are based on the looped macros. If space permits,
;      approximately 65-69 clocks can be saved by using the unrolled macros.
;      list      r=dec,x=on,t=off
;      include <PIC16.INC>      ; general PIC16 definitions
;      include <MATH16.INC>     ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI      equ      0x1A      ; random number senerator registers
RANDLO      equ      0x1B
TESTCOUNT  equ      0x20      ; counter
DATA        equ      0x21      ; beginning of test data
;*****
;*****
;      Test suite for 16/16 bit fixed point divide algorithms
;      org          0x0005
MAIN        MOV LW      RAMSTART
            MOV WF      FSR
MEMLOOP     CLRF        INDF
            INCF        FSR
            MOV LW      RAMSTOP
            SUB WF      FSR,W
            BTFSS       _Z
            GOTO        MEMLOOP
            MOV LW      0x45          ; seed for random numbers
            MOV WF      RANDLO
            MOV LW      0x30
            MOV WF      RANDHI
            MOV LW      DATA
            MOV WF      FSR
            BCF         _RP0
            BCF         _RP1
            BCF         _IRP
            CALL        TFXD1616
            MOV LW      0x99
            MOV WF      AARGB0
            MOV LW      0xAB
            MOV WF      AARGB1
            MOV LW      0x00
            MOV WF      BARGB0
            MOV LW      0xFF
            MOV WF      BARGB1
            CALL        FXD1616S
SELF        GOTO        SELF
RANDOM16     RLF         RANDHI,W      ; random number generator
            XOR WF      RANDHI,W

```

AN617

```

MOVWF    TEMPB0
SWAPF    RANDHI
SWAPF    RANDLO,W
MOVWF    TEMPB1
RLF      TEMPB1,W
RLF      TEMPB1
MOVF     TEMPB1,W
XORWF    RANDHI,W
SWAPF    RANDHI
ANDLW    0x01
RLF      TEMPB0
RLF      RANDLO
XORWF    RANDLO
RLF      RANDHI

RETLW    0
;      Test suite for FXD1616
TFXD1616    MOVLW    2
MOVWF    TESTCOUNT

D1616LOOP
CALL     RANDOM16
MOVF    RANDHI,W
MOVWF   BARGB0
;      BCF     BARGB0,MSB
;      MOVF   BARGB0,W
MOVWF   INDF
INCF    FSR
MOVF    RANDLO,W
MOVWF   BARGB1
MOVWF   INDF
INCF    FSR
CALL     RANDOM16
MOVF    RANDHI,W
MOVWF   AARGB0
;      BCF     AARGB0,MSB
;      MOVF   AARGB0,W
MOVWF   INDF
INCF    FSR
MOVF    RANDLO,W
MOVWF   AARGB1
MOVWF   INDF
INCF    FSR
CALL     FXD1616S
MOVF    AARGB0,W
MOVWF   INDF
INCF    FSR
MOVF    AARGB1,W
MOVWF   INDF
INCF    FSR
MOVF    REMB0,W
MOVWF   INDF
INCF    FSR
MOVF    REMB1,W
MOVWF   INDF
INCF    FSR
DECFSZ  TESTCOUNT
GOTO    D1616LOOP
RETLW   0x00
;*****
;*****
;      16/16 Bit Division Macros
SDIV1616L    macro
;      Max Timing:    13+14*18+17+8 = 290 clks
;      Min Timing:    13+14*16+15+3 = 255 clks
;      PM: 42          DM: 7
RLF      ACCB0,W

```

```

                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB1,W
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                RLF          ACCB1
                RLF          ACCB0
                MOVLW        15
                MOVWF        LOOPCOUNT
LOOPS1616      RLF          ACCB0,W
                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB1,W
                BTFSS        ACCB1,LSB
                GOTO         SADD66L
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                GOTO         SOK66LL
SADD66L       ADDWF         REMB1
                MOVF         BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
SOK66LL      RLF          ACCB1
                RLF          ACCB0
                DECF        LOOPCOUNT
                GOTO         LOOPS1616
                BTFSC        ACCB1,LSB
                GOTO         SOK66L
                MOVF         BARGB1,W
                ADDWF        REMB1
                MOVF         BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
SOK66L
                endm
UDIV1616L    macro
;           restore = 23 clks, nonrestore = 17 clks
;           Max Timing:    2+15*23+22 = 369 clks
;           Min Timing:    2+15*17+16 = 273 clks
;           PM: 24
;
                MOVLW        16
                MOVWF        LOOPCOUNT
LOOPU1616    RLF          ACCB0,W
                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB1,W
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                BTFSC        _C
                GOTO         UOK66LL
                MOVF         BARGB1,W
                ADDWF        REMB1
                MOVF         BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W

```

AN617

```

                ADDWF          REMB0
                BCF            _C
UOK66LL        RLF            ACCB1
                RLF            ACCB0
                DECFSZ        LOOPCOUNT
                GOTO          LOOPU1616
                endm
UDIV1515L     macro
;             Max Timing:      13+14*18+17+8 = 290 clks
;             Min Timing:      13+14*17+16+3 = 270 clks
;             PM: 42                                DM: 7
                RLF            ACCB0,W
                RLF            REMB1
                RLF            REMB0
                MOVF          BARGB1,W
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                RLF            ACCB1
                RLF            ACCB0
                MOVLW         15
                MOVWF        LOOPCOUNT
LOOPU1515     RLF            ACCB0,W
                RLF            REMB1
                RLF            REMB0
                MOVF          BARGB1,W
                BTFSS        ACCB1,LSB
                GOTO          UADD55L
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                GOTO          UOK55LL
UADD55L      ADDWF          REMB1
                MOVF          BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
UOK55LL     RLF            ACCB1
                RLF            ACCB0
                DECFSZ        LOOPCOUNT
                GOTO          LOOPU1515
                BTFSC        ACCB1,LSB
                GOTO          UOK55L
                MOVF          BARGB1,W
                ADDWF        REMB1
                MOVF          BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
UOK55L      endm
SDIV1616     macro
;             Max Timing:      7+10+6*14+14+7*14+8 = 221 clks
;             Min Timing:      7+10+6*13+13+7*13+3 = 202 clks
;             PM: 7+10+6*18+18+7*18+8 = 277    DM: 6
                variable i
                MOVF          BARGB1,W
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0

```



```

                RLF          ACCB0
                RLF          ACCB0,W
                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB1,W
                ADDWF        REMB1
                MOVF         BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
                RLF          ACCB0
                i = 2
                while i < 8
                RLF          ACCB0,W
                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB1,W
                BTFSS        ACCB0,LSB
                GOTO         SADD66#v(i)
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                GOTO         SOK66#v(i)
SADD66#v(i)    ADDWF        REMB1
                MOVF         BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
SOK66#v(i)    RLF          ACCB0
                i=i+1
                endw
                RLF          ACCB1,W
                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB1,W
                BTFSS        ACCB0,LSB
                GOTO         SADD668
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                GOTO         SOK668
SADD668      ADDWF        REMB1
                MOVF         BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
SOK668      RLF          ACCB1
                i = 9
                while i < 16
                RLF          ACCB1,W
                RLF          REMB1
                RLF          REMB0
                MOVF         BARGB1,W
                BTFSS        ACCB1,LSB
                GOTO         SADD66#v(i)
                SUBWF        REMB1
                MOVF         BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                GOTO         SOK66#v(i)
SADD66#v(i)  ADDWF        REMB1

```

AN617

```
                MOVF          BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
SOK66#v(i)     RLF          ACCB1
                i=i+1
                endw
                BTFSC        ACCB1,LSB
                GOTO        SOK66
                MOVF          BARGB1,W
                ADDWF        REMB1
                MOVF          BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
SOK66
                endm
UDIV1616      macro
;           restore = 20 clks, nonrestore = 14 clks
;           Max Timing: 16*20 = 320 clks
;           Min Timing: 16*14 = 224 clks
;           PM: 16*20 = 320          DM: 6
                variable      i
                i = 0
                while i < 16
                RLF          ACCB0,W
                RLF          REMB1
                RLF          REMB0
                MOVF          BARGB1,W
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                BTFSC        _C
                GOTO        UOK66#v(i)
                MOVF          BARGB1,W
                ADDWF        REMB1
                MOVF          BARGB0,W
                BTFSC        _C
                INCF        BARGB0,W
                ADDWF        REMB0
                BCF          _C
UOK66#v(i)    RLF          ACCB1
                RLF          ACCB0
                i=i+1
                endw
                endm
UDIV1515      macro
;           Max Timing:      7+10+6*14+14+7*14+8 = 221 clks
;           Min Timing:      7+10+6*13+13+7*13+3 = 202 clks
;           PM:      7+10+6*18+18+7*18+8 = 277          DM: 6
                variable      i
                MOVF          BARGB1,W
                SUBWF        REMB1
                MOVF          BARGB0,W
                BTFSS        _C
                INCF        BARGB0,W
                SUBWF        REMB0
                RLF          ACCB0
                RLF          ACCB0,W
                RLF          REMB1
                RLF          REMB0
                MOVF          BARGB1,W
                ADDWF        REMB1
                MOVF          BARGB0,W
```

```

    BTFSC          _C
    INCFSZ        BARGB0,W
    ADDWF         REMB0
    RLF           ACCB0
    i = 2
    while i < 8
    RLF           ACCB0,W
    RLF           REMB1
    RLF           REMB0
    MOVF         BARGB1,W
    BTFSS        ACCB0,LSB
    GOTO         UADD55#v(i)
    SUBWF        REMB1
    MOVF         BARGB0,W
    BTFSS        _C
    INCFSZ        BARGB0,W
    SUBWF        REMB0
    GOTO         UOK55#v(i)
UADD55#v(i)    ADDWF         REMB1
    MOVF         BARGB0,W
    BTFSC        _C
    INCFSZ        BARGB0,W
    ADDWF        REMB0
UOK55#v(i)    RLF           ACCB0
    i=i+1
    endw
    RLF           ACCB1,W
    RLF           REMB1
    RLF           REMB0
    MOVF         BARGB1,W
    BTFSS        ACCB0,LSB
    GOTO         UADD558
    SUBWF        REMB1
    MOVF         BARGB0,W
    BTFSS        _C
    INCFSZ        BARGB0,W
    SUBWF        REMB0
    GOTO         UOK558
UADD558      ADDWF         REMB1
    MOVF         BARGB0,W
    BTFSC        _C
    INCFSZ        BARGB0,W
    ADDWF        REMB0
UOK558      RLF           ACCB1
    i = 9
    while i < 16
    RLF           ACCB1,W
    RLF           REMB1
    RLF           REMB0
    MOVF         BARGB1,W
    BTFSS        ACCB1,LSB
    GOTO         UADD55#v(i)
    SUBWF        REMB1
    MOVF         BARGB0,W
    BTFSS        _C
    INCFSZ        BARGB0,W
    SUBWF        REMB0
    GOTO         UOK55#v(i)
UADD55#v(i)  ADDWF         REMB1
    MOVF         BARGB0,W
    BTFSC        _C
    INCFSZ        BARGB0,W
    ADDWF        REMB0
UOK55#v(i)  RLF           ACCB1
    i=i+1
    endw

```

AN617

```

        BTFSC          ACCB1,LSB
        GOTO          UOK55
        MOVF          BARGB1,W
        ADDWF        REMB1
        MOVF          BARGB0,W
        BTFSC        _C
        INCF          BARGB0,W
        ADDWF        REMB0
UOK55
        endm
;*****
;*****
;
;      16/16 Bit Signed Fixed Point Divide 16/16 -> 16.16
;      Input:  16 bit fixed point dividend in AARGB0, AARGB1
;              16 bit fixed point divisor in BARGB0, BARGB1
;      Use:    CALL    FXD1616S
;      Output: 16 bit fixed point quotient in AARGB0, AARGB1
;              16 bit fixed point remainder in REMB0, REMB1
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:      11+290+3 = 304 clks           A > 0, B > 0
;                      15+290+14 = 319 clks          A > 0, B < 0
;                      15+290+14 = 319 clks          A < 0, B > 0
;                      19+290+3 = 312 clks           A < 0, B < 0
;      Min Timing:      11+255+3 = 269 clks          A > 0, B > 0
;                      15+255+14 = 284 clks          A > 0, B < 0
;                      15+255+14 = 284 clks          A < 0, B > 0
;                      19+255+3 = 277 clks           A < 0, B < 0
;      PM: 19+42+13 = 74          DM: 8
FXD1616S  MOVF          AARGB0,W
          XORWF        BARGB0,W
          MOVWF       SIGN
          BTFSS        BARGB0,MSB          ; if MSB set, negate BARG
          GOTO          CA1616S
          COMF          BARGB1
          INCF          BARGB1
          BTFSC        _Z
          DECF          BARGB0
          COMF          BARGB0
CA1616S  BTFSS        AARGB0,MSB          ; if MSB set, negate AARG
          GOTO          C1616S
          COMF          AARGB1
          INCF          AARGB1
          BTFSC        _Z
          DECF          AARGB0
          COMF          AARGB0
C1616S  CLRF          REMB0
          CLRF          REMB1
          SDIV1616L
          BTFSS        SIGN,MSB
          RETLW        0x00
          COMF          AARGB1
          INCF          AARGB1
          BTFSC        _Z
          DECF          AARGB0
          COMF          AARGB0
          COMF          REMB1
          INCF          REMB1
          BTFSC        _Z
          DECF          REMB0
          COMF          REMB0
          RETLW        0x00
;*****
;*****
;
;      16/16 Bit Unsigned Fixed Point Divide 16/16 -> 16.16
```

```

;      Input:  16 bit unsigned fixed point dividend in AARGB0, AARGB1
;      16 bit unsigned fixed point divisor in BARGB0, BARGB1
;      Use:    CALL    FXD1616U
;      Output: 16 bit unsigned fixed point quotient in AARGB0, AARGB1
;      16 bit unsigned fixed point remainder in REMB0, REMB1
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:    2+369+2 = 373 clks
;      Min Timing:    2+273+2 = 277 clks
;      PM: 2+24+1 = 27          DM: 7
FXD1616U      CLRF      REMB0
              CLRF      REMB1
              UDIV1616L
              RETLW     0x00
;*****
;*****
;      15/15 Bit Unsigned Fixed Point Divide 15/15 -> 15.15
;      Input:  15 bit unsigned fixed point dividend in AARGB0, AARGB1
;      15 bit unsigned fixed point divisor in BARGB0, BARGB1
;      Use:    CALL    FXD1515U
;      Output: 15 bit unsigned fixed point quotient in AARGB0, AARGB1
;      15 bit unsigned fixed point remainder in REMB0, REMB1
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:    2+290+2 = 294 clks
;      Min Timing:    2+270+2 = 274 clks
;      PM: 2+42+1 = 45          DM: 7
FXD1515U      CLRF      REMB0
              CLRF      REMB1
              UDIV1515L
              RETLW     0x00
;*****
;*****
              END

```

D.7 16/8 PIC16C5X/PIC16CXX Fixed Point Divide Routines

```

;      16/8 PIC16 FIXED POINT DIVIDE ROUTINES  VERSION 1.7
;      Input:  fixed point arguments in AARG and BARG
;      Output: quotient AARG/BARG followed by remainder in REM
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed divide application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;      Routine      Clocks      Function
;      FXD1608S    188 16 bit/8 bit -> 16.08 signed fixed point divide
;      FXD1608U    294 16 bit/8 bit -> 16.08 unsigned fixed point divide
;      FXD1607U    174 16 bit/7 bit -> 16.07 unsigned fixed point divide
;      FXD1507U    166 15 bit/7 bit -> 15.07 unsigned fixed point divide
;      The above timings are based on the looped macros. If space permits,
;      approximately 41-50 clocks can be saved by using the unrolled macros.
;      list      r=dec,x=on,t=off
;      include <PIC16.INC>      ; general PIC16 definitions
;      include <MATH16.INC>     ; PIC16 math library definitions
;*****
;*****
;      Test suite storage
RANDHI      equ      0x1A      ; random number generator registers
RANDLO      equ      0x1B
TESTCOUNT  equ      0x20      ; counter
DATA        equ      0x21      ; beginning of test data
;*****
;*****
;      Test suite for 16/8 bit fixed point divide algorithms
;      org      0x0005
MAIN        MOVLW     RAMSTART
           MOVWF     FSR

```

AN617

```
MEMLOOP      CLRF      INDF
              INCF      FSR
              MOVLW    RAMSTOP
              SUBWF    FSR,W
              BTFSS    _Z
              GOTO     MEMLOOP
              MOVLW    0x45      ; seed for random numbers
              MOVWF    RANDLO
              MOVLW    0x30
              MOVWF    RANDHI
              MOVLW    DATA
              MOVWF    FSR
              BCF      _RP0
              BCF      _RP1
              BCF      _IRP
              CALL     TFXD1608
              MOVLW    0x99
              MOVWF    AARGB0
              MOVLW    0xAB
              MOVWF    AARGB1
              MOVLW    0xFF
              MOVWF    BARGB0
              CALL     FXD1608U
SELF          GOTO     SELF
RANDOM16      RLF      RANDHI,W      ; random number generator
              XORWF    RANDHI,W
              MOVWF    TEMPB0
              SWAPF    RANDHI
              SWAPF    RANDLO,W
              MOVWF    TEMPB1
              RLF      TEMPB1,W
              RLF      TEMPB1
              MOVF     TEMPB1,W
              XORWF    RANDHI,W
              SWAPF    RANDHI
              ANDLW    0x01
              RLF      TEMPB0
              RLF      RANDLO
              XORWF    RANDLO
              RLF      RANDHI

              RETLW    0
;      Test suite for FXD1608
TFXD1608     MOVLW    3
              MOVWF    TESTCOUNT
D1608LOOP    CALL     RANDOM16
              MOVF     RANDHI,W
              MOVWF    BARGB0
;
;      BCF      BARGB0,MSB
              MOVF     BARGB0,W
              MOVWF    INDF
              INCF      FSR
              CALL     RANDOM16
              MOVF     RANDHI,W
              MOVWF    AARGB0
;
;      BCF      AARGB0,MSB
              MOVF     AARGB0,W
              MOVWF    INDF
              INCF      FSR
              MOVF     RANDLO,W
              MOVWF    AARGB1
              MOVWF    INDF
              INCF      FSR
              CALL     FXD1608S
              MOVF     AARGB0,W
```

```

MOVWF      INDF
INCF
MOVF       AARGB1,W
MOVWF      INDF
INCF       FSR
MOVF       REMB0,W
MOVWF      INDF
INCF       FSR
DECFSZ    TESTCOUNT
GOTO      D1608LOOP
RETLW     0x00
;*****
;*****
;      16/08 BIT Division Macros
SDIV1608L macro
;      Max Timing:      3+5+2+5*11+10+10+6*11+10+2 = 163 clks
;      Min Timing:      3+5+2+5*11+10+10+6*11+10+2 = 163 clks
;      PM: 42                      DM: 5
                MOVF      BARGB0,W
                SUBWF    REMB0
                RLF      ACCB0
                RLF      ACCB0,W
                RLF      REMB0
                MOVF     BARGB0,W
                ADDWF    REMB0
                RLF      ACCB0
                MOVLW    6
                MOVWF    LOOPCOUNT
LOOPS1608A      RLF      ACCB0,W
                RLF      REMB0
                MOVF     BARGB0,W
                BTFSC    ACCB0,LSB
                SUBWF    REMB0
                BTFSS    ACCB0,LSB
                ADDWF    REMB0
                RLF      ACCB0
                DECFSZ   LOOPCOUNT
                GOTO     LOOPS1608A
                RLF      ACCB1,W
                RLF      REMB0
                MOVF     BARGB0,W
                BTFSC    ACCB0,LSB
                SUBWF    REMB0
                BTFSS    ACCB0,LSB
                ADDWF    REMB0
                RLF      ACCB1
                MOVLW    7
                MOVWF    LOOPCOUNT
LOOPS1608B      RLF      ACCB1,W
                RLF      REMB0
                MOVF     BARGB0,W
                BTFSC    ACCB1,LSB
                SUBWF    REMB0
                BTFSS    ACCB1,LSB
                ADDWF    REMB0
                RLF      ACCB1
                DECFSZ   LOOPCOUNT
                GOTO     LOOPS1608B
                BTFSS    ACCB1,LSB
                ADDWF    REMB0
                endm
UDIV1608L macro
;      Max Timing: 2+7*12+11+3+7*24+23 = 291 clks
;      Min Timing: 2+7*11+10+3+7*17+16 = 227 clks
;      PM: 39                      DM: 7
                MOVLW    8

```

AN617

```
LOOPU1608A    MOVWF    LOOPCOUNT
              RLF      ACCB0,W
              RLF      REMB0
              MOVF     BARGB0,W
              SUBWF   REMB0
              BTFSC   _C
              GOTO    UOK68A
              ADDWF   REMB0
              BCF     _C
UOK68A        RLF      ACCB0
              DECFSZ  LOOPCOUNT
              GOTO    LOOPU1608A
              CLRF   TEMP
              MOVLW   8
              MOVWF   LOOPCOUNT
LOOPU1608B    RLF      ACCB1,W
              RLF      REMB0
              RLF      TEMP
              MOVF     BARGB0,W
              SUBWF   REMB0
              CLRF   ACCB5
              CLRW
              BTFSS   _C
              INCFSZ  ACCB5,W
              SUBWF   TEMP
              BTFSC   _C
              GOTO    UOK68B
              MOVF     BARGB0,W
              ADDWF   REMB0
              CLRF   ACCB5
              CLRW
              BTFSC   _C
              INCFSZ  ACCB5,W
              ADDWF   TEMP
              BCF     _C
UOK68B        RLF      ACCB1
              DECFSZ  LOOPCOUNT
              GOTO    LOOPU1608B
              endm
UDIV1607L    macro
;           Max Timing:    7+6*11+10+10+6*11+10+2 = 171 clks
;           Min Timing:    7+6*11+10+10+6*11+10+2 = 171 clks
;           PM: 39                                     DM: 5
              RLF      ACCB0,W
              RLF      REMB0
              MOVF     BARGB0,W
              SUBWF   REMB0
              RLF      ACCB0
              MOVLW   7
              MOVWF   LOOPCOUNT
LOOPU1607A    RLF      ACCB0,W
              RLF      REMB0
              MOVF     BARGB0,W
              BTFSC   ACCB0,LSB
              SUBWF   REMB0
              BTFSS   ACCB0,LSB
              ADDWF   REMB0
              RLF      ACCB0
              DECFSZ  LOOPCOUNT
              GOTO    LOOPU1607A
              RLF      ACCB1,W
              RLF      REMB0
              MOVF     BARGB0,W
              BTFSC   ACCB0,LSB
              SUBWF   REMB0
              BTFSS   ACCB0,LSB
```



```

                ADDWF      REMB0
                RLF        ACCB1
                MOVLW      7
                MOVWF      LOOPCOUNT
LOOPU1607B     RLF        ACCB1,W
                RLF        REMB0
                MOVF       BARGB0,W
                BTFSC     ACCB1,LSB
                SUBWF     REMB0
                BTFSS     ACCB1,LSB
                ADDWF     REMB0
                RLF        ACCB1
                DECFSZ    LOOPCOUNT
                GOTO      LOOPU1607B
                BTFSS     ACCB1,LSB
                ADDWF     REMB0
                endm
UDIV1507L     macro
;             Max Timing:  3+5+2+5*11+10+10+6*11+10+2 = 163 clks
;             Min Timing:  3+5+2+5*11+10+10+6*11+10+2 = 163 clks
;             PM:  42                DM:  5
                MOVF       BARGB0,W
                SUBWF     REMB0
                RLF        ACCB0
                RLF        ACCB0,W
                RLF        REMB0
                MOVF       BARGB0,W
                ADDWF     REMB0
                RLF        ACCB0
                MOVLW      6
                MOVWF      LOOPCOUNT
LOOPU1507A     RLF        ACCB0,W
                RLF        REMB0
                MOVF       BARGB0,W
                BTFSC     ACCB0,LSB
                SUBWF     REMB0
                BTFSS     ACCB0,LSB
                ADDWF     REMB0
                RLF        ACCB0
                DECFSZ    LOOPCOUNT
                GOTO      LOOPU1507A
                RLF        ACCB1,W
                RLF        REMB0
                MOVF       BARGB0,W
                BTFSC     ACCB0,LSB
                SUBWF     REMB0
                BTFSS     ACCB0,LSB
                ADDWF     REMB0
                RLF        ACCB1
                MOVLW      7
                MOVWF      LOOPCOUNT
LOOPU1507B     RLF        ACCB1,W
                RLF        REMB0
                MOVF       BARGB0,W
                BTFSC     ACCB1,LSB
                SUBWF     REMB0
                BTFSS     ACCB1,LSB
                ADDWF     REMB0
                RLF        ACCB1
                DECFSZ    LOOPCOUNT
                GOTO      LOOPU1507B
                BTFSS     ACCB1,LSB
                ADDWF     REMB0
                endm
SDIV1608      macro
;             Max Timing:  3+5+14*8+2 = 122 clks

```

AN617

```
;      Min Timing:      3+5+14*8+2 = 122 clks
;      PM: 122                                     DM: 4
      variable i
      MOVF          BARGB0,W
      SUBWF        REMB0
      RLF          ACCB0
      RLF          ACCB0,W
      RLF          REMB0
      MOVF          BARGB0,W
      ADDWF        REMB0
      RLF          ACCB0
      i = 2
      while i < 8
      RLF          ACCB0,W
      RLF          REMB0
      MOVF          BARGB0,W
      BTFSC        ACCB0,LSB
      SUBWF        REMB0
      BTFSS        ACCB0,LSB
      ADDWF        REMB0
      RLF          ACCB0
      i=i+1
      endw
      RLF          ACCB1,W
      RLF          REMB0
      MOVF          BARGB0,W
      BTFSC        ACCB0,LSB
      SUBWF        REMB0
      BTFSS        ACCB0,LSB
      ADDWF        REMB0
      RLF          ACCB1
      i = 9
      while i < 16
      RLF          ACCB1,W
      RLF          REMB0
      MOVF          BARGB0,W
      BTFSC        ACCB1,LSB
      SUBWF        REMB0
      BTFSS        ACCB1,LSB
      ADDWF        REMB0
      RLF          ACCB1
      i=i+1
      endw
      BTFSS        ACCB1,LSB
      ADDWF        REMB0
      endm
UDIV1608      macro
;      restore = 9/21 clks, nonrestore = 8/14 clks
;      Max Timing: 8*9+1+8*21 = 241 clks
;      Min Timing: 8*8+1+8*14 = 177 clks
;      PM: 241                                     DM: 6
      variable          i
      i = 0
      while i < 8
      RLF          ACCB0,W
      RLF          REMB0
      MOVF          BARGB0,W
      SUBWF        REMB0
      BTFSC        _C
      GOTO         UOK68#v(i)
      ADDWF        REMB0
      BCF          _C
UOK68#v(i)    RLF          ACCB0
      i=i+1
      endw
      CLRf        TEMP
```

```

        i = 8
        while i < 16
            RLF          ACCB1,W
            RLF          REMB0
            RLF          TEMP
            MOVF         BARGB0,W
            SUBWF        REMB0
            CLRF         ACCB5
            CLRW
            BTFSS        _C
            INCF        ACCB5,W
            SUBWF        TEMP
            BTFSC        _C
            GOTO         UOK68#v(i)
            MOVF         BARGB0,W
            ADDWF        REMB0
            CLRF         ACCB5
            CLRW
            BTFSC        _C
            INCF        ACCB5,W
            ADDWF        TEMP
            BCF          _C
UOK68#v(i)  RLF          ACCB1
            i=i+1
            endw
        endm
UDIV1607  macro
;          Max Timing:    5+15*8+2 = 127 clks
;          Min Timing:    5+15*8+2 = 127 clks
;          PM: 127
            variable i
            RLF          ACCB0,W
            RLF          REMB0
            MOVF         BARGB0,W
            SUBWF        REMB0
            RLF          ACCB0
            i = 1
            while i < 8
                RLF          ACCB0,W
                RLF          REMB0
                MOVF         BARGB0,W
                BTFSC        ACCB0,LSB
                SUBWF        REMB0
                BTFSS        ACCB0,LSB
                ADDWF        REMB0
                RLF          ACCB0
                i=i+1
            endw
            RLF          ACCB1,W
            RLF          REMB0
            MOVF         BARGB0,W
            BTFSC        ACCB0,LSB
            SUBWF        REMB0
            BTFSS        ACCB0,LSB
            ADDWF        REMB0
            RLF          ACCB1
            i = 9
            while i < 16
                RLF          ACCB1,W
                RLF          REMB0
                MOVF         BARGB0,W
                BTFSC        ACCB1,LSB
                SUBWF        REMB0
                BTFSS        ACCB1,LSB
                ADDWF        REMB0
                RLF          ACCB1

```

DM: 4

AN617

```
        i=i+1
    endw
    BTFSS          ACCB1,LSB
    ADDWF         REMB0
    endm
UDIV1507    macro
;    Max Timing:      3+5+14*8+2 = 122 clks
;    Min Timing:      3+5+14*8+2 = 122 clks
;    PM: 122                                     DM: 4
    variable i
    MOVF          BARGB0,W
    SUBWF         REMB0
    RLF           ACCB0
    RLF           ACCB0,W
    RLF           REMB0
    MOVF          BARGB0,W
    ADDWF         REMB0
    RLF           ACCB0
    i = 2
    while i < 8
    RLF           ACCB0,W
    RLF           REMB0
    MOVF          BARGB0,W
    BTFSC         ACCB0,LSB
    SUBWF         REMB0
    BTFSS         ACCB0,LSB
    ADDWF         REMB0
    RLF           ACCB0
    i=i+1
    endw
    RLF           ACCB1,W
    RLF           REMB0
    MOVF          BARGB0,W
    BTFSC         ACCB0,LSB
    SUBWF         REMB0
    BTFSS         ACCB0,LSB
    ADDWF         REMB0
    RLF           ACCB1
    i = 9
    while i < 16
    RLF           ACCB1,W
    RLF           REMB0
    MOVF          BARGB0,W
    BTFSC         ACCB1,LSB
    SUBWF         REMB0
    BTFSS         ACCB1,LSB
    ADDWF         REMB0
    RLF           ACCB1
    i=i+1
    endw
    BTFSS         ACCB1,LSB
    ADDWF         REMB0
    endm

;*****
;*****

;    16/8 Bit Signed Fixed Point Divide 16/8 -> 16.08
;    Input:  16 bit signed fixed point dividend in AARGB0, AARGB1
;            8 bit signed fixed point divisor in BARGB0
;    Use:    CALL    FXD1608S
;    Output: 16 bit signed fixed point quotient in AARGB0, AARGB1
;            8 bit signed fixed point remainder in REMB0
;    Result: AARG, REM <-- AARG / BARG
;    Max Timing: 10+163+3 = 176 clks          A > 0, B > 0
;               11+163+11 = 185 clks         A > 0, B < 0
```

```

;           14+163+11 = 188 clks           A < 0, B > 0
;           15+163+3  = 181 clks           A < 0, B < 0
;   Min Timing: 10+163+3 = 176 clks       A > 0, B > 0
;           11+163+11 = 185 clks         A > 0, B < 0
;           14+163+11 = 188 clks         A < 0, B > 0
;           15+163+3  = 181 clks         A < 0, B < 0
;   PM: 15+42+10 = 67           DM: 6
FXD1608S   MOVF           AARGB0,W
           XORWF          BARGB0,W
           MOVWF          SIGN
           BTFSS          BARGB0,MSB      ; if MSB set, negate BARG
           GOTO           CA1608S
           COMF           BARGB0
           INCF           BARGB0
CA1608S   BTFSS          AARGB0,MSB      ; if MSB set, negate AARG
           GOTO           C1608S
           COMF           AARGB1
           INCF           AARGB1
           BTFSC          _Z
           DECF           AARGB0
           COMF           AARGB0
C1608S    CLRF           REMB0
           SDIV1608L
           BTFSS          SIGN,MSB
           RETLW          0x00
           COMF           AARGB1
           INCF           AARGB1
           BTFSC          _Z
           DECF           AARGB0
           COMF           AARGB0
           COMF           REMB0
           INCF           REMB0
           RETLW          0x00
;*****
;*****

;   16/8 Bit Unsigned Fixed Point Divide 16/8 -> 16.08
;   Input: 16 bit unsigned fixed point dividend in AARGB0, AARGB1
;           8 bit unsigned fixed point divisor in BARGB0
;   Use:   CALL    FXD1608U
;   Output: 16 bit unsigned fixed point quotient in AARGB0, AARGB1
;           8 bit unsigned fixed point remainder in REMB0
;   Result: AARG, REM <-- AARG / BARG
;   Max Timing: 1+291+2 = 294 clks
;   Min Timing: 1+227+2 = 230 clks
;   PM: 1+39+1 = 41           DM: 7
FXD1608U   CLRF           REMB0
           UDIV1608L
           RETLW          0x00
;*****
;*****

;   16/7 Bit Unsigned Fixed Point Divide 16/7 -> 16.07
;   Input: 16 bit unsigned fixed point dividend in AARGB0, AARGB1
;           7 bit unsigned fixed point divisor in BARGB0
;   Use:   CALL    FXD1607U
;   Output: 16 bit unsigned fixed point quotient in AARGB0, AARGB1
;           7 bit unsigned fixed point remainder in REMB0
;   Result: AARG, REM <-- AARG / BARG
;   Max Timing: 1+171+2 = 174 clks
;   Min Timing: 1+171+2 = 174 clks
;   PM: 1+39+1 = 41           DM: 5
FXD1607U   CLRF           REMB0
           UDIV1607L
           RETLW          0x00
;*****
;*****

```

AN617

```
;*****
;
; 15/7 Bit Unsigned Fixed Point Divide 15/7 -> 15.07
; Input: 15 bit unsigned fixed point dividend in AARGB0, AARGB1
;       7 bit unsigned fixed point divisor in BARGB0
; Use:   CALL   FXD1507U
; Output: 15 bit unsigned fixed point quotient in AARGB0, AARGB1
;       7 bit unsigned fixed point remainder in REMB0
; Result: AARG, REM <-- AARG / BARG
; Max Timing: 1+163+2 = 166 clks
; Min Timing: 1+163+2 = 166 clks
; PM: 1+42+1 = 44      DM: 5
FXD1507U      CLRF      REMB0
              UDIV1507L
              RETLW    0x00
;*****
;*****
                END
```

D.8 8/8 PIC16C5X/PIC16CXX Fixed Point Divide Routines

```
; 8/8 PIC16 FIXED POINT DIVIDE ROUTINES  VERSION 1.7
; Input: fixed point arguments in AARG and BARG
; Output: quotient AARG/BARG in AARG followed by remainder in REM
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed divide application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in
; performance.
; Routine      Clocks      Function
; FXD0808S     96 8 bit/8 bit -> 08.08 signed fixed point divide
; FXD0808U     100 8 bit/8 bit -> 08.08 unsigned fixed point divide
; FXD0807U     88 8 bit/7 bit -> 08.07 unsigned fixed point divide
; FXD0707U     80 7 bit/7 bit -> 07.07 unsigned fixed point divide
; The above timings are based on the looped macros. If space permits,
; approximately 19-25 clocks can be saved by using the unrolled macros.
; list      r=dec,x=on,t=off
; include <PIC16.INC>      ; general PIC16 definitions
; include <MATH16.INC>    ; PIC16 math library definitions
;*****
;*****
; Test suite storage
RANDHI      equ      0x1A      ; random number generator registers
RANDLO      equ      0x1B
TESTCOUNT  equ      0x20      ; counter
DATA        equ      0x21      ; beginning of test data
;*****
;*****
; Test suite for 8/8 bit fixed point divide algorithms
MAIN        org          0x0005
           MOVLW        RAMSTART
           MOVWF        FSR
MEMLOOP     CLRF        INDF
           INCF        FSR
           MOVLW        RAMSTOP
           SUBWF        FSR,W
           BTFSS        _Z
           GOTO        MEMLOOP
           MOVLW        0x45          ; seed for random numbers
           MOVWF        RANDLO
           MOVLW        0x30
           MOVWF        RANDHI
           MOVLW        DATA
           MOVWF        FSR
           BCF        _RP0
           BCF        _RP1
           BCF        _IRP
```

```

                CALL          TFXD0808
SELF           GOTO          SELF
RANDOM16       RLF           RANDHI,W           ; random number generator
                XORWF        RANDHI,W
                MOVWF        TEMPB0
                SWAPF        RANDHI
                SWAPF        RANDLO,W
                MOVWF        TEMPB1
                RLF          TEMPB1,W
                RLF          TEMPB1
                MOVF         TEMPB1,W
                XORWF        RANDHI,W
                SWAPF        RANDHI
                ANDLW        0x01
                RLF          TEMPB0
                RLF          RANDLO
                XORWF        RANDLO
                RLF          RANDHI

                RETLW        0
;           Test suite for FXD0808
TFXD0808      MOVLW         3
                MOVWF        TESTCOUNT
D0808LOOP
                CALL          RANDOM16
                MOVF         RANDHI,W
                MOVWF        BARGB0
;           BCF          BARGB0,MSB
;           MOVF         BARGB0,W
                MOVWF        INDF
                INCF        FSR
                CALL          RANDOM16
                MOVF         RANDHI,W
                MOVWF        AARGB0
;           BCF          AARGB0,MSB
;           MOVF         AARGB0,W
                MOVWF        INDF
                INCF        FSR
                CALL          FXD0808S
                MOVF         AARGB0,W
                MOVWF        INDF
                INCF        FSR
                MOVF         REMB0,W
                MOVWF        INDF
                INCF        FSR
                MOVF         TESTCOUNT
                GOTO          D0808LOOP
                RETLW        0x00
;*****
;*****
;           08/08 BIT Division Macros
SDIV0808L    macro
;           Max Timing:      3+5+2+5*11+10+2 = 77 clks
;           Min Timing:      3+5+2+5*11+10+2 = 77 clks
;           PM: 22                               DM: 4
                MOVF         BARGB0,W
                SUBWF        REMB0
                RLF          ACCB0
                RLF          ACCB0,W
                RLF          REMB0
                MOVF         BARGB0,W
                ADDWF        REMB0
                RLF          ACCB0
                MOVLW        6
                MOVWF        LOOPCOUNT
LOOPS0808A   RLF          ACCB0,W

```

AN617

```

        RLF          REMB0
        MOVF         BARGB0,W
        BTFSC       ACCB0,LSB
        SUBWF       REMB0
        BTFSS       ACCB0,LSB
        ADDWF       REMB0
        RLF         ACCB0
        DECFSZ      LOOPCOUNT
        GOTO        LOOPS0808A
        BTFSS       ACCB0,LSB
        ADDWF       REMB0
        endm
UDIV0808L macro
;      Max Timing: 2+7*12+11 = 97 clks
;      Min Timing: 2+7*11+10 = 89 clks
;      PM: 13                      DM: 4
        MOVLW       8
        MOVWF      LOOPCOUNT
LOOPU0808A RLF          ACCB0,W
        RLF        REMB0
        MOVF       BARGB0,W
        SUBWF     REMB0
        BTFSC     _C
        GOTO      UOK88A
        ADDWF    REMB0
        BCF       _C
UOK88A  RLF          ACCB0
        DECFSZ    LOOPCOUNT
        GOTO      LOOPU0808A
        endm
UDIV0807L macro
;      Max Timing: 7+6*11+10+2 = 85 clks
;      Min Timing: 7+6*11+10+2 = 85 clks
;      PM: 19                      DM: 4
        RLF          ACCB0,W
        RLF          REMB0
        MOVF         BARGB0,W
        SUBWF       REMB0
        RLF         ACCB0
        MOVLW       7
        MOVWF      LOOPCOUNT
LOOPU0807 RLF          ACCB0,W
        RLF        REMB0
        MOVF       BARGB0,W
        BTFSC     ACCB0,LSB
        SUBWF     REMB0
        BTFSS     ACCB0,LSB
        ADDWF    REMB0
        RLF         ACCB0
        DECFSZ    LOOPCOUNT
        GOTO      LOOPU0807
        BTFSS     ACCB0,LSB
        ADDWF    REMB0
        endm
UDIV0707L macro
;      Max Timing: 3+5+2+5*11+10+2 = 77 clks
;      Min Timing: 3+5+2+5*11+10+2 = 77 clks
;      PM: 22                      DM: 4
        MOVF         BARGB0,W
        SUBWF       REMB0
        RLF         ACCB0
        RLF         ACCB0,W
        RLF         REMB0
        MOVF       BARGB0,W
        ADDWF      REMB0
        RLF         ACCB0
```



```

                MOVLW          6
LOOPU0707      MOVWF          LOOPCOUNT
                RLF           ACCB0,W
                RLF           REMB0
                MOVF          BARGB0,W
                BTFSC        ACCB0,LSB
                SUBWF        REMB0
                BTFSS        ACCB0,LSB
                ADDWF        REMB0
                RLF           ACCB0
                DECFSZ        LOOPCOUNT
                GOTO         LOOPU0707
                BTFSS        ACCB0,LSB
                ADDWF        REMB0
                endm
SDIV0808      macro
;           Max Timing:      3+5+6*8+2 = 58 clks
;           Min Timing:      3+5+6*8+2 = 58 clks
;           PM: 58
                variable i
                MOVF          BARGB0,W
                SUBWF        REMB0
                RLF           ACCB0
                RLF           ACCB0,W
                RLF           REMB0
                MOVF          BARGB0,W
                ADDWF        REMB0
                RLF           ACCB0
                i = 2
                while i < 8
                RLF           ACCB0,W
                RLF           REMB0
                MOVF          BARGB0,W
                BTFSC        ACCB0,LSB
                SUBWF        REMB0
                BTFSS        ACCB0,LSB
                ADDWF        REMB0
                RLF           ACCB0
                i=i+1
                endw
                BTFSS        ACCB0,LSB
                ADDWF        REMB0
                endm
UDIV0808      macro
;           restore = 9 clks, nonrestore = 8 clks
;           Max Timing:      8*9 = 72 clks
;           Min Timing:      8*8 = 64 clks
;           PM: 72
                variable      i
                i = 0
                while i < 8
                RLF           ACCB0,W
                RLF           REMB0
                MOVF          BARGB0,W
                SUBWF        REMB0
                BTFSC        _C
                GOTO         UOK88#v(i)
                ADDWF        REMB0
                BCF           _C
UOK88#v(i)    RLF           ACCB0
                i=i+1
                endw
                endm
UDIV0807      macro
;           Max Timing:      5+7*8+2 = 63 clks
;           Min Timing:      5+7*8+2 = 63 clks

```

AN617

```
;          PM: 63                                     DM: 3
          variable i
          RLF          ACCB0,W
          RLF          REMB0
          MOVF         BARGB0,W
          SUBWF        REMB0
          RLF          ACCB0
          i = 1
          while i < 8
          RLF          ACCB0,W
          RLF          REMB0
          MOVF         BARGB0,W
          BTFSC        ACCB0,LSB
          SUBWF        REMB0
          BTFSS        ACCB0,LSB
          ADDWF        REMB0
          RLF          ACCB0
          i=i+1
          endw
          BTFSS        ACCB0,LSB
          ADDWF        REMB0
          endm
UDIV0707    macro
;          Max Timing:      3+5+6*8+2 = 58 clks
;          Min Timing:      3+5+6*8+2 = 58 clks
;          PM: 58                                     DM: 3
          variable i
          MOVF         BARGB0,W
          SUBWF        REMB0
          RLF          ACCB0
          RLF          ACCB0,W
          RLF          REMB0
          MOVF         BARGB0,W
          ADDWF        REMB0
          RLF          ACCB0
          i = 2
          while i < 8
          RLF          ACCB0,W
          RLF          REMB0
          MOVF         BARGB0,W
          BTFSC        ACCB0,LSB
          SUBWF        REMB0
          BTFSS        ACCB0,LSB
          ADDWF        REMB0
          RLF          ACCB0
          i=i+1
          endw
          BTFSS        ACCB0,LSB
          ADDWF        REMB0
          endm

;*****
;*****

;          8/8 Bit Signed Fixed Point Divide 8/8 -> 08.08
;          Input:  8 bit signed fixed point dividend in AARGB0
;          Input:  8 bit signed fixed point divisor in BARGB0
;          Use:    CALL    FXD0808S
;          Output: 8 bit signed fixed point quotient in AARGB0
;          Output: 8 bit signed fixed point remainder in REMB0
;          Result: AARG, REM <-- AARG / BARG
;          Max Timing: 10+77+3 = 90 clks          A > 0, B > 0
;                   11+77+8 = 96 clks          A > 0, B < 0
;                   11+77+8 = 96 clks          A < 0, B > 0
;                   12+77+3 = 92 clks          A < 0, B < 0
;          Min Timing: 10+77+3 = 90 clks          A > 0, B > 0
```

```

;          11+77+8 = 96 clks          A > 0, B < 0
;          11+77+8 = 96 clks          A < 0, B > 0
;          12+77+3 = 92 clks          A < 0, B < 0
;      PM: 12+22+7 = 41          DM: 5
FXD0808S      MOVF          AARGB0,W
              XORWF          BARGB0,W
              MOVWF          SIGN
              BTFSS          BARGB0,MSB          ; if MSB set, negate BARG
              GOTO          CA0808S
              COMF          BARGB0
              INCF          BARGB0
CA0808S      BTFSS          AARGB0,MSB          ; if MSB set, negate AARG
              GOTO          C0808S
              COMF          AARGB0
              INCF          AARGB0
C0808S      CLRF          REMB0
              SDIV0808L
              BTFSS          SIGN,MSB
              RETLW          0x00
              COMF          AARGB0
              INCF          AARGB0
              COMF          REMB0
              INCF          REMB0
              RETLW          0x00
;*****
;*****

;      8/8 Bit Unsigned Fixed Point Divide 8/8 -> 08.08
;      Input:  8 bit unsigned fixed point dividend in AARGB0
;              8 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL      FXD0808U
;      Output: 8 bit unsigned fixed point quotient in AARGB0
;              8 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:  1+97+2 = 100 clks
;      Min Timing:  1+89+2 = 92 clks
;      PM: 1+13+1 = 15          DM: 4
FXD0808U      CLRF          REMB0
              UDIV0808L
              RETLW          0x00
;*****
;*****

;      8/7 Bit Unsigned Fixed Point Divide 8/7 -> 08.07
;      Input:  8 bit unsigned fixed point dividend in AARGB0
;              7 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL      FXD0807U
;      Output: 8 bit unsigned fixed point quotient in AARGB0
;              7 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:  1+85+2 = 88 clks
;      Min Timing:  1+85+2 = 88 clks
;      PM: 1+19+1 = 21          DM: 4
FXD0807U      CLRF          REMB0
              UDIV0807L
              RETLW          0x00
;*****
;*****

;      7/7 Bit Unsigned Fixed Point Divide 7/7 -> 07.07
;      Input:  7 bit unsigned fixed point dividend in AARGB0
;              7 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL      FXD0707U
;      Output: 7 bit unsigned fixed point quotient in AARGB0
;              7 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM  <--  AARG / BARG

```

AN617

```
;      Max Timing:      1+77+2 = 80 clks
;      Min Timing:      1+77+2 = 80 clks
;      PM: 1+22+1 = 44      DM: 4
FXD0707U      CLRF      REMB0
              UDIV0707L
              RETLW      0x00
;*****
;*****
              END
```

NOTES:

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX E: PIC17CXX MULTIPLY ROUTINES

```

; PIC17 FIXED POINT MULTIPLY ROUTINES          VERSION 1.3
; Input:  fixed point arguments in AARG and BARG
; Output: product AARG*BARG in AARG
; All timings are worst case cycle counts
; It is useful to note that the additional routines FXM3115U, FXM1515U, and
; FXM1507U can be called in a signed multiply application in the special case
; where AARG > 0 and BARG > 0, thereby offering some improvement in
; performance.
; Routine      Clocks      Function
; FXM0808S     53          08x08 -> 16 bit signed fixed point multiply
; FXM0808U     39          08x08 -> 16 bit unsigned fixed point multiply
; FXM0707U     37          07x07 -> 14 bit unsigned fixed point multiply
; FXM1608S     79          16x08 -> 24 bit signed fixed point multiply
; FXM1608U     75          16x08 -> 24 bit unsigned fixed point multiply
; FXM1507U     69          15x07 -> 22 bit unsigned fixed point multiply
; FXM1616S     175         16x16 -> 32 bit signed fixed point multiply
; FXM1616U     156         16x16 -> 32 bit unsigned fixed point multiply
; FXM1515U     150         15x15 -> 30 bit unsigned fixed point multiply
; FXM2416S     223         24x16 -> 40 bit signed fixed point multiply
; FXM2416U     203         24x16 -> 40 bit unsigned fixed point multiply
; FXM2315U     194         23x15 -> 38 bit unsigned fixed point multiply
; FXM2424S     346         24x24 -> 48 bit signed fixed point multiply
; FXM2424U     316         24x24 -> 48 bit unsigned fixed point multiply
; FXM2323U     308         23x23 -> 46 bit unsigned fixed point multiply
; FXM3216S     270         32x16 -> 48 bit signed fixed point multiply
; FXM3216U     265         32x16 -> 48 bit unsigned fixed point multiply
; FXM3115U     254         31x15 -> 46 bit unsigned fixed point multiply
; FXM3224S     417         32x24 -> 56 bit signed fixed point multiply
; FXM3224U     410         32x24 -> 56 bit unsigned fixed point multiply
; FXM3123U     399         31x23 -> 54 bit unsigned fixed point multiply
; FXM3232S     572         32x32 -> 64 bit signed fixed point multiply
; FXM3232U     563         32x32 -> 64 bit unsigned fixed point multiply
; FXM3131U     543         31x31 -> 62 bit unsigned fixed point multiply

list      r=dec,x=on,t=off,p=17C42
include <PIC17.INC>          ; general PIC17 definitions

include <MATH17.INC>        ; PIC17 math library definitions
;*****
;*****
; Test suite storage
RANDHI     equ      0x2B     ; random number generator registers
RANDLO     equ      0x2C
TESTCODE   equ      0x2D     ; integer code labeling test contained in following data
NUMTESTS   equ      0x2E     ; number of tests contained in following data
TESTCOUNT equ      0x2F     ; counter
DATA       equ      0x30     ; beginning of test data
;*****
;*****
; Test suite for fixed point multiplication algorithms
MAIN       org          0x0021
           MOVLW        RAMSTART
           MOVPF        WREG,FSR0
MEMLOOP    CLRF        INDF0
           INCFSZ       FSR0
           GOTO         MEMLOOP
           BSF          RTCSTA,5
;           MOVPF        RTCCH,WREG
           MOVLW        0x45             ; seed for random numbers
           MOVPF        WREG,RANDLO
;           MOVPF        RTCCL,WREG
           MOVLW        0x30
           MOVPF        WREG,RANDHI

```

```

        MOVLW          0x30
        MOVFPF         WREG,FSR0
        BCF            _FS1
        BSF            _FS0
;
;      CALL           TFXM0808
;      CALL           TFXM1608
;      CALL           TFXM1616
;      CALL           TFXM2416
;      CALL           TFXM2424
;      CALL           TFXM3216
;      CALL           TFXM3224
;      CALL           TFXM3232
SELF    GOTO          SELF
RANDOM16 RLCF          RANDHI,W          ; random number generator
        XORWF         RANDHI,W
        RLCF          WREG
        SWAPF        RANDHI
        SWAPF        RANDLO,W
        RLNCF        WREG
        XORWF        RANDHI,W
        SWAPF        RANDHI
        ANDLW        0x01
        RLCF          RANDLO
        XORWF        RANDLO
        RLCF          RANDHI

        RETLW         0
;      Test suite for FXM0808
TFXM0808 MOVLW         52
        MOVFPF        WREG,TESTCOUNT
        MOVFPF        WREG,NUMTESTS
        MOVLW         1
        MOVFPF        WREG,TESTCODE
M0808LOOP
        CALL          RANDOM16
        MOVFPF        RANDHI,WREG
        MOVFPF        WREG,BARGB0
;      BCF            BARGB0,MSB

        MOVFPF        RANDLO,WREG
        MOVFPF        WREG,AARGB0
;      BCF            AARGB0,MSB
        MOVFPF        AARGB0,INDF0
        MOVFPF        BARGB0,INDF0
        CALL          FXM0808S
;      CALL           FXM0808U
;      CALL           FXM0707U
        MOVFPF        AARGB0,INDF0
        MOVFPF        AARGB1,INDF0
        DECFSZ        TESTCOUNT
        GOTO          M0808LOOP
        RETLW         0x00
;      Test suite for FXM1608
TFXM1608 MOVLW         34
        MOVFPF        WREG,TESTCOUNT
        MOVFPF        WREG,NUMTESTS
        MOVLW         2
        MOVFPF        WREG,TESTCODE
M1608LOOP
        CALL          RANDOM16
        MOVFPF        RANDHI,WREG
        MOVFPF        WREG,BARGB0
;      BCF            BARGB0,MSB
        CALL          RANDOM16
        MOVFPF        RANDHI,WREG
        MOVFPF        WREG,AARGB0

```

AN617

```
MOVFP          RANDLO,WREG
MOVFP          WREG,AARB1
;
BCF            AARB0,MSB
MOVFP          AARB0,INDF0
MOVFP          AARB1,INDF0
MOVFP          BARB0,INDF0
CALL          FXM1608S
;
CALL          FXM1608U
;
CALL          FXM1507U
MOVFP          AARB0,INDF0
MOVFP          AARB1,INDF0
MOVFP          AARB2,INDF0
DECFSZ        TESTCOUNT
GOTO          M1608LOOP
RETLW         0x00
;
;           Test suite for FXM1616
TFXM1616      MOVLW         26
MOVFP          WREG,TESTCOUNT
MOVFP          WREG,NUMTESTS
MOVLW         3
MOVFP          WREG,TESTCODE
M1616LOOP
CALL          RANDOM16
MOVFP          RANDHI,WREG
MOVFP          WREG,BARB0
MOVFP          RANDLO,WREG
MOVFP          WREG,BARB1
;
BCF            BARB0,MSB
CALL          RANDOM16
MOVFP          RANDHI,WREG
MOVFP          WREG,AARB0
MOVFP          RANDLO,WREG
MOVFP          WREG,AARB1
;
BCF            AARB0,MSB
MOVFP          AARB0,INDF0
MOVFP          AARB1,INDF0
MOVFP          BARB0,INDF0
MOVFP          BARB1,INDF0
;
CALL          FXM1616S
CALL          FXM1616U
;
CALL          FXM1515U
MOVFP          AARB0,INDF0
MOVFP          AARB1,INDF0
MOVFP          AARB2,INDF0
MOVFP          AARB3,INDF0
DECFSZ        TESTCOUNT
GOTO          M1616LOOP
RETLW         0x00
;
;           Test suite for FXM2416
TFXM2416      MOVLW         20
MOVFP          WREG,TESTCOUNT
MOVFP          WREG,NUMTESTS
MOVLW         4
MOVFP          WREG,TESTCODE
M2416LOOP
CALL          RANDOM16
MOVFP          RANDHI,WREG
MOVFP          WREG,BARB0
MOVFP          RANDLO,WREG
MOVFP          WREG,BARB1
;
BCF            BARB0,MSB
CALL          RANDOM16
MOVFP          RANDHI,WREG
MOVFP          WREG,AARB0
MOVFP          RANDLO,WREG
MOVFP          WREG,AARB1
```



```

CALL          RANDOM16
MOVFP        RANDHI , WREG
MOVPF        WREG , AARGB2
;
BCF          AARGB0 , MSB
MOVFP        AARGB0 , INDF0
MOVFP        AARGB1 , INDF0
MOVFP        AARGB2 , INDF0
MOVFP        BARGB0 , INDF0
MOVFP        BARGB1 , INDF0
CALL         FXM2416S
;
CALL         FXM2416U
;
CALL         FXM2315U
MOVFP        AARGB0 , INDF0
MOVFP        AARGB1 , INDF0
MOVFP        AARGB2 , INDF0
MOVFP        AARGB3 , INDF0
MOVFP        AARGB4 , INDF0
DECFSZ      TESTCOUNT
GOTO        M2416LOOP
RETLW       0x00
;
;           Test suite for FXM2424
TFXM2424    MOVLW       17
MOVPF        WREG , TESTCOUNT
MOVPF        WREG , NUMTESTS
MOVLW       5
MOVPF        WREG , TESTCODE
M2424LOOP
CALL         RANDOM16
MOVFP        RANDHI , WREG
MOVPF        WREG , BARGB0
MOVFP        RANDLO , WREG
MOVPF        WREG , BARGB1
CALL         RANDOM16
MOVFP        RANDHI , WREG
MOVPF        WREG , BARGB2
BCF          BARGB0 , MSB
MOVFP        RANDLO , WREG
MOVPF        WREG , AARGB0
CALL         RANDOM16
MOVFP        RANDHI , WREG
MOVPF        WREG , AARGB1
MOVFP        RANDLO , WREG
MOVPF        WREG , AARGB2
BCF          AARGB0 , MSB
MOVFP        AARGB0 , INDF0
MOVFP        AARGB1 , INDF0
MOVFP        AARGB2 , INDF0
MOVFP        BARGB0 , INDF0
MOVFP        BARGB1 , INDF0
MOVFP        BARGB2 , INDF0
;
;           Test suite for FXM2424S
CALL         FXM2424S
;
CALL         FXM2424U
CALL         FXM2323U
MOVFP        AARGB0 , INDF0
MOVFP        AARGB1 , INDF0
MOVFP        AARGB2 , INDF0
MOVFP        AARGB3 , INDF0
MOVFP        AARGB4 , INDF0
MOVFP        AARGB5 , INDF0
DECFSZ      TESTCOUNT
GOTO        M2424LOOP
RETLW       0x00
;
;           Test suite for FXM3216
TFXM3216    MOVLW       17
MOVPF        WREG , TESTCOUNT

```

AN617

```

MOVFP          WREG, NUMTESTS
MOVLW          6
MOVFP          WREG, TESTCODE

M3216LOOP
CALL           RANDOM16
MOVFP         RANDHI, WREG
MOVFP         WREG, BARGB0
MOVFP         RANDLO, WREG
MOVFP         WREG, BARGB1
BCF           BARGB0, MSB
CALL           RANDOM16

MOVFP         RANDHI, WREG
MOVFP         WREG, AARGB0
MOVFP         RANDLO, WREG
MOVFP         WREG, AARGB1
CALL           RANDOM16
MOVFP         RANDHI, WREG
MOVFP         WREG, AARGB2
MOVFP         RANDLO, WREG
MOVFP         WREG, AARGB3
BCF           AARGB0, MSB
MOVFP         AARGB0, INDF0
MOVFP         AARGB1, INDF0
MOVFP         AARGB2, INDF0
MOVFP         AARGB3, INDF0
MOVFP         BARGB0, INDF0
MOVFP         BARGB1, INDF0
;
;
CALL           FXM3216S
CALL           FXM3216U
CALL           FXM3115U
MOVFP         AARGB0, INDF0
MOVFP         AARGB1, INDF0
MOVFP         AARGB2, INDF0
MOVFP         AARGB3, INDF0
MOVFP         AARGB4, INDF0
MOVFP         AARGB5, INDF0
DECFSZ        TESTCOUNT
GOTO          M3216LOOP
RETLW         0x00
;
;   Test suite for FXM3224
TFXM3224      MOVLW          14
MOVFP         WREG, TESTCOUNT
MOVFP         WREG, NUMTESTS
MOVLW          6
MOVFP         WREG, TESTCODE

M3224LOOP
CALL           RANDOM16
MOVFP         RANDHI, WREG
MOVFP         WREG, BARGB0
MOVFP         RANDLO, WREG
MOVFP         WREG, BARGB1
CALL           RANDOM16
MOVFP         RANDHI, WREG
MOVFP         WREG, BARGB2
;
BCF           BARGB0, MSB
CALL           RANDOM16

MOVFP         RANDHI, WREG
MOVFP         WREG, AARGB0
MOVFP         RANDLO, WREG
MOVFP         WREG, AARGB1
CALL           RANDOM16
MOVFP         RANDHI, WREG
MOVFP         WREG, AARGB2
MOVFP         RANDLO, WREG
```

```

MOVFPF      WREG, AARGB3
;
BCF         AARGB0, MSB
MOVFPF      AARGB0, INDF0
MOVFPF      AARGB1, INDF0
MOVFPF      AARGB2, INDF0
MOVFPF      AARGB3, INDF0
MOVFPF      BARGB0, INDF0
MOVFPF      BARGB1, INDF0
MOVFPF      BARGB2, INDF0
CALL        FXM3224S
;
CALL        FXM3224U
;
CALL        FXM3123U
MOVFPF      AARGB0, INDF0
MOVFPF      AARGB1, INDF0
MOVFPF      AARGB2, INDF0
MOVFPF      AARGB3, INDF0
MOVFPF      AARGB4, INDF0
MOVFPF      AARGB5, INDF0
MOVFPF      AARGB6, INDF0
DECFSZ     TESTCOUNT
GOTO       M3224LOOP
RETLW      0x00
;
Test suite for FXM3232
TFXM3232   MOVLW      13
MOVFPF      WREG, TESTCOUNT
MOVFPF      WREG, NUMTESTS
MOVLW      7
MOVFPF      WREG, TESTCODE

M3232LOOP

CALL        RANDOM16
MOVFPF      RANDHI, WREG
MOVFPF      WREG, BARGB0
MOVFPF      RANDLO, WREG
MOVFPF      WREG, BARGB1
CALL        RANDOM16
MOVFPF      RANDHI, WREG
MOVFPF      WREG, BARGB2
MOVFPF      RANDLO, WREG
MOVFPF      WREG, BARGB3
BCF         BARGB0, MSB
CALL        RANDOM16

MOVFPF      RANDHI, WREG
MOVFPF      WREG, AARGB0
MOVFPF      RANDLO, WREG
MOVFPF      WREG, AARGB1
CALL        RANDOM16
MOVFPF      RANDHI, WREG
MOVFPF      WREG, AARGB2
MOVFPF      RANDLO, WREG
MOVFPF      WREG, AARGB3
BCF         AARGB0, MSB
MOVFPF      AARGB0, INDF0
MOVFPF      AARGB1, INDF0
MOVFPF      AARGB2, INDF0
MOVFPF      AARGB3, INDF0
MOVFPF      BARGB0, INDF0
MOVFPF      BARGB1, INDF0
MOVFPF      BARGB2, INDF0
MOVFPF      BARGB3, INDF0
;
CALL        FXM3232S
;
CALL        FXM3232U
CALL        FXM3131U
MOVFPF      AARGB0, INDF0
MOVFPF      AARGB1, INDF0
MOVFPF      AARGB2, INDF0

```

AN617

```
MOVFP      AARGB3,INDF0
MOVFP      AARGB4,INDF0
MOVFP      AARGB5,INDF0
MOVFP      AARGB6,INDF0
MOVFP      AARGB7,INDF0
DECFSZ    TESTCOUNT
GOTO      M3232LOOP
RETLW     0x00
;*****
;*****
;      Multiplication Macros
SMUL0808   macro
;      Max Timing:      4+6+6*5+3 = 43 clks
;      Min Timing:      4+14+3 = 21 clks
;      PM: 4+2*7+5+6*5+3 = 56          DM: 5
      variable i
      i = 0
      CLRF          SIGN
      BTFSC         AARGB0,MSB
      COMF          SIGN
      MOVFP         AARGB0,WREG

      while i < 7

      BTFSC         BARGB0,i
      GOTO          SM0808NA#v(i)
      i = i + 1
      endw
      CLRF          ACCB0          ; if we get here, BARG = 0
      RETLW        0
SM0808NA0  RLCF          SIGN
      RRCF          ACCB0
      RRCF          ACCB1
      i = 1
      while i < 7
      BTFSC         BARGB0,i
SM0808NA#v(i) RLCF          SIGN
      RRCF          ACCB0
      RRCF          ACCB1
      i = i + 1
      endw
      RLCF          SIGN
      RRCF          ACCB0
      RRCF          ACCB1
      endm
UMUL0808   macro
;      Max Timing:      2+5+7*4 = 35 clks
;      Min Timing:      2+16+3 = 21 clks
;      PM: 2+2*8+4+7*4 = 50          DM: 3
      variable i
      i = 0
      BCF           _C          ; clear carry for first right shift
      MOVFP         AARGB0,WREG

      while i < 8

      BTFSC         BARGB0,i
      GOTO          UM0808NA#v(i)
      i = i + 1
      endw
      CLRF          ACCB0          ; if we get here, BARG = 0
      RETLW        0
UM0808NA0  RRCF          ACCB0
      RRCF          ACCB1
      i = 1
```

```

        while    i < 8
        BTFSC    BARGB0,i
        ADDWF    ACCB0
UM0808NA#v(i)  RRCF    ACCB0
        RRCF    ACCB1
        i = i + 1
        endw
        endm

UMUL0707      macro
;      Max Timing:    2+5+6*4+2 = 33 clks
;      Min Timing:    2+14+3 = 19 clks
;      PM: 2+2*7+4+6*4+2 = 46          DM: 3
        variable i
        i = 0
        BCF     _C          ; clear carry for first right shift
        MOVFP   AARGB0,WREG
        while i < 7

                BTFSC    BARGB0,i
                GOTO     UM0707NA#v(i)
                i = i + 1
                endw
                CLRF     ACCB0          ; if we get here, BARG = 0
                RETLW    0
UM0707NA0     RRCF    ACCB0
                RRCF    ACCB1
                i = 1
                while  i < 7
                BTFSC    BARGB0,i
                ADDWF    ACCB0
UM0707NA#v(i) RRCF    ACCB0
                RRCF    ACCB1
                i = i + 1
                endw
                RRCF    ACCB0
                RRCF    ACCB1
                endm
;-----
SSMUL1608     macro
;      Max Timing:    3+6+6*9+3 = 66 clks
;      Min Timing:    3+21+5 = 29 clks
;      PM: 3+3*7+7+6*9+3 = 88          DM: 6
        variable i
        i = 0
        BTFSC    AARGB0,MSB
        COMF     ACCB2
        RLCF     ACCB0,W

        while i < 7

                BTFSC    BARGB0,i
                GOTO     SSM1608NA#v(i)
                BCF     ACCB2,7-i
                i = i + 1
                endw
                CLRF     ACCB0          ; if we get here, BARG = 0
                CLRF    ACCB1
                CLRF    ACCB2
                RETLW    0
SSM1608NA0    RRCF    ACCB0
                RRCF    ACCB1
                RRCF    ACCB2
                i = 1
                while  i < 7
                BTFSS    BARGB0,i

```

AN617

```
SSM1608A#v(i)    GOTO          SSM1608NA#v(i)
                  MOVFP          TEMPB1,WREG
                  ADDWF          ACCB1
                  MOVFP          TEMPB0,WREG
                  ADDWFC         ACCB0

SSM1608NA#v(i)   RRCF          ACCB0
                  RRCF          ACCB1
                  RRCF          ACCB2
                  i = i + 1
                  endw
                  RRCF          ACCB0
                  RRCF          ACCB1
                  RRCF          ACCB2
                  endm

SMUL1608         macro
;      Max Timing:      7+6*10+4 = 71 clks
;      Min Timing:      7*2+4 = 18 clks
;      PM: 7*2+7+6*10+4 = 85          DM: 6
                variable i
                i = 0
                while i < 7

                    BTFSC          BARGB0,i
                    GOTO          SM1608NA#v(i)
                    i = i + 1
                endw
                CLRF          ACCB0          ; if we get here, BARG = 0
                CLRF          ACCB1
                RETLW         0
SM1608NA0        RLCF          TEMPB0,W
                  RRCF          ACCB0
                  RRCF          ACCB1
                  RRCF          ACCB2
                  i = 1
                  while i < 7
                      BTFSS          BARGB0,i
                      GOTO          SM1608NA#v(i)
SM1608A#v(i)     MOVFP          TEMPB1,WREG
                  ADDWF          ACCB1
                  MOVFP          TEMPB0,WREG
                  ADDWFC         ACCB0
SM1608NA#v(i)   RLCF          TEMPB0,W
                  RRCF          ACCB0
                  RRCF          ACCB1
                  RRCF          ACCB2
                  i = i + 1
                  endw
                  RLCF          TEMPB0,W
                  RRCF          ACCB0
                  RRCF          ACCB1
                  RRCF          ACCB2
                  endm

UMUL1608         macro
;      Max Timing:      1+6+7*9 = 70 clks
;      Min Timing:      1+8*2+4 = 21 clks
;      PM: 1+8*2+6+7*9 = 86          DM: 6
                variable i
                i = 0

                BCF          _C
                while i < 8

                    BTFSC          BARGB0,i
                    GOTO          UM1608NA#v(i)
```

```

        i = i + 1
    endw
    CLRF          ACCB0          ; if we get here, BARG = 0
    CLRF          ACCB1
    RETLW        0
UM1608NA0      RRCF          ACCB0
               RRCF          ACCB1
               RRCF          ACCB2
               i = 1
               while i < 8
UM1608A#v(i)   BTFSS        BARGB0,i
               GOTO        UM1608NA#v(i)
               MOVFP        TEMPB1,WREG
               ADDWF        ACCB1
               MOVFP        TEMPB0,WREG
               ADDWFC       ACCB0
UM1608NA#v(i)  RRCF          ACCB0
               RRCF          ACCB1
               RRCF          ACCB2
               i = i + 1
    endw
    endm
UMUL1507      macro
;      Max Timing:      1+6+6*9+3 = 64 clks
;      Min Timing:      1+7*2+4 = 19 clks
;      PM: 1+7*2+6+6*9+3 = 78          DM: 6
               variable i
               i = 0
               BCF          _C
               while i < 7
UM1507NA0     BTFSC        BARGB0,i
               GOTO        UM1507NA#v(i)
               i = i + 1
    endw
    CLRF          ACCB0          ; if we get here, BARG = 0
    CLRF          ACCB1
    RETLW        0
UM1507NA0     RRCF          ACCB0
               RRCF          ACCB1
               RRCF          ACCB2
               i = 1
               while i < 7
UM1507A#v(i)  BTFSS        BARGB0,i
               GOTO        UM1507NA#v(i)
               MOVFP        TEMPB1,WREG
               ADDWF        ACCB1
               MOVFP        TEMPB0,WREG
               ADDWFC       ACCB0
UM1507NA#v(i) RRCF          ACCB0
               RRCF          ACCB1
               RRCF          ACCB2
               i = i + 1
    endw
    RRCF          ACCB0
    RRCF          ACCB1
    RRCF          ACCB2
    endm
;-----
SMUL1616      macro
;      Max Timing:      7+7*10+7*11+5 = 159 clks
;      Min Timing:      15*2+4 = 34 clks
;      PM: 15*2+7+7*10+7*11+5 = 193          DM: 8
               variable i
               i = 0
               while i < 15

```

AN617

```

    if i < 8
    BTFSC          BARGB1,i
    else
    BTFSC          BARGB0,i-8
    endif
    GOTO          SM1616NA#v(i)
    i = i + 1
endw
CLRF          ACCB0          ; if we get here, BARG = 0
CLRF          ACCB1
RETLW        0
SM1616NA0    RLCF          TEMPB0,W
             RRCF          ACCB0
             RRCF          ACCB1
             RRCF          ACCB2
             i = 1
             while i < 15
             if i < 8
             BTFSS          BARGB1,i
             else
             BTFSS          BARGB0,i-8
             endif
             GOTO          SM1616NA#v(i)
SM1616A#v(i) MOVFP          TEMPB1,WREG
             ADDWF          ACCB1
             MOVFP          TEMPB0,WREG
             ADDWFC          ACCB0
SM1616NA#v(i) RLCF          TEMPB0,W
             RRCF          ACCB0
             RRCF          ACCB1
             RRCF          ACCB2
             if i > 7
             RRCF          ACCB3
             endif
             i = i + 1
             endw
             RLCF          TEMPB0,W
             RRCF          ACCB0
             RRCF          ACCB1
             RRCF          ACCB2
             RRCF          ACCB3
             endm
UMUL1616    macro
;           Max Timing:      1+6+7*9+8*10 = 150 clks
;           Min Timing:      1+16*2+4 = 37 clks
;           PM: 1+16*2+4+6+7*9+8*10 = 186          DM: 8
             variable i

             i = 0

             BCF          _C
             while i < 16

             if i < 8
             BTFSC          BARGB1,i
             else
             BTFSC          BARGB0,i-8
             endif

             GOTO          UM1616NA#v(i)
             i = i + 1
             endw
             CLRF          ACCB0          ; if we get here, BARG = 0
             CLRF          ACCB1
             RETLW        0

```



```

UM1616NA0      RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                i = 1
                while i < 16
                if i < 8
                BTFSS          BARGB1,i
                else
                BTFSS          BARGB0,i-8
                endif
                GOTO          UM1616NA#v(i)
UM1616A#v(i)   MOVFP          TEMPB1,WREG
                ADDWF          ACCB1
                MOVFP          TEMPB0,WREG
                ADDWFC          ACCB0
UM1616NA#v(i)   RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                if i > 7
                RRCF          ACCB3
                endif
                i = i + 1
                endw
                endm
UMUL1515       macro
;      Max Timing:      1+6+7*9+7*10+4 = 144 clks
;      Min Timing:      1+15*2+4 = 35 clks
;      PM: 1+16*2+4+6+7*9+7*10+4 = 180          DM: 8
                variable i

                i = 0

                BCF          _C
                while i < 15

                if i < 8
                BTFSC          BARGB1,i
                else
                BTFSC          BARGB0,i-8
                endif

                GOTO          UM1515NA#v(i)
                i = i + 1
                endw
                CLRF          ACCB0          ; if we get here, BARG = 0
                CLRF          ACCB1
                RETLW          0
UM1515NA0      RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                i = 1
                while i < 15
                if i < 8
                BTFSS          BARGB1,i
                else
                BTFSS          BARGB0,i-8
                endif
                GOTO          UM1515NA#v(i)
UM1515A#v(i)   MOVFP          TEMPB1,WREG
                ADDWF          ACCB1
                MOVFP          TEMPB0,WREG
                ADDWFC          ACCB0
UM1515NA#v(i)   RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                if i > 7

```

AN617

```

        RRCF          ACCB3
    endif
    i = i + 1
endw
RRCF          ACCB0
RRCF          ACCB1
RRCF          ACCB2
RRCF          ACCB3
endm

;-----
SMUL2416      macro
;      Max Timing:      8+7*13+7*14+6 = 203 clks
;      Min Timing:      15*2+5 = 35 clks
;      PM: 15*2+9+7*13+7*14+6 = 234          DM: 10
        variable i
        i = 0
        while i < 15

            if i < 8
                BTFSC          BARGB1,i
            else

                BTFSC          BARGB0,i-8
            endif
            GOTO          SM2416NA#v(i)
            i = i + 1
        endw
        CLRF          ACCB0          ; if we get here, BARG = 0
        CLRF          ACCB1
        CLRF          ACCB2
        RETLW          0
SM2416NA0    RLCF          TEMPB0,W
        RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        i = 1
        while i < 15
            if i < 8
                BTFSS          BARGB1,i
            else
                BTFSS          BARGB0,i-8
            endif
            GOTO          SM2416NA#v(i)
SM2416A#v(i) MOVFP          TEMPB2,WREG
        ADDWF          ACCB2
        MOVFP          TEMPB1,WREG
        ADDWFC          ACCB1
        MOVFP          TEMPB0,WREG
        ADDWFC          ACCB0
SM2416NA#v(i) RLCF          TEMPB0,W
        RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        if i > 7
            RRCF          ACCB4
        endif
        i = i + 1
        endw
        RLCF          TEMPB0,W
        RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        RRCF          ACCB4

```

```

                                endm
UMUL2416      macro
;      Max Timing:      1+7*7*12+8*13 = 196 clks
;      Min Timing:      1+16*2+5 = 38 clks
;      PM: 1+16*2+5+7*7*12+8*13 = 233          DM: 10
                variable i

                i = 0

                BCF          _C
                while i < 16

                if i < 8
                BTFSC        BARGB1,i
                else
                BTFSC        BARGB0,i-8
                endif

                GOTO         UM2416NA#v(i)
                i = i + 1
                endw
                CLRF         ACCB0          ; if we get here, BARG = 0
                CLRF         ACCB1
                CLRF         ACCB2
                RETLW        0
UM2416NA0     RRCF         ACCB0
                RRCF         ACCB1
                RRCF         ACCB2
                RRCF         ACCB3
                i = 1
                while i < 16
                if i < 8
                BTFSS        BARGB1,i
                else
                BTFSS        BARGB0,i-8
                endif
                GOTO         UM2416NA#v(i)
UM2416A#v(i)  MOVFP        TEMPB2,WREG
                ADDWF        ACCB2
                MOVFP        TEMPB1,WREG
                ADDWFC        ACCB1
                MOVFP        TEMPB0,WREG
                ADDWFC        ACCB0
UM2416NA#v(i) RRCF         ACCB0
                RRCF         ACCB1
                RRCF         ACCB2
                RRCF         ACCB3
                if i > 7
                RRCF         ACCB4
                endif
                i = i + 1
                endw
                endm
UMUL2315      macro
;      Max Timing:      1+7*7*12+7*13+5 = 188 clks
;      Min Timing:      1+15*2+5 = 36 clks
;      PM: 1+15*2+5+7*7*2+7*13+5 = 223          DM: 10
                variable i

                i = 0

                BCF          _C
                while i < 15

                if i < 8
                BTFSC        BARGB1,i

```

AN617

```

    else
    BTFSC          BARGB0,i-8
    endif

    GOTO          UM2315NA#v(i)
    i = i + 1
    endw
    CLRF          ACCB0          ; if we get here, BARG = 0
    CLRF          ACCB1
    CLRF          ACCB2
    RETLW        0
UM2315NA0      RRCF          ACCB0
               RRCF          ACCB1
               RRCF          ACCB2
               RRCF          ACCB3
    i = 1
    while i < 15
    if i < 8
    BTFSS          BARGB1,i
    else
    BTFSS          BARGB0,i-8
    endif
    GOTO          UM2315NA#v(i)
UM2315A#v(i)  MOVFP          TEMPB2,WREG
               ADDWF        ACCB2
               MOVFP        TEMPB1,WREG
               ADDWFC       ACCB1
               MOVFP        TEMPB0,WREG
               ADDWFC       ACCB0
UM2315NA#v(i) RRCF          ACCB0
               RRCF          ACCB1
               RRCF          ACCB2
               RRCF          ACCB3
    if i > 7
    RRCF          ACCB4
    endif
    i = i + 1
    endw
    RRCF          ACCB0
    RRCF          ACCB1
    RRCF          ACCB2
    RRCF          ACCB3
    RRCF          ACCB4
    endm
;-----
SMUL2424      macro
;           Max Timing:      8+7*13+8*14+7*15+7 = 323 clks
;           Min Timing:      23*2+5 = 51 clks
;           PM: 23*2+9+7*13+8*14+7*15+7 = 370           DM: 12
               variable i
               i = 0
               while i < 23

               if i < 8
               BTFSC          BARGB2,i
               endif
               if (i >= 8) && (i < 16)
               BTFSC          BARGB1,i-8
               endif
               if (i >= 16)
               BTFSC          BARGB0,i-16
               endif
               GOTO          SM2424NA#v(i)
               i = i + 1
               endw
               CLRF          ACCB0          ; if we get here, BARG = 0
```

```

                CLRFB          ACCB1
                CLRFB          ACCB2
SM2424NA0      RETLW          0
                RLCFB          TEMPB0,W
                RRCFB          ACCB0
                RRCFB          ACCB1
                RRCFB          ACCB2
                RRCFB          ACCB3
                i = 1
                while i < 23
                if i < 8
                BTFSS          BARGB2,i
                endif
                if (i >= 8) && (i < 16)
                BTFSS          BARGB1,i-8
                endif
                if (i >= 16)
                BTFSS          BARGB0,i-16
                endif
SM2424A#v(i)   MOVFB          SM2424NA#v(i)
                MOVFB          TEMPB2,WREG
                ADDWF          ACCB2
                MOVFB          TEMPB1,WREG
                ADDWFC          ACCB1
                MOVFB          TEMPB0,WREG
                ADDWFC          ACCB0
SM2424NA#v(i) RLCFB          TEMPB0,W
                RRCFB          ACCB0
                RRCFB          ACCB1
                RRCFB          ACCB2
                RRCFB          ACCB3
                if i > 7
                RRCFB          ACCB4
                endif
                if i > 15
                RRCFB          ACCB5
                endif
                i = i + 1
                endw
                RLCFB          TEMPB0,W
                RRCFB          ACCB0
                RRCFB          ACCB1
                RRCFB          ACCB2
                RRCFB          ACCB3
                RRCFB          ACCB4
                RRCFB          ACCB5
                endm
UMUL2424      macro
;           Max Timing:      1+7*7*12+8*13+8*14 = 308 clks
;           Min Timing:      1+24*2+5 = 54 clks
;           PM: 1+24*2+8+7*12+8*13+8*14 = 357          DM: 12
                variable i
                i = 0

                BCF          _C
                while i < 24

                if i < 8
                BTFSC          BARGB2,i
                endif
                if (i >= 8) && (i < 16)
                BTFSC          BARGB1,i-8
                endif
                if (i >= 16)
                BTFSC          BARGB0,i-16
                endif

```

AN617

```
GOTO          UM2424NA#v(i)
i = i + 1
endw
CLRF          ACCB0          ; if we get here, BARG = 0
CLRF          ACCB1
CLRF          ACCB2
RETLW        0
UM2424NA0
RRCF          ACCB0
RRCF          ACCB1
RRCF          ACCB2
RRCF          ACCB3
i = 1
while i < 24
if i < 8
BTFSS        BARGB2,i
endif
if (i >= 8) && (i < 16)
BTFSS        BARGB1,i-8
endif
if (i >= 16)
BTFSS        BARGB0,i-16
endif
GOTO          UM2424NA#v(i)
UM2424A#v(i)
MOVFP        TEMPB2,WREG
ADDWF        ACCB2
MOVFP        TEMPB1,WREG
ADDWFC       ACCB1
MOVFP        TEMPB0,WREG
ADDWFC       ACCB0
UM2424NA#v(i)
RRCF          ACCB0
RRCF          ACCB1
RRCF          ACCB2
RRCF          ACCB3
if i > 7
RRCF          ACCB4
endif
if i > 15
RRCF          ACCB5
endif
i = i + 1
endw
endm
UMUL2323
macro
; Max Timing: 1+7*7*12+8*13+7*14+6 = 300 clks
; Min Timing: 1+23*2+5 = 52 clks
; PM: 1+23*2+8+7*12+8*13+7*14+6 = 347          DM: 12
variable i
i = 0

BCF          _C
while i < 23

if i < 8
BTFSC        BARGB2,i
endif
if (i >= 8) && (i < 16)
BTFSC        BARGB1,i-8
endif
if (i >= 16)
BTFSC        BARGB0,i-16
endif
GOTO          UM2323NA#v(i)
i = i + 1
endw
CLRF          ACCB0          ; if we get here, BARG = 0
CLRF          ACCB1
```

```

                CLRF          ACCB2
                RETLW        0
UM2323NA0      RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                i = 1
                while i < 23
                if i < 8
                BTFSS          BARGB2,i
                endif
                if (i >= 8) && (i < 16)
                BTFSS          BARGB1,i-8
                endif
                if (i >= 16)
                BTFSS          BARGB0,i-16
                endif
UM2323A#v(i)   GOTO          UM2323NA#v(i)
                MOVFP        TEMPB2,WREG
                ADDWF        ACCB2
                MOVFP        TEMPB1,WREG
                ADDWFC        ACCB1
                MOVFP        TEMPB0,WREG
                ADDWFC        ACCB0
UM2323NA#v(i)  RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                if i > 7
                RRCF          ACCB4
                endif
                if i > 15
                RRCF          ACCB5
                endif
                i = i + 1
                endw
                RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                RRCF          ACCB5
                endm
;-----
SMUL3216      macro
;      Max Timing:      9+7*16+7*17+7 = 247 clks
;      Min Timing:      15*2+6 = 36 clks
;      PM: 15*2+11+7*16+7*17+7 = 279          DM: 13
                variable i
                i = 0
                while i < 15

                if i < 8
                BTFSC          BARGB1,i
                endif
                if (i >= 8)
                BTFSC          BARGB0,i-8
                endif
                GOTO          SM3216NA#v(i)
                i = i + 1
                endw
                CLRF          ACCB0          ; if we get here, BARG = 0
                CLRF          ACCB1
                CLRF          ACCB2
                CLRF          ACCB3
                RETLW        0

```

AN617

```
SM3216NA0      RLCF          TEMPB0,W
                RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                i = 1
                while i < 15
                if i < 8
                BTFSS          BARGB1,i
                endif
                if (i >= 8)
                BTFSS          BARGB0,i-8
                endif
                GOTO          SM3216NA#v(i)
SM3216A#v(i)   MOVFP          TEMPB3,WREG
                ADDWF         ACCB3
                MOVFP         TEMPB2,WREG
                ADDWFC        ACCB2
                MOVFP         TEMPB1,WREG
                ADDWFC        ACCB1
                MOVFP         TEMPB0,WREG
                ADDWFC        ACCB0
SM3216NA#v(i)  RLCF          TEMPB0,W
                RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                if i > 7
                RRCF          ACCB5
                endif
                i = i + 1
                endw
                RLCF          TEMPB0,W
                RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                RRCF          ACCB5
                endm
UMUL3216      macro
;           Max Timing:      1+8+7*16+8*17 = 257 clks
;           Min Timing:      16*2+6 = 38 clks
;           PM: 1+16*2+10+7*16+8*17 = 291           DM: 13
                variable i
                i = 0
                BCF          _C
                while i < 16

                if i < 8
                BTFSC          BARGB1,i
                endif
                if (i >= 8)
                BTFSC          BARGB0,i-8
                endif
                GOTO          UM3216NA#v(i)
                i = i + 1
                endw
                CLRF          ACCB0           ; if we get here, BARG = 0
                CLRF          ACCB1
                CLRF          ACCB2
                CLRF          ACCB3
                RETLW         0
UM3216NA0     RRCF          ACCB0
```



```

                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                i = 1
                while i < 16
                if i < 8
                BTFSS          BARGB1,i
                endif
                if (i >= 8)
                BTFSS          BARGB0,i-8
                endif
                GOTO          UM3216NA#v(i)
UM3216A#v(i)    MOVFP          TEMPB3,WREG
                ADDWF          ACCB3
                MOVFP          TEMPB2,WREG
                ADDWFC          ACCB2
                MOVFP          TEMPB1,WREG
                ADDWFC          ACCB1
                MOVFP          TEMPB0,WREG
                ADDWFC          ACCB0
UM3216NA#v(i)  RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                if i > 7
                RRCF          ACCB5
                endif
                i = i + 1
                endw
                endm
UMUL3115      macro
;           Max Timing:      1+8+7*16+7*17+6 = 246 clks
;           Min Timing:      15*2+6 = 36 clks
;           PM: 1+15*2+10+7*16+7*17+6 = 278           DM: 13
                variable i
                i = 0
                BCF          _C
                while i < 15

                if i < 8
                BTFSC          BARGB1,i
                endif
                if (i >= 8)
                BTFSC          BARGB0,i-8
                endif
                GOTO          UM3115NA#v(i)
                i = i + 1
                endw
                CLRF          ACCB0           ; if we get here, BARG = 0
                CLRF          ACCB1
                CLRF          ACCB2
                CLRF          ACCB3
                RETLW          0
UM3115NA0     RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                i = 1
                while i < 15
                if i < 8
                BTFSS          BARGB1,i
                endif
                if (i >= 8)

```

AN617

```

        BTFSS          BARGB0,i-8
    endif
    GOTO              UM3115NA#v(i)
UM3115A#v(i)  MOVFP          TEMPB3,WREG
              ADDWF          ACCB3
              MOVFP          TEMPB2,WREG
              ADDWFC         ACCB2
              MOVFP          TEMPB1,WREG
              ADDWFC         ACCB1
              MOVFP          TEMPB0,WREG
              ADDWFC         ACCB0
UM3115NA#v(i) RRCF          ACCB0
              RRCF          ACCB1
              RRCF          ACCB2
              RRCF          ACCB3
              RRCF          ACCB4
              if i > 7
              RRCF          ACCB5
              endif
              i = i + 1
              endw
              RRCF          ACCB0
              RRCF          ACCB1
              RRCF          ACCB2
              RRCF          ACCB3
              RRCF          ACCB4
              RRCF          ACCB5
              endm
;-----
SMUL3224      macro
;      Max Timing:      9*7*16+8*17+7*18+8 = 391 clks
;      Min Timing:      23*2+6 = 52 clks
;      PM: 23*2+11+7*16+8*17+7*18+8 = 439          DM: 15
              variable i
              i = 0
              while i < 23

              if i < 8
              BTFSC          BARGB2,i
              endif
              if (i >= 8) && (i < 16)
              BTFSC          BARGB1,i-8
              endif
              if (i >= 16)
              BTFSC          BARGB0,i-16
              endif
              GOTO          SM3224NA#v(i)
              i = i + 1
              endw
              CLRF          ACCB0          ; if we get here, BARG = 0
              CLRF          ACCB1
              CLRF          ACCB2
              CLRF          ACCB3
SM3224NA0    RETLW          0
              RLCF          TEMPB0,W
              RRCF          ACCB0
              RRCF          ACCB1
              RRCF          ACCB2
              RRCF          ACCB3
              RRCF          ACCB4
              i = 1
              while i < 23
              if i < 8
              BTFSS          BARGB2,i
              endif
              if (i >= 8) && (i < 16)
```

```

                BTFSS          BARGB1,i-8
            endif
            if (i >= 16)
                BTFSS          BARGB0,i-16
            endif
SM3224A#v(i)   GOTO          SM3224NA#v(i)
                MOVFP         TEMPB3,WREG
                ADDWF         ACCB3
                MOVFP         TEMPB2,WREG
                ADDWFC        ACCB2
                MOVFP         TEMPB1,WREG
                ADDWFC        ACCB1
                MOVFP         TEMPB0,WREG
                ADDWFC        ACCB0
SM3224NA#v(i)  RLCF          TEMPB0,W
                RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                if i > 7
                RRCF          ACCB5
                endif
                if i > 15
                RRCF          ACCB6
                endif
                i = i + 1
            endw
                RLCF          TEMPB0,W
                RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2
                RRCF          ACCB3
                RRCF          ACCB4
                RRCF          ACCB5
                RRCF          ACCB6
            endm
UMUL3224      macro
;           Max Timing:      1+8*7*16+8*17+8*18 = 401 clks
;           Min Timing:      24*2+6 = 54 clks
;           PM: 1+24*2+10+7*16+8*17+8*18 = 451           DM: 15
                variable i
                i = 0
                BCF          _C
                while i < 24

                    if i < 8
                        BTFSC          BARGB2,i
                    endif
                    if (i >= 8) && (i < 16)
                        BTFSC          BARGB1,i-8
                    endif
                    if (i >= 16)
                        BTFSC          BARGB0,i-16
                    endif
                    GOTO          UM3224NA#v(i)
                    i = i + 1
                endw
                CLRF          ACCB0           ; if we get here, BARG = 0
                CLRF          ACCB1
                CLRF          ACCB2
                CLRF          ACCB3
                RETLW         0
UM3224NA0     RRCF          ACCB0
                RRCF          ACCB1
                RRCF          ACCB2

```

AN617

```

        RRCF          ACCB3
        RRCF          ACCB4
        i = 1
        while i < 24
        if i < 8
        BTFSS          BARGB2,i
        endif
        if (i >= 8) && (i < 16)
        BTFSS          BARGB1,i-8
        endif
        if (i >= 16)
        BTFSS          BARGB0,i-16
        endif
        GOTO          UM3224NA#v(i)
UM3224A#v(i)  MOVFP          TEMPB3,WREG
              ADDWF          ACCB3
              MOVFP          TEMPB2,WREG
              ADDWFC         ACCB2
              MOVFP          TEMPB1,WREG
              ADDWFC         ACCB1
              MOVFP          TEMPB0,WREG
              ADDWFC         ACCB0
UM3224NA#v(i) RRCF          ACCB0
              RRCF          ACCB1
              RRCF          ACCB2
              RRCF          ACCB3
              RRCF          ACCB4
              if i > 7
              RRCF          ACCB5
              endif
              if i > 15
              RRCF          ACCB6
              endif
              i = i + 1
        endw
        endm
UMUL3123    macro
;           Max Timing:      1+8+7*16+8*17+7*18+7 = 390 clks
;           Min Timing:      23*2+6 = 52 clks
;           PM: 1+23*2+10+7*16+8*17+7*18+7 = 438           DM: 15
              variable i
              i = 0
              BCF          _C
              while i < 23

              if i < 8
              BTFSC          BARGB2,i
              endif
              if (i >= 8) && (i < 16)
              BTFSC          BARGB1,i-8
              endif
              if (i >= 16)
              BTFSC          BARGB0,i-16
              endif
              GOTO          UM3123NA#v(i)
              i = i + 1
        endw
              CLRF          ACCB0           ; if we get here, BARG = 0
              CLRF          ACCB1
              CLRF          ACCB2
              CLRF          ACCB3
              RETLW         0
UM3123NA0   RRCF          ACCB0
              RRCF          ACCB1
              RRCF          ACCB2
              RRCF          ACCB3
```

```

        RRCF          ACCB4
        i = 1
        while i < 23
        if i < 8
        BTFSS          BARGB2,i
        endif
        if (i >= 8) && (i < 16)
        BTFSS          BARGB1,i-8
        endif
        if (i >= 16)
        BTFSS          BARGB0,i-16
        endif
        GOTO          UM3123NA#v(i)
UM3123A#v(i)  MOVFP          TEMPB3,WREG
              ADDWF          ACCB3
              MOVFP          TEMPB2,WREG
              ADDWFC         ACCB2
              MOVFP          TEMPB1,WREG
              ADDWFC         ACCB1
              MOVFP          TEMPB0,WREG
              ADDWFC         ACCB0
UM3123NA#v(i) RRCF          ACCB0
              RRCF          ACCB1
              RRCF          ACCB2
              RRCF          ACCB3
              RRCF          ACCB4
              if i > 7
              RRCF          ACCB5
              endif
              if i > 15
              RRCF          ACCB6
              endif
              i = i + 1
              endw
              RRCF          ACCB0
              RRCF          ACCB1
              RRCF          ACCB2
              RRCF          ACCB3
              RRCF          ACCB4
              RRCF          ACCB5
              RRCF          ACCB6
              endm
;-----
SMUL3232      macro
;      Max Timing:      9+7*16+8*17+8*18+7*19+9 = 543 clks
;      Min Timing:      31*2+6 = 68 clks
;      PM: 31*2+11+7*16+8*17+8*18+7*19+9 = 607          DM: 16
              variable i
              i = 0
              while i < 31

              if i < 8
              BTFSC          BARGB3,i
              endif
              if (i >= 8) && (i < 16)
              BTFSC          BARGB2,i-8
              endif
              if (i >= 16) && (i < 24)
              BTFSC          BARGB1,i-16
              endif
              if (i >= 24)
              BTFSC          BARGB0,i-24
              endif
              GOTO          SM3232NA#v(i)
              i = i + 1
              endw

```

AN617

```

        CLRF          ACCB0          ; if we get here, BARG = 0
        CLRF          ACCB1
        CLRF          ACCB2
        CLRF          ACCB3
        RETLW         0
SM3232NA0  RLCF          TEMPB0,W
        RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        RRCF          ACCB4
        i = 1
        while i < 31
        if i < 8
        BTFSS         BARGB3,i
        endif
        if (i >= 8) && (i < 16)
        BTFSS         BARGB2,i-8
        endif
        if (i >= 16) && (i < 24)
        BTFSS         BARGB1,i-16
        endif
        if (i >= 24)
        BTFSS         BARGB0,i-24
        endif
        GOTO          SM3232NA#v(i)
SM3232A#v(i) MOVFP          TEMPB3,WREG
        ADDWF         ACCB3
        MOVFP          TEMPB2,WREG
        ADDWFC        ACCB2
        MOVFP          TEMPB1,WREG
        ADDWFC        ACCB1
        MOVFP          TEMPB0,WREG
        ADDWFC        ACCB0
SM3232NA#v(i) RLCF          TEMPB0,W
        RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        RRCF          ACCB4
        if i > 7
        RRCF          ACCB5
        endif
        if i > 15
        RRCF          ACCB6
        endif
        if i > 23
        RRCF          ACCB7
        endif
        i = i + 1
        endw
        RLCF          TEMPB0,W
        RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        RRCF          ACCB4
        RRCF          ACCB5
        RRCF          ACCB6
        RRCF          ACCB7
        endm
UMUL3232  macro
;      Max Timing:    1+8+7*16+8*17+8*18+8*19 = 553 clks
;      Min Timing:    32*2+6 = 70 clks
;      PM: 1+32*2+10+7*16+8*17+8*18+8*19 = 619          DM: 16
        variable i
```

```

i = 0
BCF          _C
while i < 32

    if i < 8
    BTFSC     BARGB3,i
    endif
    if (i >= 8) && (i < 16)
    BTFSC     BARGB2,i-8
    endif
    if (i >= 16) && (i < 24)
    BTFSC     BARGB1,i-16
    endif
    if (i >= 24)
    BTFSC     BARGB0,i-24
    endif
    GOTO      UM3232NA#v(i)
    i = i + 1
endw
CLRF        ACCB0          ; if we get here, BARG = 0
CLRF        ACCB1
CLRF        ACCB2
CLRF        ACCB3
RETLW      0
UM3232NA0   RRCF          ACCB0
            RRCF          ACCB1
            RRCF          ACCB2
            RRCF          ACCB3
            RRCF          ACCB4
i = 1
while i < 32
    if i < 8
    BTFSS     BARGB3,i
    endif
    if (i >= 8) && (i < 16)
    BTFSS     BARGB2,i-8
    endif
    if (i >= 16) && (i < 24)
    BTFSS     BARGB1,i-16
    endif
    if (i >= 24)
    BTFSS     BARGB0,i-24
    endif
    GOTO      UM3232NA#v(i)
UM3232A#v(i) MOVFP      TEMPB3,WREG
            ADDWF      ACCB3
            MOVFP      TEMPB2,WREG
            ADDWFC     ACCB2
            MOVFP      TEMPB1,WREG
            ADDWFC     ACCB1
            MOVFP      TEMPB0,WREG
            ADDWFC     ACCB0
UM3232NA#v(i) RRCF          ACCB0
            RRCF          ACCB1
            RRCF          ACCB2
            RRCF          ACCB3
            RRCF          ACCB4
            if i > 7
            RRCF          ACCB5
            endif
            if i > 15
            RRCF          ACCB6
            endif
            if i > 23
            RRCF          ACCB7
            endif

```

AN617

```
        i = i + 1
        endw
    endm
UMUL3131    macro
;           Max Timing:      1+8+7*16+8*17+8*18+7*19+9 = 533 clks
;           Min Timing:      31*2+6 = 68 clks
;           PM: 1+31*2+10+7*16+8*17+8*18+7*19+9 = 597           DM: 16
        variable i
        i = 0
        BCF          _C
        while i < 31

            if i < 8
                BTFSC          BARGB3,i
            endif
            if (i >= 8) && (i < 16)
                BTFSC          BARGB2,i-8
            endif
            if (i >= 16) && (i < 24)
                BTFSC          BARGB1,i-16
            endif
            if (i >= 24)
                BTFSC          BARGB0,i-24
            endif
            GOTO          UM3131NA#v(i)
            i = i + 1
        endw
        CLRF          ACCB0           ; if we get here, BARG = 0
        CLRF          ACCB1
        CLRF          ACCB2
        CLRF          ACCB3
        RETLW         0
UM3131NA0    RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        RRCF          ACCB4
        i = 1
        while i < 31
            if i < 8
                BTFSS          BARGB3,i
            endif
            if (i >= 8) && (i < 16)
                BTFSS          BARGB2,i-8
            endif
            if (i >= 16) && (i < 24)
                BTFSS          BARGB1,i-16
            endif
            if (i >= 24)
                BTFSS          BARGB0,i-24
            endif
            GOTO          UM3131NA#v(i)
UM3131A#v(i) MOVFP          TEMPB3,WREG
        ADDWF          ACCB3
        MOVFP          TEMPB2,WREG
        ADDWFC         ACCB2
        MOVFP          TEMPB1,WREG
        ADDWFC         ACCB1
        MOVFP          TEMPB0,WREG
        ADDWFC         ACCB0
UM3131NA#v(i) RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        RRCF          ACCB4
        if i > 7
```



```

                RRCF          ACCB5
            endif
            if i > 15
                RRCF          ACCB6
            endif
            if i > 23
                RRCF          ACCB7
            endif
            i = i + 1
        endw
        RRCF          ACCB0
        RRCF          ACCB1
        RRCF          ACCB2
        RRCF          ACCB3
        RRCF          ACCB4
        RRCF          ACCB5
        RRCF          ACCB6
        RRCF          ACCB7
    endm
;*****
;*****
;
;      8x8 Bit Signed Fixed Point Multiply 08 x 08 -> 16
;      Input:  8 bit signed fixed point multiplicand in AARGB0
;              8 bit signed fixed point multiplier in BARGB0
;      Use:    CALL      FXM0808S
;      Output: 16 bit signed fixed point product in AARGB0, AARGB1
;      Result: AARG <-- AARG * BARG
;      Max Timing:      5+43+2 = 50 clks          B > 0
;                      8+43+2 = 53 clks          B < 0
;      Min Timing:      5+21 = 26 clks
;      PM: 8+56+1 = 65          DM: 5
FXM0808S      BTFSS          BARGB0,MSB
              GOTO          M0808SOK
              COMF          BARGB0          ; make multiplier BARG > 0
              INCF          BARGB0
              COMF          AARGB0
              INCF          AARGB0
M0808SOK      CLRF          ACCB1          ; clear partial product
              MOVVPF        AARGB0,TEMPB0
              SMUL0808
              RETLW         0x00
;*****
;*****
;
;      8x8 Bit Unsigned Fixed Point Multiply 08 x 08 -> 16
;      Input:  8 bit unsigned fixed point multiplicand in AARGB0
;              8 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL      FXM0808U
;      Output: 16 bit unsigned fixed point product in AARGB0, AARGB1
;      Result: AARG <-- AARG * BARG
;      Max Timing:      2+35+2 = 39 clks
;      Min Timing:      2+21 = 23 clks
;      PM: 2+50+1 = 53          DM: 3
FXM0808U      CLRF          ACCB1          ; clear partial product
              MOVVPF        AARGB0,TEMPB0
              UMUL0808
              RETLW         0x00
;*****
;*****
;
;      7x7 Bit Unsigned Fixed Point Multiply 07 x 07 -> 14
;      Input:  7 bit unsigned fixed point multiplicand in AARGB0
;              7 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL      FXM0707U
;      Output: 14 bit unsigned fixed point product in AARGB0, AARGB1

```

AN617

```
;      Result: AARG <-- AARG * BARG
;      Max Timing:      2+33+2 = 37 clks
;      Min Timing:      2+19 = 21 clks
;      PM: 2+46+1 = 49          DM: 3
FXM0707U      CLRF          ACCB1          ; clear partial product
              MOVVPF        AARGB0,TEMPB0
              UMUL0707
              RETLW         0x00
;*****
;*****

;      16x8 Bit Signed Fixed Point Multiply 16 x 08 -> 24
;      Input:  16 bit signed fixed point multiplicand in AARGB0
;              8 bit signed fixed point multiplier in BARGB0
;      Use:    CALL      FXM1608S
;      Output: 24 bit signed fixed point product in AARGB0, AARGB1
;      Result: AARG <-- AARG * BARG
;      Max Timing:      6+66+2 = 74 clks          B > 0
;                  11+66+2 = 79 clks          B < 0
;      Min Timing:      6+29 = 35 clks
;      PM: 11+88+1 = 100          DM: 6
FXM1608S      BTFSS        BARGB0,MSB
              GOTO         M1608SOK
              COMF         BARGB0          ; make multiplier BARG > 0
              INCF         BARGB0
              COMF         AARGB0
              COMF         AARGB1
              INFSNZ       AARGB1
              INCF         AARGB0
M1608SOK      CLRF          ACCB2          ; clear partial product
              MOVVPF        AARGB0,TEMPB0
              MOVVPF        AARGB1,TEMPB1
              SSMUL1608
              RETLW         0x00
;*****
;*****

;      16x8 Bit Unsigned Fixed Point Multiply 16 x 08 -> 24
;      Input:  16 bit unsigned fixed point multiplicand in AARGB0
;              8 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL      FXM1608U
;      Output: 24 bit unsigned fixed point product in AARGB0, AARGB1, AARGB2
;      Result: AARG <-- AARG * BARG
;      Max Timing:      3+70+2 = 75 clks
;      Min Timing:      3+21 = 24 clks
;      PM: 3+86+1 = 90          DM: 6
FXM1608U      CLRF          ACCB2
              MOVVPF        AARGB0,TEMPB0
              MOVVPF        AARGB1,TEMPB1
              UMUL1608
              RETLW         0x00
;*****
;*****

;      15x7 Bit Unsigned Fixed Point Multiply 15 x 07 -> 22
;      Input:  15 bit unsigned fixed point multiplicand in AARGB0,AARGB1
;              7 bit unsigned fixed point multiplier in BARGB0
;      Use:    CALL      FXM1507U
;      Output: 22 bit unsigned fixed point product in AARGB0, AARGB1, AARGB2
;      Result: AARG <-- AARG * BARG
;      Max Timing:      3+64+2 = 69 clks
;      Min Timing:      3+21 = 24 clks
;      PM: 3+78+1 = 82          DM: 6
FXM1507U      CLRF          ACCB2
              MOVVPF        AARGB0,TEMPB0
              MOVVPF        AARGB1,TEMPB1
```

```

UMUL1507
RETLW      0x00
;*****
;*****

;      16x16 Bit Signed Fixed Point Multiply 16 x 16 -> 32
;      Input:  16 bit signed fixed point multiplicand in AARGB0, AARGB1
;              16 bit signed fixed point multiplier in BARGB0, BARGB1
;      Use:    CALL   FXM1616S
;      Output: 32 bit signed fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3
;      Result: AARG <-- AARG * BARG
;      Max Timing:      7+159+2 = 168 clks          B > 0
;                      14+159+2 = 175 clks         B < 0
;      Min Timing:     6+34 = 24 clks
;      PM: 11+193+1 = 97          DM: 8
FXM1616S   BTFSS      BARGB0,MSB
           GOTO       M1616SOK
           COMF       BARGB0      ; make multiplier BARG > 0
           COMF       BARGB1
           INFSNZ     BARGB1
           INCF       BARGB0
           COMF       AARGB0
           COMF       AARGB1
           INFSNZ     AARGB1
           INCF       AARGB0
M1616SOK   CLRF       ACCB2      ; clear partial product
           CLRF       ACCB3
           MOVVPF     AARGB0,TEMPB0
           MOVVPF     AARGB1,TEMPB1
           SMUL1616
           RETLW      0x00
;*****
;*****

;      16x16 Bit Unsigned Fixed Point Multiply 16 x 16 -> 32
;      Input:  16 bit unsigned fixed point multiplicand in AARGB0, AARGB1
;              16 bit unsigned fixed point multiplier in BARGB0, BARGB1
;      Use:    CALL   FXM1616U
;      Output: 32 bit unsigned fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3
;      Result: AARG <-- AARG * BARG
;      Max Timing:      4+150+2 = 156 clks
;      Min Timing:     4+37 = 41 clks
;      PM: 4+186+1 = 191          DM: 8
FXM1616U   CLRF       ACCB2
           CLRF       ACCB3
           MOVVPF     AARGB0,TEMPB0
           MOVVPF     AARGB1,TEMPB1
           UMUL1616
           RETLW      0x00
;*****
;*****

;      15x15 Bit Unsigned Fixed Point Multiply 15 x 15 -> 30
;      Input:  15 bit unsigned fixed point multiplicand in AARGB0, AARGB1
;              15 bit unsigned fixed point multiplier in BARGB0, BARGB1
;      Use:    CALL   FXM1515U
;      Output: 30 bit unsigned fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3
;      Result: AARG <-- AARG * BARG
;      Max Timing:      4+144+2 = 150 clks
;      Min Timing:     4+35 = 39 clks
;      PM: 4+180+1 = 185          DM: 8
FXM1515U   CLRF       ACCB2
           CLRF       ACCB3

```

AN617

```
MOVPF          AARGB0,TEMPB0
MOVPF          AARGB1,TEMPB1
UMUL1515
RETLW         0x00
;*****
;*****

;      24x16 Bit Signed Fixed Point Multiply 24 x 16 -> 40
;      Input:  24 bit signed fixed point multiplicand in AARGB0, AARGB1,
;              AARGB2
;              16 bit signed fixed point multiplier in BARGB0, BARGB1
;      Use:    CALL    FXM2416S
;      Output: 40 bit signed fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3, AARGB4
;      Result: AARG <-- AARG * BARG
;      Max Timing:      8+203+2 = 213 clks           B > 0
;                      18+203+2 = 223 clks           B < 0
;      Min Timing:     8+35 = 43 clks
;      PM: 18+234+1 = 253           DM: 10
FXM2416S      BTFSS          BARGB0,MSB
              GOTO          M2416SOK
              COMF          BARGB0           ; make multiplier BARG > 0
              COMF          BARGB1
              INFSNZ        BARGB1
              INCF          BARGB0
              CLRF          WREG
              COMF          AARGB0
              COMF          AARGB1
              COMF          AARGB2
              INCF          AARGB2
              ADDWFC        AARGB1
              ADDWFC        AARGB0
M2416SOK      CLRF          ACCB3           ; clear partial product
              CLRF          ACCB4
              MOVPF         AARGB0,TEMPB0
              MOVPF         AARGB1,TEMPB1
              MOVPF         AARGB2,TEMPB2
              SMUL2416
              RETLW         0x00
;*****
;*****

;      24x16 Bit Unsigned Fixed Point Multiply 24 x 16 -> 40
;      Input:  24 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;              AARGB2
;              16 bit unsigned fixed point multiplier in BARGB0, BARGB1
;      Use:    CALL    FXM2416U
;      Output: 40 bit unsigned fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3, AARGB4
;      Result: AARG <-- AARG * BARG
;      Max Timing:     5+196+2 = 203 clks
;      Min Timing:     5+38 = 43 clks
;      PM: 5+233+1 = 239           DM: 10
FXM2416U      CLRF          ACCB3
              CLRF          ACCB4
              MOVPF         AARGB0,TEMPB0
              MOVPF         AARGB1,TEMPB1
              MOVPF         AARGB2,TEMPB2
              UMUL2416
              RETLW         0x00
;*****
;*****

;      23x15 Bit Unsigned Fixed Point Multiply 23 x 15 -> 38
;      Input:  23 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;              AARGB2
```

```

;          15 bit unsigned fixed point multiplier in BARGB0, BARGB1
; Use:    CALL    FXM2315U
; Output: 38 bit unsigned fixed point product in AARGB0, AARGB1,
;         AARGB2, AARGB3, AARGB4
; Result: AARG <-- AARG * BARG
; Max Timing:    4+188+2 = 194 clks
; Min Timing:    5+36 = 41 clks
; PM: 5+223+1 = 229          DM: 10
FXM2315U    CLRF          ACCB3
            CLRF          ACCB4
            MOVVPF        AARGB0,TEMPB0
            MOVVPF        AARGB1,TEMPB1
            MOVVPF        AARGB2,TEMPB2
            UMUL2315
            RETLW         0x00
;*****
;*****

;          24x24 Bit Signed Fixed Point Multiply 24 x 24 -> 48
; Input:   24 bit signed fixed point multiplicand in AARGB0, AARGB1,
;         AARGB2
;         24 bit signed fixed point multiplier in BARGB0, BARGB1,
;         BARGB2
; Use:    CALL    FXM2424S
; Output: 48 bit signed fixed point product in AARGB0, AARGB1,
;         AARGB2, AARGB3, AARGB4, AARGB5
; Result: AARG <-- AARG * BARG
; Max Timing:    9+323+2 = 334 clks          B > 0
;             21+323+2 = 346 clks          B < 0
; Min Timing:    9+51 = 60 clks
; PM: 21+370+1 = 392          DM: 12
FXM2424S    BTFSS        BARGB0,MSB
            GOTO         M2424SOK
            CLRF          WREG
            COMF          BARGB0          ; make multiplier BARG > 0
            COMF          BARGB1
            COMF          BARGB2
            INCF          BARGB2
            ADDWFC        BARGB1
            ADDWFC        BARGB0
            COMF          AARGB0
            COMF          AARGB1
            COMF          AARGB2
            INCF          AARGB2
            ADDWFC        AARGB1
            ADDWFC        AARGB0
M2424SOK    CLRF          ACCB3          ; clear partial product
            CLRF          ACCB4
            CLRF          ACCB5
            MOVVPF        AARGB0,TEMPB0
            MOVVPF        AARGB1,TEMPB1
            MOVVPF        AARGB2,TEMPB2
            SMUL2424
            RETLW         0x00
;*****
;*****

;          24x24 Bit Unsigned Fixed Point Multiply 24 x 24 -> 48
; Input:   24 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;         AARGB2
;         24 bit unsigned fixed point multiplier in BARGB0, BARGB1,
;         BARGB2
; Use:    CALL    FXM2424U
; Output: 48 bit unsigned fixed point product in AARGB0, AARGB1,
;         AARGB2, AARGB3, AARGB4, AARGB5
; Result: AARG <-- AARG * BARG

```

AN617

```
;      Max Timing:      6+308+2 = 316 clks
;      Min Timing:      6+54 = 60 clks
;      PM: 6+357+1 = 364          DM: 12
FXM2424U      CLRF          ACCB3
              CLRF          ACCB4
              CLRF          ACCB5
              MOVPF         AARGB0,TEMPB0
              MOVPF         AARGB1,TEMPB1
              MOVPF         AARGB2,TEMPB2
              UMUL2424
              RETLW         0x00
;*****
;*****

;      23x23 Bit Unsigned Fixed Point Multiply 23 x 23 -> 46
;      Input:  23 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;              AARGB2
;              23 bit unsigned fixed point multiplier in BARGB0, BARGB1,
;              BARGB2
;      Use:    CALL      FXM2323U
;      Output: 46 bit unsigned fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3, AARGB4, AARGB5
;      Result: AARG <-- AARG * BARG
;      Max Timing:      6+300+2 = 308 clks
;      Min Timing:      6+52 = 58 clks
;      PM: 6+347+1 = 354          DM: 12
FXM2323U      CLRF          ACCB3
              CLRF          ACCB4
              CLRF          ACCB5
              MOVPF         AARGB0,TEMPB0
              MOVPF         AARGB1,TEMPB1
              MOVPF         AARGB2,TEMPB2
              UMUL2323
              RETLW         0x00
;*****
;*****

;      32x16 Bit Signed Fixed Point Multiply 32 x 16 -> 48
;      Input:  32 bit signed fixed point multiplicand in AARGB0, AARGB1,
;              AARGB2, AARGB3
;              16 bit signed fixed point multiplier in BARGB0, BARGB1
;      Use:    CALL      FXM3216S
;      Output: 48 bit signed fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3, AARGB4, AARGB5
;      Result: AARG <-- AARG * BARG
;      Max Timing:      9+247+2 = 258 clks          B > 0
;              21+247+2 = 270 clks          B < 0
;      Min Timing:      10+36 = 46 clks
;      PM: 21+279+1 = 301          DM: 13
FXM3216S      BTFSS         BARGB0,MSB
              GOTO          M3216SOK
              CLRF          WREG
              COMF          BARGB0          ; make multiplier BARG > 0
              COMF          BARGB1
              INCF          BARGB1
              ADDWFC        BARGB0
              COMF          AARGB0
              COMF          AARGB1
              COMF          AARGB2
              COMF          AARGB3
              INCF          AARGB3
              ADDWFC        AARGB2
              ADDWFC        AARGB1
              ADDWFC        AARGB0
M3216SOK      CLRF          ACCB4          ; clear partial product
              CLRF          ACCB5
```

```

MOVPF          AARGB0,TEMPB0
MOVPF          AARGB1,TEMPB1
MOVPF          AARGB2,TEMPB2
MOVPF          AARGB3,TEMPB3
SMUL3216
RETLW         0x00
;*****
;*****
;
;   32x16 Bit Unsigned Fixed Point Multiply 32 x 16 -> 48
;   Input:  32 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;           AARGB2, AARGB3
;           16 bit unsigned fixed point multiplier in BARGB0, BARGB1
;   Use:    CALL    FXM3216U
;   Output: 48 bit unsigned fixed point product in AARGB0, AARGB1,
;           AARGB2, AARGB3, AARGB4, AARGB5
;   Result: AARG <-- AARG * BARG
;   Max Timing: 6+257+2 = 265 clks
;   Min Timing: 6+38 = 44 clks
;   PM: 6+291+1 = 298          DM: 13
FXM3216U      CLRF          ACCB4
              CLRF          ACCB5
              MOVPF        AARGB0,TEMPB0
              MOVPF        AARGB1,TEMPB1
              MOVPF        AARGB2,TEMPB2
              MOVPF        AARGB3,TEMPB3
              UMUL3216
              RETLW        0x00
;*****
;*****
;
;   31x15 Bit Unsigned Fixed Point Multiply 31 x 15 -> 46
;   Input:  31 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;           AARGB2, AARGB3
;           15 bit unsigned fixed point multiplier in BARGB0, BARGB1
;   Use:    CALL    FXM3115U
;   Output: 46 bit unsigned fixed point product in AARGB0, AARGB1,
;           AARGB2, AARGB3, AARGB4, AARGB5, AARGB6
;   Result: AARG <-- AARG * BARG
;   Max Timing: 6+246+2 = 254 clks
;   Min Timing: 6+36 = 42 clks
;   PM: 6+278+1 = 285          DM: 13
FXM3115U      CLRF          ACCB4
              CLRF          ACCB5
              MOVPF        AARGB0,TEMPB0
              MOVPF        AARGB1,TEMPB1
              MOVPF        AARGB2,TEMPB2
              MOVPF        AARGB3,TEMPB3
              UMUL3115
              RETLW        0x00
;*****
;*****
;
;   32x24 Bit Signed Fixed Point Multiply 32 x 24 -> 56
;   Input:  32 bit signed fixed point multiplicand in AARGB0, AARGB1,
;           AARGB2, AARGB3
;           24 bit signed fixed point multiplier in BARGB0, BARGB1,
;           BARGB2
;   Use:    CALL    FXM3224S
;   Output: 56 bit signed fixed point product in AARGB0, AARGB1,
;           AARGB2, AARGB3, AARGB4, AARGB5, AARGB6
;   Result: AARG <-- AARG * BARG
;   Max Timing: 10+391+2 = 403 clks          B > 0
;           24+391+2 = 417 clks          B < 0
;   Min Timing: 10+52 = 62 clks
;   PM: 24+439+1 = 464          DM: 15

```

AN617

```
FXM3224S      BTFSS          BARGB0,MSB
               GOTO          M3224SOK
               CLRF          WREG
               COMF          BARGB0          ; make multiplier BARG > 0
               COMF          BARGB1
               COMF          BARGB2
               INCF          BARGB2
               ADDWFC        BARGB1
               ADDWFC        BARGB0
               COMF          AARGB0
               COMF          AARGB1
               COMF          AARGB2
               COMF          AARGB3
               INCF          AARGB3
               ADDWFC        AARGB2
               ADDWFC        AARGB1
               ADDWFC        AARGB0
M3224SOK      CLRF          ACCB4          ; clear partial product
               CLRF          ACCB5
               CLRF          ACCB6
               MOVPF         AARGB0,TEMPB0
               MOVPF         AARGB1,TEMPB1
               MOVPF         AARGB2,TEMPB2
               MOVPF         AARGB3,TEMPB3
               SMUL3224
               RETLW         0x00
;*****
;*****
;
;      32x24 Bit Unsigned Fixed Point Multiply 32 x 24 -> 56
;      Input:  32 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;              AARGB2, AARGB3
;              24 bit unsigned fixed point multiplier in BARGB0, BARGB1,
;              BARGB2
;      Use:    CALL      FXM3224U
;      Output: 56 bit unsigned fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3, AARGB4, AARGB5, AARGB6
;      Result: AARG <-- AARG * BARG
;      Max Timing: 7+401+2 = 410 clks
;      Min Timing: 7+54 = 61 clks
;      PM: 7+451+1 = 459          DM: 15
FXM3224U      CLRF          ACCB4
               CLRF          ACCB5
               CLRF          ACCB6
               MOVPF         AARGB0,TEMPB0
               MOVPF         AARGB1,TEMPB1
               MOVPF         AARGB2,TEMPB2
               MOVPF         AARGB3,TEMPB3
               UMUL3224
               RETLW         0x00
;*****
;*****
;
;      31x23 Bit Unsigned Fixed Point Multiply 31 x 23 -> 54
;      Input:  31 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;              AARGB2, AARGB3
;              23 bit unsigned fixed point multiplier in BARGB0, BARGB1,
;              BARGB2
;      Use:    CALL      FXM3123U
;      Output: 54 bit unsigned fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3, AARGB4, AARGB5, AARGB6
;      Result: AARG <-- AARG * BARG
;      Max Timing: 7+390+2 = 399 clks
;      Min Timing: 7+52 = 59 clks
;      PM: 7+438+1 = 446          DM: 15
FXM3123U      CLRF          ACCB4
```



```

        CLRF          ACCB5
        CLRF          ACCB6
        MOVPF        AARGB0,TEMPB0
        MOVPF        AARGB1,TEMPB1
        MOVPF        AARGB2,TEMPB2
        MOVPF        AARGB3,TEMPB3
        UMUL3123
        RETLW        0x00
;*****
;*****
;
;   32x32 Bit Signed Fixed Point Multiply 32 x 32 -> 64
;   Input:  32 bit signed fixed point multiplicand in AARGB0, AARGB1,
;           AARGB2, AARGB3
;           32 bit signed fixed point multiplier in BARGB0, BARGB1,
;           BARGB2, BARGB3
;   Use:    CALL    FXM3232S
;   Output: 64 bit signed fixed point product in AARGB0, AARGB1,
;           AARGB2, AARGB3, AARGB4, AARGB5, AARGB6, AARGB7
;   Result: AARG <-- AARG * BARG
;   Max Timing:  11+543+2 = 556 clks          B > 0
;               27+543+2 = 572 clks          B < 0
;   Min Timing:  11+68 = 79 clks
;   PM: 27+607+1 = 635          DM: 16
FXM3232S    BTFS    BARGB0,MSB
            GOTO    M3232SOK
            CLRF    WREG
            COMF    BARGB0          ; make multiplier BARG > 0
            COMF    BARGB1
            COMF    BARGB2
            COMF    BARGB3
            INCF    BARGB3
            ADDWFC  BARGB2
            ADDWFC  BARGB1
            ADDWFC  BARGB0
            COMF    AARGB0
            COMF    AARGB1
            COMF    AARGB2
            COMF    AARGB3
            INCF    AARGB3
            ADDWFC  AARGB2
            ADDWFC  AARGB1
            ADDWFC  AARGB0
M3232SOK    CLRF    ACCB4          ; clear partial product
            CLRF    ACCB5
            CLRF    ACCB6
            CLRF    ACCB7
            MOVPF  AARGB0,TEMPB0
            MOVPF  AARGB1,TEMPB1
            MOVPF  AARGB2,TEMPB2
            MOVPF  AARGB3,TEMPB3
            SMUL3232
            RETLW  0x00
;*****
;*****
;
;   32x32 Bit Unsigned Fixed Point Multiply 32 x 32 -> 64
;   Input:  32 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;           AARGB2, AARGB3
;           32 bit unsigned fixed point multiplier in BARGB0, BARGB1,
;           BARGB2, BARGB3
;   Use:    CALL    FXM3232U
;   Output: 64 bit unsigned fixed point product in AARGB0, AARGB1,
;           AARGB2, AARGB3, AARGB4, AARGB5, AARGB6, AARGB7
;   Result: AARG <-- AARG * BARG
;   Max Timing:  8+553+2 = 563 clks

```

AN617

```
;      Min Timing:      8+70 = 78 clks
;      PM: 8+619+1 = 628          DM: 16
FXM3232U      CLRF          ACCB4
              CLRF          ACCB5
              CLRF          ACCB6
              CLRF          ACCB7
              MOVPF         AARGB0,TEMPB0
              MOVPF         AARGB1,TEMPB1
              MOVPF         AARGB2,TEMPB2
              MOVPF         AARGB3,TEMPB3
              UMUL3232
              RETLW         0x00
;*****
;*****

;      31x31 Bit Unsigned Fixed Point Multiply 31 x 31 -> 62
;      Input:  31 bit unsigned fixed point multiplicand in AARGB0, AARGB1,
;              AARGB2, AARGB3
;      31 bit unsigned fixed point multiplier in BARGB0, BARGB1,
;              BARGB2, BARGB3
;      Use:    CALL    FXM3131U
;      Output: 62 bit unsigned fixed point product in AARGB0, AARGB1,
;              AARGB2, AARGB3, AARGB4, AARGB5, AARGB6, AARGB7
;      Result: AARG <-- AARG * BARG
;      Max Timing:      8+533+2 = 543 clks
;      Min Timing:      8+68 = 76 clks
;      PM: 8+597+1 = 606          DM: 16
FXM3131U      CLRF          ACCB4
              CLRF          ACCB5
              CLRF          ACCB6
              CLRF          ACCB7
              MOVPF         AARGB0,TEMPB0
              MOVPF         AARGB1,TEMPB1
              MOVPF         AARGB2,TEMPB2
              MOVPF         AARGB3,TEMPB3
              UMUL3131
              RETLW         0x00
;*****
;*****

      END
```

NOTES:

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX F: PIC17CXX DIVIDE ROUTINES

Table of Contents for Appendix F

F.1	PIC17CXX Fixed Point Divide Routines A.....	2-386
F.2	PIC17CXX Fixed Point Divide Routines B.....	2-421
F.3	PIC17CXX Fixed Point Divide Routines C	2-450

F.1 PIC17CXX Fixed Point Divide Routines A

```

; PIC17 FIXED POINT DIVIDE ROUTINES A    VERSION 1.8
; Input:  fixed point arguments in AARG and BARG
; Output: quotient AARG/BARG followed by remainder in REM
; All timings are worst case cycle counts
; It is useful to note that the additional unsigned routines requiring a non-power of two
; argument can be called in a signed divide application where it is known that the
; respective argument is nonnegative, thereby offering some improvement in
; performance.
;
; Routine           Clocks      Function
; FXD3232S          614 32 bit/32 bit -> 32.32 signed fixed point divide
; FXD3232U          683 32 bit/32 bit -> 32.32 unsigned fixed point divide
; FXD3231U          588 32 bit/31 bit -> 32.31 unsigned fixed point divide
; FXD3131U          579 31 bit/31 bit -> 31.31 unsigned fixed point divide
; FXD3224S          514 32 bit/24 bit -> 32.24 signed fixed point divide
; FXD3224U          584 32 bit/24 bit -> 32.24 unsigned fixed point divide
; FXD3223U          489 32 bit/23 bit -> 32.23 unsigned fixed point divide
; FXD3123U          481 31 bit/23 bit -> 31.23 unsigned fixed point divide
;
; list  r=dec,x=on,t=off,p=17C42
; include <PIC17.INC>           ; general PIC17 definitions
;
; include <MATH17.INC>         ; PIC17 math library definitions
;*****
;*****
; Test suite storage
RANDHI      equ    0x2B      ; random number generator registers
RANDLO      equ    0x2C
TESTCODE    equ    0x2D      ; integer code labeling test contained in following data
NUMTESTS    equ    0x2E      ; number of tests contained in following data
TESTCOUNT  equ    0x2F      ; counter
DATA        equ    0x30      ; beginning of test data
;*****
;*****
; Test suite for fixed point divide algorithms
;
; org 0x0021
MAIN        MOVLW    RAMSTART
            MOVPF    WREG,FSR0
MEMLOOP     CLRF     INDF0
            INCFSZ   FSR0
            GOTO    MEMLOOP
            BSF     RTCSTA,5
;
            MOVPF   WREG,WREG
            MOVLW   0x45          ; seed for random numbers
            MOVPF   WREG,RANDLO
;
            MOVPF   RTCCL,WREG
            MOVLW   0x30
            MOVPF   WREG,RANDHI
            MOVLW   0x30
            MOVPF   WREG,FSR0
            BCF     _FS1
            BSF     _FS0
            CALL    TFXD3232
;
            CALL    TFXD3224
SELF        GOTO    SELF
RANDOM16     RLCF     RANDHI,W          ; random number generator

```

```

XORWF      RANDHI ,W
RLCF      WREG
SWAPF     RANDHI
SWAPF     RANDLO ,W
RLNCF     WREG
XORWF     RANDHI ,W
SWAPF     RANDHI
ANDLW     0x01
RLCF      RANDLO
XORWF     RANDLO
RLCF      RANDHI

      RETLW      0
;      Test suite for FXD3232
TFXD3232      MOVLW      13
              MOVFPF     WREG ,TESTCOUNT
              MOVFPF     WREG ,NUMTESTS
              MOVLW      5
              MOVFPF     WREG ,TESTCODE

D3232LOOP
      CALL      RANDOM16
      MOVFPF     RANDHI ,WREG
      MOVFPF     WREG ,BARGB0
      MOVFPF     RANDLO ,WREG
      MOVFPF     WREG ,BARGB1
      CALL      RANDOM16
      MOVFPF     RANDHI ,WREG
      MOVFPF     WREG ,BARGB2
      MOVFPF     RANDLO ,WREG
      MOVFPF     WREG ,BARGB3
;      BCF      BARGB0 ,MSB
      MOVFPF     BARGB0 ,INDF0
      MOVFPF     BARGB1 ,INDF0
      MOVFPF     BARGB2 ,INDF0
      MOVFPF     BARGB3 ,INDF0
      CALL      RANDOM16
      MOVFPF     RANDHI ,WREG
      MOVFPF     WREG ,AARGB0
      MOVFPF     RANDLO ,WREG
      MOVFPF     WREG ,AARGB1
      CALL      RANDOM16
      MOVFPF     RANDHI ,WREG
      MOVFPF     WREG ,AARGB2
      MOVFPF     RANDLO ,WREG
      MOVFPF     WREG ,AARGB3

;      BCF      AARGB0 ,MSB
      MOVFPF     AARGB0 ,INDF0
      MOVFPF     AARGB1 ,INDF0
      MOVFPF     AARGB2 ,INDF0
      MOVFPF     AARGB3 ,INDF0
;      CALL      FXD3232S
;      CALL      FXD3232U
;      CALL      FXD3231U
;      CALL      FXD3131U
      MOVFPF     AARGB0 ,INDF0
      MOVFPF     AARGB1 ,INDF0
      MOVFPF     AARGB2 ,INDF0
      MOVFPF     AARGB3 ,INDF0
      MOVFPF     REMB0 ,INDF0
      MOVFPF     REMB1 ,INDF0
      MOVFPF     REMB2 ,INDF0
      MOVFPF     REMB3 ,INDF0
      DECFSZ     TESTCOUNT
      GOTO      D3232LOOP
      RETLW     0x00

```

AN617

```
;      Test suite for FXD3224
TFXD3224      MOVLW      14
              MOVFPF    WREG,TESTCOUNT
              MOVFPF    WREG,NUMTESTS
              MOVLW     5
              MOVFPF    WREG,TESTCODE

D3224LOOP
              CALL      RANDOM16
              MOVFPF    RANDHI,WREG
              MOVFPF    WREG,BARGB0
              MOVFPF    RANDLO,WREG
              MOVFPF    WREG,BARGB1
              CALL      RANDOM16
              MOVFPF    RANDHI,WREG
              MOVFPF    WREG,BARGB2
;
              BCF      BARGB0,MSB
              MOVFPF    BARGB0,INDF0
              MOVFPF    BARGB1,INDF0
              MOVFPF    BARGB2,INDF0
              CALL      RANDOM16
              MOVFPF    RANDHI,WREG
              MOVFPF    WREG,AARGB0
              MOVFPF    RANDLO,WREG
              MOVFPF    WREG,AARGB1
              CALL      RANDOM16
              MOVFPF    RANDHI,WREG
              MOVFPF    WREG,AARGB2
              MOVFPF    RANDLO,WREG
              MOVFPF    WREG,AARGB3

;
              BCF      AARGB0,MSB
              MOVFPF    AARGB0,INDF0
              MOVFPF    AARGB1,INDF0
              MOVFPF    AARGB2,INDF0
              MOVFPF    AARGB3,INDF0
              CALL      FXD3224S
;
;
;
              CALL      FXD3224U
;
;
              CALL      FXD3123U
              MOVFPF    AARGB0,INDF0
              MOVFPF    AARGB1,INDF0
              MOVFPF    AARGB2,INDF0
              MOVFPF    AARGB3,INDF0
              MOVFPF    REMB0,INDF0
              MOVFPF    REMB1,INDF0
              MOVFPF    REMB2,INDF0
              DECFSZ    TESTCOUNT
              GOTO     D3224LOOP
              RETLW    0x00
;*****
;*****
;      32/32 Bit Division Macros
SDIV3232      macro
;      Max Timing:      9+14+30*18+10 = 573 clks
;      Min Timing:      9+14+30*17+3 = 536 clks
;      PM: 9+14+30*24+10 = 753          DM: 12
              variable i
              MOVFPF    BARGB3,WREG
              SUBWF     REMB3
              MOVFPF    BARGB2,WREG
              SUBWFB    REMB2
              MOVFPF    BARGB1,WREG
              SUBWFB    REMB1
              MOVFPF    BARGB0,WREG
              SUBWFB    REMB0
              RLCF      ACCB0
```

```

RLCF          ACCB0 , W
RLCF          REMB3
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB3 , WREG
ADDWF        REMB3
MOVFP        BARGB2 , WREG
ADDWFC       REMB2
MOVFP        BARGB1 , WREG
ADDWFC       REMB1
MOVFP        BARGB0 , WREG
ADDWFC       REMB0
RLCF          ACCB0
i = 2
while i < 8
RLCF          ACCB0 , W
RLCF          REMB3
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB3 , WREG
BTFSS       ACCB0 , LSB
GOTO        SADD22#v(i)
SUBWF       REMB3
MOVFP        BARGB2 , WREG
SUBWFB      REMB2
MOVFP        BARGB1 , WREG
SUBWFB      REMB1
MOVFP        BARGB0 , WREG
SUBWFB      REMB0
GOTO        SOK22#v(i)
SADD22#v(i)  ADDWF       REMB3
MOVFP        BARGB2 , WREG
ADDWFC       REMB2
MOVFP        BARGB1 , WREG
ADDWFC       REMB1
MOVFP        BARGB0 , WREG
ADDWFC       REMB0
SOK22#v(i)  RLCF          ACCB0
i=i+1
endw
RLCF          ACCB1 , W
RLCF          REMB3
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB3 , WREG
BTFSS       ACCB0 , LSB
GOTO        SADD228
SUBWF       REMB3
MOVFP        BARGB2 , WREG
SUBWFB      REMB2
MOVFP        BARGB1 , WREG
SUBWFB      REMB1
MOVFP        BARGB0 , WREG
SUBWFB      REMB0
GOTO        SOK228
SADD228     ADDWF       REMB3
MOVFP        BARGB2 , WREG
ADDWFC       REMB2
MOVFP        BARGB1 , WREG
ADDWFC       REMB1
MOVFP        BARGB0 , WREG
ADDWFC       REMB0
SOK228     RLCF          ACCB1

```

AN617

```

        i = 9
        while i < 16
            RLCF          ACCB1,W
            RLCF          REMB3
            RLCF          REMB2
            RLCF          REMB1
            RLCF          REMB0
            MOVFP        BARGB3,WREG
            BTFSS        ACCB1,LSB
            GOTO         SADD22#v(i)
            SUBWF        REMB3
            MOVFP        BARGB2,WREG
            SUBWFB       REMB2
            MOVFP        BARGB1,WREG
            SUBWFB       REMB1
            MOVFP        BARGB0,WREG
            SUBWFB       REMB0
            GOTO         SOK22#v(i)
SADD22#v(i)  ADDWF        REMB3
            MOVFP        BARGB2,WREG
            ADDWFC       REMB2
            MOVFP        BARGB1,WREG
            ADDWFC       REMB1
            MOVFP        BARGB0,WREG
            ADDWFC       REMB0
SOK22#v(i)  RLCF          ACCB1
            i=i+1
            endw
            RLCF          ACCB2,W
            RLCF          REMB3
            RLCF          REMB2
            RLCF          REMB1
            RLCF          REMB0
            MOVFP        BARGB3,WREG
            BTFSS        ACCB1,LSB
            GOTO         SADD2216
            SUBWF        REMB3
            MOVFP        BARGB2,WREG
            SUBWFB       REMB2
            MOVFP        BARGB1,WREG
            SUBWFB       REMB1
            MOVFP        BARGB0,WREG
            SUBWFB       REMB0
SADD2216    GOTO         SOK2216
            ADDWF        REMB3
            MOVFP        BARGB2,WREG
            ADDWFC       REMB2
            MOVFP        BARGB1,WREG
            ADDWFC       REMB1
            MOVFP        BARGB0,WREG
            ADDWFC       REMB0
SOK2216    RLCF          ACCB2
            i = 17
            while i < 24
                RLCF          ACCB2,W
                RLCF          REMB3
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP        BARGB3,WREG
                BTFSS        ACCB2,LSB
                GOTO         SADD22#v(i)
                SUBWF        REMB3
                MOVFP        BARGB2,WREG
                SUBWFB       REMB2
                MOVFP        BARGB1,WREG
```



```

SUBWFB          REMB1
MOVFP          BARGB0, WREG
SUBWFB          REMB0
GOTO          SOK22#v(i)
SADD22#v(i)    ADDWF          REMB3
MOVFP          BARGB2, WREG
ADDWFC        REMB2
MOVFP          BARGB1, WREG
ADDWFC        REMB1
MOVFP          BARGB0, WREG
ADDWFC        REMB0
SOK22#v(i)    RLCF          ACCB2
              i=i+1
              endw
              RLCF          ACCB3, W
              RLCF          REMB3
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
MOVFP          BARGB3, WREG
BTFSS        ACCB2, LSB
GOTO          SADD2224
SUBWF        REMB3
MOVFP          BARGB2, WREG
SUBWFB        REMB2
MOVFP          BARGB1, WREG
SUBWFB        REMB1
MOVFP          BARGB0, WREG
SUBWFB        REMB0
GOTO          SOK2224
SADD2224      ADDWF          REMB3
MOVFP          BARGB2, WREG
ADDWFC        REMB2
MOVFP          BARGB1, WREG
ADDWFC        REMB1
MOVFP          BARGB0, WREG
ADDWFC        REMB0
SOK2224      RLCF          ACCB3
              i = 25
              while i < 32
              RLCF          ACCB3, W
              RLCF          REMB3
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
MOVFP          BARGB3, WREG
BTFSS        ACCB3, LSB
GOTO          SADD22#v(i)
SUBWF        REMB3
MOVFP          BARGB2, WREG
SUBWFB        REMB2
MOVFP          BARGB1, WREG
SUBWFB        REMB1
MOVFP          BARGB0, WREG
SUBWFB        REMB0
GOTO          SOK22#v(i)
SADD22#v(i)    ADDWF          REMB3
MOVFP          BARGB2, WREG
ADDWFC        REMB2
MOVFP          BARGB1, WREG
ADDWFC        REMB1
MOVFP          BARGB0, WREG
ADDWFC        REMB0
SOK22#v(i)    RLCF          ACCB3
              i=i+1
              endw

```

AN617

```

    BTFSC          ACCB3,LSB
    GOTO          SOK22
    MOVFP         BARGB3,WREG
    ADDWF        REMB3
    MOVFP         BARGB2,WREG
    ADDWFC       REMB2
    MOVFP         BARGB1,WREG
    ADDWFC       REMB1
    MOVFP         BARGB0,WREG
    ADDWFC       REMB0
SOK22
    endm
UDIV3232
macro
;   restore = 25/30 clks, nonrestore = 17/20 clks
;   Max Timing: 16*25+1+16*30 = 881 clks
;   Min Timing: 16*17+1+16*20 = 593 clks
;   PM: 16*25+1+16*30 = 881          DM: 13
    variable      i
    i = 0
    while i < 8
        RLCF          ACCB0,W
        RLCF          REMB3
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        MOVFP         BARGB3,WREG
        SUBWF         REMB3
        MOVFP         BARGB2,WREG
        SUBWFB        REMB2
        MOVFP         BARGB1,WREG
        SUBWFB        REMB1
        MOVFP         BARGB0,WREG
        SUBWFB        REMB0
        BTFSC         _C
        GOTO          UOK22#v(i)
        MOVFP         BARGB3,WREG
        ADDWF        REMB3
        MOVFP         BARGB2,WREG
        ADDWFC       REMB2
        MOVFP         BARGB1,WREG
        ADDWFC       REMB1
        MOVFP         BARGB0,WREG
        ADDWFC       REMB0
        BCF          _C
UOK22#v(i)
        RLCF          ACCB0
        i=i+1
    endw
    i = 8
    while i < 16
        RLCF          ACCB1,W
        RLCF          REMB3
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        MOVFP         BARGB3,WREG
        SUBWF         REMB3
        MOVFP         BARGB2,WREG
        SUBWFB        REMB2
        MOVFP         BARGB1,WREG
        SUBWFB        REMB1
        MOVFP         BARGB0,WREG
        SUBWFB        REMB0
        BTFSC         _C
        GOTO          UOK22#v(i)
        MOVFP         BARGB3,WREG
        ADDWF        REMB3

```

```

MOVFP      BARGB2,WREG
ADDWFC     REMB2
MOVFP      BARGB1,WREG
ADDWFC     REMB1
MOVFP      BARGB0,WREG
ADDWFC     REMB0
BCF        _C
UOK22#v(i) RLCF        ACCB1
           i=i+1
           endw
           CLRF        TEMP
           i = 16
           while i < 24
           RLCF        ACCB2,W
           RLCF        REMB3
           RLCF        REMB2
           RLCF        REMB1
           RLCF        REMB0
           RLCF        TEMP
           MOVFP      BARGB3,WREG
           SUBWF      REMB3
           MOVFP      BARGB2,WREG
           SUBWFB     REMB2
           MOVFP      BARGB1,WREG
           SUBWFB     REMB1
           MOVFP      BARGB0,WREG
           SUBWFB     REMB0
           CLRF      WREG
           SUBWFB     TEMP
           BTFSC     _C
           GOTO      UOK22#v(i)
           MOVFP      BARGB3,WREG
           ADDWF      REMB3
           MOVFP      BARGB2,WREG
           ADDWFC     REMB2
           MOVFP      BARGB1,WREG
           ADDWFC     REMB1
           MOVFP      BARGB0,WREG
           ADDWFC     REMB0
           CLRF      WREG
           ADDWFC     TEMP
           BCF        _C
UOK22#v(i) RLCF        ACCB2
           i=i+1
           endw
           i = 24
           while i < 32
           RLCF        ACCB3,W
           RLCF        REMB3
           RLCF        REMB2
           RLCF        REMB1
           RLCF        REMB0
           RLCF        TEMP
           MOVFP      BARGB3,WREG
           SUBWF      REMB3
           MOVFP      BARGB2,WREG
           SUBWFB     REMB2
           MOVFP      BARGB1,WREG
           SUBWFB     REMB1
           MOVFP      BARGB0,WREG
           SUBWFB     REMB0
           CLRF      WREG
           SUBWFB     TEMP
           BTFSC     _C
           GOTO      UOK22#v(i)
           MOVFP      BARGB3,WREG

```

AN617

```

                ADDWF          REMB3
                MOVFP         BARGB2,WREG
                ADDWFC        REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
                CLRF          WREG
                ADDWFC        TEMP
                BCF           _C
UOK22#v(i)     RLCF          ACCB3
                i=i+1
                endw
                endm
NDIV3232      macro
;             Max Timing:      16+31*21+10 = 677 clks
;             Min Timing: 16+31*20+3 = 639 clks
;             PM: 16+31*29+10 = 925          DM: 13
                variable i
                RLCF          ACCB0,W
                RLCF          REMB3
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB3,WREG
                SUBWF         REMB3
                MOVFP         BARGB2,WREG
                SUBWFB        REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                CLRF          TEMP,W
                SUBWFB        TEMP
                RLCF          ACCB0
                i = 1
                while i < 8
                RLCF          ACCB0,W
                RLCF          REMB3
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                RLCF          TEMP
                MOVFP         BARGB3,WREG
                BTFSS         ACCB0,LSB
                GOTO         NADD22#v(i)
                SUBWF         REMB3
                MOVFP         BARGB2,WREG
                SUBWFB        REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                CLRF          WREG
                SUBWFB        TEMP
                GOTO         NOK22#v(i)
NADD22#v(i)   ADDWF          REMB3
                MOVFP         BARGB2,WREG
                ADDWFC        REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
                CLRF          WREG
                ADDWFC        TEMP
```

```

NOK22#v(i)    RLCF          ACCB0
              i=i+1
              endw
              RLCF          ACCB1,W
              RLCF          REMB3
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
              RLCF          TEMP
              MOVFP         BARGB3,WREG
              BTFSS         ACCB0,LSB
              GOTO          NADD228
              SUBWF         REMB3
              MOVFP         BARGB2,WREG
              SUBWFB        REMB2
              MOVFP         BARGB1,WREG
              SUBWFB        REMB1
              MOVFP         BARGB0,WREG
              SUBWFB        REMB0
              CLRF          WREG
              SUBWFB        TEMP
              GOTO          NOK228
NADD228      ADDWF         REMB3
              MOVFP         BARGB2,WREG
              ADDWFC        REMB2
              MOVFP         BARGB1,WREG
              ADDWFC        REMB1
              MOVFP         BARGB0,WREG
              ADDWFC        REMB0
              CLRF          WREG
              ADDWFC        TEMP
NOK228      RLCF          ACCB1
              i = 9
              while i < 16
              RLCF          ACCB1,W
              RLCF          REMB3
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
              RLCF          TEMP
              MOVFP         BARGB3,WREG
              BTFSS         ACCB1,LSB
              GOTO          NADD22#v(i)
              SUBWF         REMB3
              MOVFP         BARGB2,WREG
              SUBWFB        REMB2
              MOVFP         BARGB1,WREG
              SUBWFB        REMB1
              MOVFP         BARGB0,WREG
              SUBWFB        REMB0
              CLRF          WREG
              SUBWFB        TEMP
              GOTO          NOK22#v(i)
NADD22#v(i)  ADDWF         REMB3
              MOVFP         BARGB2,WREG
              ADDWFC        REMB2
              MOVFP         BARGB1,WREG
              ADDWFC        REMB1
              MOVFP         BARGB0,WREG
              ADDWFC        REMB0
              CLRF          WREG
              ADDWFC        TEMP
NOK22#v(i)  RLCF          ACCB1
              i=i+1

```

AN617

```
        endw
        RLCF          ACCB2,W
        RLCF          REMB3
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        RLCF          TEMP
        MOVFP         BARGB3,WREG
        BTFSS         ACCB1,LSB
        GOTO          NADD2216
        SUBWF         REMB3
        MOVFP         BARGB2,WREG
        SUBWFB        REMB2
        MOVFP         BARGB1,WREG
        SUBWFB        REMB1
        MOVFP         BARGB0,WREG
        SUBWFB        REMB0
        CLRF          WREG
        SUBWFB        TEMP
NADD2216  GOTO          NOK2216
        ADDWF         REMB3
        MOVFP         BARGB2,WREG
        ADDWFC        REMB2
        MOVFP         BARGB1,WREG
        ADDWFC        REMB1
        MOVFP         BARGB0,WREG
        ADDWFC        REMB0
        CLRF          WREG
        ADDWFC        TEMP

NOK2216  RLCF          ACCB2
        i = 17
        while i < 24
        RLCF          ACCB2,W
        RLCF          REMB3
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        RLCF          TEMP
        MOVFP         BARGB3,WREG
        BTFSS         ACCB2,LSB
        GOTO          NADD22#v(i)
        SUBWF         REMB3
        MOVFP         BARGB2,WREG
        SUBWFB        REMB2
        MOVFP         BARGB1,WREG
        SUBWFB        REMB1
        MOVFP         BARGB0,WREG
        SUBWFB        REMB0
        CLRF          WREG
        SUBWFB        TEMP
NADD22#v(i)  GOTO          NOK22#v(i)
        ADDWF         REMB3
        MOVFP         BARGB2,WREG
        ADDWFC        REMB2
        MOVFP         BARGB1,WREG
        ADDWFC        REMB1
        MOVFP         BARGB0,WREG
        ADDWFC        REMB0
        CLRF          WREG
        ADDWFC        TEMP

NOK22#v(i)  RLCF          ACCB2
        i=i+1
        endw
        RLCF          ACCB3,W
```

	RLCF	REMB3
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	RLCF	TEMP
	MOVFP	BARGB3, WREG
	BTFSS	ACCB2, LSB
	GOTO	NADD2224
	SUBWF	REMB3
	MOVFP	BARGB2, WREG
	SUBWFB	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	CLRF	WREG
	SUBWFB	TEMP
	GOTO	NOK2224
NADD2224	ADDWF	REMB3
	MOVFP	BARGB2, WREG
	ADDWFC	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
	CLRF	WREG
	ADDWFC	TEMP
NOK2224	RLCF	ACCB3
	i = 25	
	while i < 32	
	RLCF	ACCB3, W
	RLCF	REMB3
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	RLCF	TEMP
	MOVFP	BARGB3, WREG
	BTFSS	ACCB3, LSB
	GOTO	NADD22#v(i)
	SUBWF	REMB3
	MOVFP	BARGB2, WREG
	SUBWFB	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	CLRF	WREG
	SUBWFB	TEMP
	GOTO	NOK22#v(i)
NADD22#v(i)	ADDWF	REMB3
	MOVFP	BARGB2, WREG
	ADDWFC	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
	CLRF	WREG
	ADDWFC	TEMP
NOK22#v(i)	RLCF	ACCB3
	i=i+1	
	endw	
	BTFSC	ACCB3, LSB
	GOTO	NOK22
	MOVFP	BARGB3, WREG

AN617

```

        ADDWF          REMB3
        MOVFP         BARGB2,WREG
        ADDWFC       REMB2
        MOVFP         BARGB1,WREG
        ADDWFC       REMB1
        MOVFP         BARGB0,WREG
        ADDWFC       REMB0

NOK22
        endm
UDIV3231
        macro
;      Max Timing:    14+31*18+10 = 582 clks
;      Min Timing:    14+31*17+3 = 544 clks
;      PM: 14+31*24+10 = 768
;
        variable i
        RLCF          ACCB0,W
        RLCF          REMB3
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        MOVFP         BARGB3,WREG
        SUBWF         REMB3
        MOVFP         BARGB2,WREG
        SUBWFB        REMB2
        MOVFP         BARGB1,WREG
        SUBWFB        REMB1
        MOVFP         BARGB0,WREG
        SUBWFB        REMB0
        RLCF          ACCB0
        i = 1
        while i < 8
        RLCF          ACCB0,W
        RLCF          REMB3
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        MOVFP         BARGB3,WREG
        BTFSS         ACCB0,LSB
        GOTO          UADD21#v(i)
        SUBWF         REMB3
        MOVFP         BARGB2,WREG
        SUBWFB        REMB2
        MOVFP         BARGB1,WREG
        SUBWFB        REMB1
        MOVFP         BARGB0,WREG
        SUBWFB        REMB0
        GOTO          UOK21#v(i)
UADD21#v(i)
        ADDWF          REMB3
        MOVFP         BARGB2,WREG
        ADDWFC       REMB2
        MOVFP         BARGB1,WREG
        ADDWFC       REMB1
        MOVFP         BARGB0,WREG
        ADDWFC       REMB0
UOK21#v(i)
        RLCF          ACCB0
        i=i+1
        endw
        RLCF          ACCB1,W
        RLCF          REMB3
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        MOVFP         BARGB3,WREG
        BTFSS         ACCB0,LSB
        GOTO          UADD218
        SUBWF         REMB3
        MOVFP         BARGB2,WREG
```


	SUBWFB	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	UOK218
UADD218	ADDWF	REMB3
	MOVFP	BARGB2, WREG
	ADDWFC	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
UOK218	RLCF	ACCB1
	i = 9	
	while i < 16	
	RLCF	ACCB1, W
	RLCF	REMB3
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB3, WREG
	BTFSS	ACCB1, LSB
	GOTO	UADD21#v(i)
	SUBWF	REMB3
	MOVFP	BARGB2, WREG
	SUBWFB	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	UOK21#v(i)
UADD21#v(i)	ADDWF	REMB3
	MOVFP	BARGB2, WREG
	ADDWFC	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
UOK21#v(i)	RLCF	ACCB1
	i=i+1	
	endw	
	RLCF	ACCB2, W
	RLCF	REMB3
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB3, WREG
	BTFSS	ACCB1, LSB
	GOTO	UADD2116
	SUBWF	REMB3
	MOVFP	BARGB2, WREG
	SUBWFB	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	UOK2116
UADD2116	ADDWF	REMB3
	MOVFP	BARGB2, WREG
	ADDWFC	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
UOK2116	RLCF	ACCB2

AN617

```

        i = 17
        while i < 24
            RLCF          ACCB2,W
            RLCF          REMB3
            RLCF          REMB2
            RLCF          REMB1
            RLCF          REMB0
            MOVFP        BARGB3,WREG
            BTFSS        ACCB2,LSB
            GOTO         UADD21#v(i)
            SUBWF        REMB3
            MOVFP        BARGB2,WREG
            SUBWFB       REMB2
            MOVFP        BARGB1,WREG
            SUBWFB       REMB1
            MOVFP        BARGB0,WREG
            SUBWFB       REMB0
            GOTO         UOK21#v(i)
UADD21#v(i)  ADDWF        REMB3
            MOVFP        BARGB2,WREG
            ADDWFC       REMB2
            MOVFP        BARGB1,WREG
            ADDWFC       REMB1
            MOVFP        BARGB0,WREG
            ADDWFC       REMB0
UOK21#v(i)  RLCF          ACCB2
            i=i+1
            endw
            RLCF          ACCB3,W
            RLCF          REMB3
            RLCF          REMB2
            RLCF          REMB1
            RLCF          REMB0
            MOVFP        BARGB3,WREG
            BTFSS        ACCB2,LSB
            GOTO         UADD2124
            SUBWF        REMB3
            MOVFP        BARGB2,WREG
            SUBWFB       REMB2
            MOVFP        BARGB1,WREG
            SUBWFB       REMB1
            MOVFP        BARGB0,WREG
            SUBWFB       REMB0
            GOTO         UOK2124
UADD2124   ADDWF        REMB3
            MOVFP        BARGB2,WREG
            ADDWFC       REMB2
            MOVFP        BARGB1,WREG
            ADDWFC       REMB1
            MOVFP        BARGB0,WREG
            ADDWFC       REMB0
UOK2124   RLCF          ACCB3
            i = 25
            while i < 32
                RLCF          ACCB3,W
                RLCF          REMB3
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP        BARGB3,WREG
                BTFSS        ACCB3,LSB
                GOTO         UADD21#v(i)
                SUBWF        REMB3
                MOVFP        BARGB2,WREG
                SUBWFB       REMB2
                MOVFP        BARGB1,WREG
```

```

                SUBWFB          REMB1
                MOVFP          BARGB0, WREG
                SUBWFB          REMB0
                GOTO           UOK21#v(i)
UADD21#v(i)    ADDWF          REMB3
                MOVFP          BARGB2, WREG
                ADDWFC         REMB2
                MOVFP          BARGB1, WREG
                ADDWFC         REMB1
                MOVFP          BARGB0, WREG
                ADDWFC         REMB0
UOK21#v(i)    RLCF           ACCB3
                i=i+1
                endw
                BTFSC         ACCB3, LSB
                GOTO           UOK21
                MOVFP          BARGB3, WREG
                ADDWF          REMB3
                MOVFP          BARGB2, WREG
                ADDWFC         REMB2
                MOVFP          BARGB1, WREG
                ADDWFC         REMB1
                MOVFP          BARGB0, WREG
                ADDWFC         REMB0
UOK21
                endm
UDIV3131      macro
;           Max Timing:      9+14+30*18+10 = 573 clks
;           Min Timing:      9+14+30*17+3 = 536 clks
;           PM: 9+14+30*24+10 = 753           DM: 12
                variable i
                MOVFP          BARGB3, WREG
                SUBWF          REMB3
                MOVFP          BARGB2, WREG
                SUBWFB         REMB2
                MOVFP          BARGB1, WREG
                SUBWFB         REMB1
                MOVFP          BARGB0, WREG
                SUBWFB         REMB0
                RLCF          ACCB0
                RLCF          ACCB0, W
                RLCF          REMB3
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB3, WREG
                ADDWF          REMB3
                MOVFP          BARGB2, WREG
                ADDWFC         REMB2
                MOVFP          BARGB1, WREG
                ADDWFC         REMB1
                MOVFP          BARGB0, WREG
                ADDWFC         REMB0
                RLCF          ACCB0
                i = 2
                while i < 8
                RLCF          ACCB0, W
                RLCF          REMB3
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB3, WREG
                BTFSS         ACCB0, LSB
                GOTO           UADD11#v(i)
                SUBWF          REMB3
                MOVFP          BARGB2, WREG

```

AN617

```

SUBWFB          REMB2
MOVFP          BARGB1,WREG
SUBWFB          REMB1
MOVFP          BARGB0,WREG
SUBWFB          REMB0
GOTO           UOK11#v(i)
UADD11#v(i)    ADDWF          REMB3
MOVFP          BARGB2,WREG
ADDWFC         REMB2
MOVFP          BARGB1,WREG
ADDWFC         REMB1
MOVFP          BARGB0,WREG
ADDWFC         REMB0
UOK11#v(i)    RLCF          ACCB0
              i=i+1
              endw
              RLCF          ACCB1,W
              RLCF          REMB3
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
              MOVFP         BARGB3,WREG
              BTFSS         ACCB0,LSB
              GOTO           UADD118
              SUBWF         REMB3
              MOVFP         BARGB2,WREG
              SUBWFB        REMB2
              MOVFP         BARGB1,WREG
              SUBWFB        REMB1
              MOVFP         BARGB0,WREG
              SUBWFB        REMB0
              GOTO           UOK118
UADD118        ADDWF          REMB3
MOVFP          BARGB2,WREG
ADDWFC         REMB2
MOVFP          BARGB1,WREG
ADDWFC         REMB1
MOVFP          BARGB0,WREG
ADDWFC         REMB0
UOK118        RLCF          ACCB1
              i = 9
              while i < 16
              RLCF          ACCB1,W
              RLCF          REMB3
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
              MOVFP         BARGB3,WREG
              BTFSS         ACCB1,LSB
              GOTO           UADD11#v(i)
              SUBWF         REMB3
              MOVFP         BARGB2,WREG
              SUBWFB        REMB2
              MOVFP         BARGB1,WREG
              SUBWFB        REMB1
              MOVFP         BARGB0,WREG
              SUBWFB        REMB0
              GOTO           UOK11#v(i)
UADD11#v(i)    ADDWF          REMB3
MOVFP          BARGB2,WREG
ADDWFC         REMB2
MOVFP          BARGB1,WREG
ADDWFC         REMB1
MOVFP          BARGB0,WREG
ADDWFC         REMB0
UOK11#v(i)    RLCF          ACCB1
```

```

        i=i+1
    endw
    RLCF          ACCB2,W
    RLCF          REMB3
    RLCF          REMB2
    RLCF          REMB1
    RLCF          REMB0
    MOVFP        BARGB3,WREG
    BTFSS        ACCB1,LSB
    GOTO         UADD1116
    SUBWF        REMB3
    MOVFP        BARGB2,WREG
    SUBWFB       REMB2
    MOVFP        BARGB1,WREG
    SUBWFB       REMB1
    MOVFP        BARGB0,WREG
    SUBWFB       REMB0
    GOTO         UOK1116
UADD1116      ADDWF          REMB3
    MOVFP        BARGB2,WREG
    ADDWFC       REMB2
    MOVFP        BARGB1,WREG
    ADDWFC       REMB1
    MOVFP        BARGB0,WREG
    ADDWFC       REMB0
UOK1116      RLCF          ACCB2
    i = 17
    while i < 24
    RLCF          ACCB2,W
    RLCF          REMB3
    RLCF          REMB2
    RLCF          REMB1
    RLCF          REMB0
    MOVFP        BARGB3,WREG
    BTFSS        ACCB2,LSB
    GOTO         UADD11#v(i)
    SUBWF        REMB3
    MOVFP        BARGB2,WREG
    SUBWFB       REMB2
    MOVFP        BARGB1,WREG
    SUBWFB       REMB1
    MOVFP        BARGB0,WREG
    SUBWFB       REMB0
    GOTO         UOK11#v(i)
UADD11#v(i)  ADDWF          REMB3
    MOVFP        BARGB2,WREG
    ADDWFC       REMB2
    MOVFP        BARGB1,WREG
    ADDWFC       REMB1
    MOVFP        BARGB0,WREG
    ADDWFC       REMB0
UOK11#v(i)   RLCF          ACCB2
    i=i+1
    endw
    RLCF          ACCB3,W
    RLCF          REMB3
    RLCF          REMB2
    RLCF          REMB1
    RLCF          REMB0
    MOVFP        BARGB3,WREG
    BTFSS        ACCB2,LSB
    GOTO         UADD1124
    SUBWF        REMB3
    MOVFP        BARGB2,WREG
    SUBWFB       REMB2
    MOVFP        BARGB1,WREG

```

AN617

```

                SUBWFB      REMB1
                MOVFP      BARGB0,WREG
                SUBWFB      REMB0
                GOTO       UOK1124
UADD1124      ADDWF      REMB3
                MOVFP      BARGB2,WREG
                ADDWFC      REMB2
                MOVFP      BARGB1,WREG
                ADDWFC      REMB1
                MOVFP      BARGB0,WREG
                ADDWFC      REMB0
UOK1124      RLCF      ACCB3
                i = 25
                while i < 32
                RLCF      ACCB3,W
                RLCF      REMB3
                RLCF      REMB2
                RLCF      REMB1
                RLCF      REMB0
                MOVFP      BARGB3,WREG
                BTFSS      ACCB3,LSB
                GOTO       UADD11#v(i)
                SUBWF      REMB3
                MOVFP      BARGB2,WREG
                SUBWFB      REMB2
                MOVFP      BARGB1,WREG
                SUBWFB      REMB1
                MOVFP      BARGB0,WREG
                SUBWFB      REMB0
                GOTO       UOK11#v(i)
UADD11#v(i)  ADDWF      REMB3
                MOVFP      BARGB2,WREG
                ADDWFC      REMB2
                MOVFP      BARGB1,WREG
                ADDWFC      REMB1
                MOVFP      BARGB0,WREG
                ADDWFC      REMB0
UOK11#v(i)  RLCF      ACCB3
                i=i+1
                endw
                BTFSC      ACCB3,LSB
                GOTO       UOK11
                MOVFP      BARGB3,WREG
                ADDWF      REMB3
                MOVFP      BARGB2,WREG
                ADDWFC      REMB2
                MOVFP      BARGB1,WREG
                ADDWFC      REMB1
                MOVFP      BARGB0,WREG
                ADDWFC      REMB0
UOK11
                endm
;*****
;*****
;      32/24 Bit Division Macros
SDIV3224      macro
;      Max Timing:      7+11+30*15+8 = 476 clks
;      Min Timing:      7+11+30*14+3 = 441 clks
;      PM: 7+11+30*19+8 = 596          DM: 10
                variable i
                MOVFP      BARGB2,WREG
                SUBWF      REMB2
                MOVFP      BARGB1,WREG
                SUBWFB      REMB1
                MOVFP      BARGB0,WREG
                SUBWFB      REMB0
```

```

RLCF          ACCB0
RLCF          ACCB0,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB2,WREG
ADDWF        REMB2
MOVFP        BARGB1,WREG
ADDWFC       REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
RLCF          ACCB0
i = 2
while i < 8
RLCF          ACCB0,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB2,WREG
BTFSS       ACCB0,LSB
GOTO        SADD24#v(i)
SUBWF       REMB2
MOVFP        BARGB1,WREG
SUBWFB      REMB1
MOVFP        BARGB0,WREG
SUBWFB      REMB0
GOTO        SOK24#v(i)
SADD24#v(i)  ADDWF        REMB2
MOVFP        BARGB1,WREG
ADDWFC       REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
SOK24#v(i)  RLCF          ACCB0
i=i+1
endw
RLCF          ACCB1,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB2,WREG
BTFSS       ACCB0,LSB
GOTO        SADD248
SUBWF       REMB2
MOVFP        BARGB1,WREG
SUBWFB      REMB1
MOVFP        BARGB0,WREG
SUBWFB      REMB0
GOTO        SOK248
SADD248     ADDWF        REMB2
MOVFP        BARGB1,WREG
ADDWFC       REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
SOK248     RLCF          ACCB1
i = 9
while i < 16
RLCF          ACCB1,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB2,WREG
BTFSS       ACCB1,LSB
GOTO        SADD24#v(i)
SUBWF       REMB2
MOVFP        BARGB1,WREG
SUBWFB      REMB1

```

AN617

```
MOVFP          BARGB0,WREG
SUBWFB         REMB0
SOK24#v(i)    GOTO      SOK24#v(i)
SADD24#v(i)   ADDWF     REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
SOK24#v(i)    RLCF     ACCB1
               i=i+1
               endw
RLCF          ACCB2,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP          BARGB2,WREG
BTFSS         ACCB1,LSB
GOTO          SADD2416
SUBWF         REMB2
MOVFP          BARGB1,WREG
SUBWFB        REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO          SOK2416
SADD2416     ADDWF     REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
SOK2416     RLCF     ACCB2
               i = 17
               while i < 24
RLCF          ACCB2,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP          BARGB2,WREG
BTFSS         ACCB2,LSB
GOTO          SADD24#v(i)
SUBWF         REMB2
MOVFP          BARGB1,WREG
SUBWFB        REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO          SOK24#v(i)
SADD24#v(i)   ADDWF     REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
SOK24#v(i)    RLCF     ACCB2
               i=i+1
               endw
RLCF          ACCB3,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP          BARGB2,WREG
BTFSS         ACCB2,LSB
GOTO          SADD2424
SUBWF         REMB2
MOVFP          BARGB1,WREG
SUBWFB        REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO          SOK2424
```



```

SADD2424      ADDWF      REMB2
               MOVFP      BARGB1,WREG
               ADDWFC     REMB1
               MOVFP      BARGB0,WREG
               ADDWFC     REMB0
SOK2424      RLCF      ACCB3
               i = 25
               while i < 32
               RLCF      ACCB3,W
               RLCF      REMB2
               RLCF      REMB1
               RLCF      REMB0
               MOVFP      BARGB2,WREG
               BTFSS     ACCB3,LSB
               GOTO      SADD24#v(i)
               SUBWF     REMB2
               MOVFP      BARGB1,WREG
               SUBWFB    REMB1
               MOVFP      BARGB0,WREG
               SUBWFB    REMB0
               GOTO      SOK24#v(i)
SADD24#v(i)  ADDWF      REMB2
               MOVFP      BARGB1,WREG
               ADDWFC     REMB1
               MOVFP      BARGB0,WREG
               ADDWFC     REMB0
SOK24#v(i)  RLCF      ACCB3
               i=i+1
               endw
               BTFSC     ACCB3,LSB
               GOTO      SOK24
               MOVFP      BARGB2,WREG
               ADDWF     REMB2
               MOVFP      BARGB1,WREG
               ADDWFC     REMB1
               MOVFP      BARGB0,WREG
               ADDWFC     REMB0

SOK24
               endm
UDIV3224      macro
;           restore = 20/25 clks, nonrestore = 14/17 clks
;           Max Timing: 16*20+1+16*25 = 721 clks
;           Min Timing: 16*14+1+16*17 = 497 clks
;           PM: 16*20+1+16*25 = 721          DM: 11
               variable      i
               i = 0
               while i < 8
               RLCF      ACCB0,W
               RLCF      REMB2
               RLCF      REMB1
               RLCF      REMB0
               MOVFP      BARGB2,WREG
               SUBWF     REMB2
               MOVFP      BARGB1,WREG
               SUBWFB    REMB1
               MOVFP      BARGB0,WREG
               SUBWFB    REMB0
               BTFSC     _C
               GOTO      UOK24#v(i)
               MOVFP      BARGB2,WREG
               ADDWF     REMB2
               MOVFP      BARGB1,WREG
               ADDWFC     REMB1
               MOVFP      BARGB0,WREG
               ADDWFC     REMB0
               BCF      _C

```

AN617

```
UOK24#v(i)    RLCF          ACCB0
              i=i+1
              endw
              i = 8
              while i < 16
                RLCF          ACCB1,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                BTFSC         _C
                GOTO          UOK24#v(i)
                MOVFP         BARGB2,WREG
                ADDWF         REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
                BCF           _C
UOK24#v(i)    RLCF          ACCB1
              i=i+1
              endw
              CLRF          TEMP
              i = 16
              while i < 24
                RLCF          ACCB2,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                RLCF          TEMP
                MOVFP         BARGB2,WREG
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                CLRF         WREG
                SUBWFB        TEMP
                BTFSC         _C
                GOTO          UOK24#v(i)
                MOVFP         BARGB2,WREG
                ADDWF         REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
                CLRF         WREG
                ADDWFC        TEMP
                BCF           _C
UOK24#v(i)    RLCF          ACCB2
              i=i+1
              endw
              i = 24
              while i < 32
                RLCF          ACCB3,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                RLCF          TEMP
                MOVFP         BARGB2,WREG
                SUBWF         REMB2
```

```

MOVFP      BARGB1,WREG
SUBWFB     REMB1
MOVFP      BARGB0,WREG
SUBWFB     REMB0
CLRF       WREG
SUBWFB     TEMP
BTFSC     _C
GOTO      UOK24#v(i)
MOVFP      BARGB2,WREG
ADDWF     REMB2
MOVFP      BARGB1,WREG
ADDWFC    REMB1
MOVFP      BARGB0,WREG
ADDWFC    REMB0
CLRF       WREG
ADDWFC    TEMP
BCF        _C
UOK24#v(i) RLCF      ACCB3
            i=i+1
            endw
            endm
NDIV3224   macro
;           Max Timing:      13+31*18+8 = 579 clks
;           Min Timing:    13+31*17+3 = 543 clks
;           PM: 13+31*24+8 = 765           DM: 11
            variable i
            RLCF      ACCB0,W
            RLCF      REMB2
            RLCF      REMB1
            RLCF      REMB0
            MOVFP     BARGB2,WREG
            SUBWF     REMB2
            MOVFP     BARGB1,WREG
            SUBWFB    REMB1
            MOVFP     BARGB0,WREG
            SUBWFB    REMB0
            CLRF      TEMP,W
            SUBWFB    TEMP
            RLCF      ACCB0
            i = 1
            while i < 8
            RLCF      ACCB0,W
            RLCF      REMB2
            RLCF      REMB1
            RLCF      REMB0
            RLCF      TEMP
            MOVFP     BARGB2,WREG
            BTFSS     ACCB0,LSB
            GOTO      NADD24#v(i)
            SUBWF     REMB2
            MOVFP     BARGB1,WREG
            SUBWFB    REMB1
            MOVFP     BARGB0,WREG
            SUBWFB    REMB0
            CLRF      WREG
            SUBWFB    TEMP
            GOTO      NOK24#v(i)
NADD24#v(i) ADDWF     REMB2
            MOVFP     BARGB1,WREG
            ADDWFC    REMB1
            MOVFP     BARGB0,WREG
            ADDWFC    REMB0
            CLRF      WREG
            ADDWFC    TEMP
NOK24#v(i) RLCF      ACCB0

```

AN617

```

        i=i+1
    endw
    RLCF          ACCB1,W
    RLCF          REMB2
    RLCF          REMB1
    RLCF          REMB0
    RLCF          TEMP
    MOVFP        BARGB2,WREG
    BTFSS        ACCB0,LSB
    GOTO         NADD248
    SUBWF        REMB2
    MOVFP        BARGB1,WREG
    SUBWFB       REMB1
    MOVFP        BARGB0,WREG
    SUBWFB       REMB0
    CLRF         WREG
    SUBWFB       TEMP
    GOTO         NOK248
NADD248    ADDWF          REMB2
    MOVFP        BARGB1,WREG
    ADDWFC       REMB1
    MOVFP        BARGB0,WREG
    ADDWFC       REMB0
    CLRF         WREG
    ADDWFC       TEMP

NOK248    RLCF          ACCB1
        i = 9
    while i < 16
        RLCF          ACCB1,W
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        RLCF          TEMP
        MOVFP        BARGB2,WREG
        BTFSS        ACCB1,LSB
        GOTO         NADD24#v(i)
        SUBWF        REMB2
        MOVFP        BARGB1,WREG
        SUBWFB       REMB1
        MOVFP        BARGB0,WREG
        SUBWFB       REMB0
        CLRF         WREG
        SUBWFB       TEMP
        GOTO         NOK24#v(i)
NADD24#v(i)    ADDWF          REMB2
    MOVFP        BARGB1,WREG
    ADDWFC       REMB1
    MOVFP        BARGB0,WREG
    ADDWFC       REMB0
    CLRF         WREG
    ADDWFC       TEMP

NOK24#v(i)    RLCF          ACCB1
        i=i+1
    endw
    RLCF          ACCB2,W
    RLCF          REMB2
    RLCF          REMB1
    RLCF          REMB0
    RLCF          TEMP
    MOVFP        BARGB2,WREG
    BTFSS        ACCB1,LSB
    GOTO         NADD2416
    SUBWF        REMB2
    MOVFP        BARGB1,WREG
```

	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	CLRF	WREG
	SUBWFB	TEMP
	GOTO	NOK2416
NADD2416	ADDWF	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
	CLRF	WREG
	ADDWFC	TEMP
NOK2416	RLCF	ACCB2
	i = 17	
	while i < 24	
	RLCF	ACCB2, W
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	RLCF	TEMP
	MOVFP	BARGB2, WREG
	BTFSS	ACCB2, LSB
	GOTO	NADD24#v(i)
	SUBWF	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	CLRF	WREG
	SUBWFB	TEMP
	GOTO	NOK24#v(i)
NADD24#v(i)	ADDWF	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
	CLRF	WREG
	ADDWFC	TEMP
NOK24#v(i)	RLCF	ACCB2
	i=i+1	
	endw	
	RLCF	ACCB3, W
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	RLCF	TEMP
	MOVFP	BARGB2, WREG
	BTFSS	ACCB2, LSB
	GOTO	NADD2424
	SUBWF	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	CLRF	WREG
	SUBWFB	TEMP
	GOTO	NOK2424
NADD2424	ADDWF	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
	CLRF	WREG

AN617

```

                ADDWFC          TEMP
NOK2424        RLCF            ACCB3
                i = 25
                while i < 32
                RLCF            ACCB3,W
                RLCF            REMB2
                RLCF            REMB1
                RLCF            REMB0
                RLCF            TEMP
                MOVFP           BARGB2,WREG
                BTFSS           ACCB3,LSB
                GOTO            NADD24#v(i)
                SUBWF           REMB2
                MOVFP           BARGB1,WREG
                SUBWFB          REMB1
                MOVFP           BARGB0,WREG
                SUBWFB          REMB0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK24#v(i)
NADD24#v(i)    ADDWF           REMB2
                MOVFP           BARGB1,WREG
                ADDWFC          REMB1
                MOVFP           BARGB0,WREG
                ADDWFC          REMB0
                CLRF            WREG
                ADDWFC          TEMP
NOK24#v(i)    RLCF            ACCB3
                i=i+1
                endw
                BTFSC           ACCB3,LSB
                GOTO            NOK24
                MOVFP           BARGB2,WREG
                ADDWF           REMB2
                MOVFP           BARGB1,WREG
                ADDWFC          REMB1
                MOVFP           BARGB0,WREG
                ADDWFC          REMB0
NOK24         endm
UDIV3223      macro
;           Max Timing:      11+31*15+8 = 484 clks
;           Min Timing:      11+31*14+3 = 448 clks
;           PM: 11+31*19+8 = 608
;
                variable i
                RLCF            ACCB0,W
                RLCF            REMB2
                RLCF            REMB1
                RLCF            REMB0
                MOVFP           BARGB2,WREG
                SUBWF           REMB2
                MOVFP           BARGB1,WREG
                SUBWFB          REMB1
                MOVFP           BARGB0,WREG
                SUBWFB          REMB0
                RLCF            ACCB0
                i = 1
                while i < 8
                RLCF            ACCB0,W
                RLCF            REMB2
                RLCF            REMB1
                RLCF            REMB0
                MOVFP           BARGB2,WREG
                BTFSS           ACCB0,LSB
                DM: 10

```

```

                GOTO          UADD23#v(i)
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
UADD23#v(i)    GOTO          UOK23#v(i)
                ADDWF         REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
UOK23#v(i)    RLCF          ACCB0
                i=i+1
                endw
                RLCF          ACCB1,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                BTFSS         ACCB0,LSB
                GOTO          UADD238
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                GOTO          UOK238
UADD238      ADDWF         REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
UOK238      RLCF          ACCB1
                i = 9
                while i < 16
                RLCF          ACCB1,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                BTFSS         ACCB1,LSB
                GOTO          UADD23#v(i)
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                GOTO          UOK23#v(i)
UADD23#v(i)  ADDWF         REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
UOK23#v(i)  RLCF          ACCB1
                i=i+1
                endw
                RLCF          ACCB2,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                BTFSS         ACCB1,LSB
                GOTO          UADD2316
                SUBWF         REMB2
                MOVFP         BARGB1,WREG

```

AN617

```

SUBWFB          REMB1
MOVFP           BARGB0,WREG
SUBWFB          REMB0
GOTO            UOK2316
UADD2316        ADDWF          REMB2
MOVFP           BARGB1,WREG
ADDWFC          REMB1
MOVFP           BARGB0,WREG
ADDWFC          REMB0
UOK2316        RLCF          ACCB2
               i = 17
               while i < 24
RLCF            ACCB2,W
RLCF            REMB2
RLCF            REMB1
RLCF            REMB0
MOVFP           BARGB2,WREG
BTFSS          ACCB2,LSB
GOTO            UADD23#v(i)
SUBWF          REMB2
MOVFP           BARGB1,WREG
SUBWFB          REMB1
MOVFP           BARGB0,WREG
SUBWFB          REMB0
GOTO            UOK23#v(i)
UADD23#v(i)    ADDWF          REMB2
MOVFP           BARGB1,WREG
ADDWFC          REMB1
MOVFP           BARGB0,WREG
ADDWFC          REMB0
UOK23#v(i)    RLCF          ACCB2
               i=i+1
               endw
RLCF            ACCB3,W
RLCF            REMB2
RLCF            REMB1
RLCF            REMB0
MOVFP           BARGB2,WREG
BTFSS          ACCB2,LSB
GOTO            UADD2324
SUBWF          REMB2
MOVFP           BARGB1,WREG
SUBWFB          REMB1
MOVFP           BARGB0,WREG
SUBWFB          REMB0
GOTO            UOK2324
UADD2324        ADDWF          REMB2
MOVFP           BARGB1,WREG
ADDWFC          REMB1
MOVFP           BARGB0,WREG
ADDWFC          REMB0
UOK2324        RLCF          ACCB3
               i = 25
               while i < 32
RLCF            ACCB3,W
RLCF            REMB2
RLCF            REMB1
RLCF            REMB0
MOVFP           BARGB2,WREG
BTFSS          ACCB3,LSB
GOTO            UADD23#v(i)
SUBWF          REMB2
MOVFP           BARGB1,WREG
SUBWFB          REMB1
MOVFP           BARGB0,WREG
SUBWFB          REMB0
```



```

UADD23#v(i)    GOTO          UOK23#v(i)
                ADDWF        REMB2
                MOVFP        BARGB1,WREG
                ADDWFC       REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0
UOK23#v(i)     RLCF          ACCB3
                i=i+1
                endw
                BTFSC        ACCB3,LSB
                GOTO         UOK23
                MOVFP        BARGB2,WREG
                ADDWF        REMB2
                MOVFP        BARGB1,WREG
                ADDWFC       REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0
UOK23
                endm
UDIV3123       macro
;           Max Timing:      7+11+30*15+8 = 476 clks
;           Min Timing:      7+11+30*14+3 = 441 clks
;           PM: 7+11+30*19+8 = 596           DM: 10
                variable i
                MOVFP        BARGB2,WREG
                SUBWF        REMB2
                MOVFP        BARGB1,WREG
                SUBWFB       REMB1
                MOVFP        BARGB0,WREG
                SUBWFB       REMB0
                RLCF         ACCB0
                RLCF         ACCB0,W
                RLCF         REMB2
                RLCF         REMB1
                RLCF         REMB0
                MOVFP        BARGB2,WREG
                ADDWF        REMB2
                MOVFP        BARGB1,WREG
                ADDWFC       REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0
                RLCF         ACCB0
                i = 2
                while i < 8
                RLCF         ACCB0,W
                RLCF         REMB2
                RLCF         REMB1
                RLCF         REMB0
                MOVFP        BARGB2,WREG
                BTFSS        ACCB0,LSB
                GOTO         UADD13#v(i)
                SUBWF        REMB2
                MOVFP        BARGB1,WREG
                SUBWFB       REMB1
                MOVFP        BARGB0,WREG
                SUBWFB       REMB0
                GOTO         UOK13#v(i)
UADD13#v(i)    ADDWF        REMB2
                MOVFP        BARGB1,WREG
                ADDWFC       REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0
UOK13#v(i)     RLCF          ACCB0
                i=i+1
                endw
                RLCF         ACCB1,W

```

AN617

```

        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        MOVFP        BARGB2,WREG
        BTFSS        ACCB0,LSB
        GOTO         UADD138
        SUBWF        REMB2
        MOVFP        BARGB1,WREG
        SUBWFB       REMB1
        MOVFP        BARGB0,WREG
        SUBWFB       REMB0
        GOTO         UOK138
UADD138  ADDWF        REMB2
        MOVFP        BARGB1,WREG
        ADDWFC       REMB1
        MOVFP        BARGB0,WREG
        ADDWFC       REMB0
UOK138   RLCF          ACCB1
        i = 9
        while i < 16
        RLCF          ACCB1,W
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        MOVFP        BARGB2,WREG
        BTFSS        ACCB1,LSB
        GOTO         UADD13#v(i)
        SUBWF        REMB2
        MOVFP        BARGB1,WREG
        SUBWFB       REMB1
        MOVFP        BARGB0,WREG
        SUBWFB       REMB0
        GOTO         UOK13#v(i)
UADD13#v(i)  ADDWF        REMB2
        MOVFP        BARGB1,WREG
        ADDWFC       REMB1
        MOVFP        BARGB0,WREG
        ADDWFC       REMB0
UOK13#v(i)  RLCF          ACCB1
        i=i+1
        endw
        RLCF          ACCB2,W
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
        MOVFP        BARGB2,WREG
        BTFSS        ACCB1,LSB
        GOTO         UADD1316
        SUBWF        REMB2
        MOVFP        BARGB1,WREG
        SUBWFB       REMB1
        MOVFP        BARGB0,WREG
        SUBWFB       REMB0
        GOTO         UOK1316
UADD1316  ADDWF        REMB2
        MOVFP        BARGB1,WREG
        ADDWFC       REMB1
        MOVFP        BARGB0,WREG
        ADDWFC       REMB0
UOK1316   RLCF          ACCB2
        i = 17
        while i < 24
        RLCF          ACCB2,W
        RLCF          REMB2
        RLCF          REMB1
        RLCF          REMB0
```

```

MOVFP          BARGB2,WREG
BTFSS         ACCB2,LSB
GOTO          UADD13#v(i)
SUBWF         REMB2
MOVFP          BARGB1,WREG
SUBWFB        REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO          UOK13#v(i)
UADD13#v(i)    ADDWF         REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK13#v(i)    RLCF          ACCB2
               i=i+1
               endw
RLCF          ACCB3,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP          BARGB2,WREG
BTFSS         ACCB2,LSB
GOTO          UADD1324
SUBWF         REMB2
MOVFP          BARGB1,WREG
SUBWFB        REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO          UOK1324
UADD1324      ADDWF         REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK1324      RLCF          ACCB3
               i = 25
               while i < 32
RLCF          ACCB3,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP          BARGB2,WREG
BTFSS         ACCB3,LSB
GOTO          UADD13#v(i)
SUBWF         REMB2
MOVFP          BARGB1,WREG
SUBWFB        REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO          UOK13#v(i)
UADD13#v(i)    ADDWF         REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK13#v(i)    RLCF          ACCB3
               i=i+1
               endw
BTFSC         ACCB3,LSB
GOTO          UOK13
MOVFP          BARGB2,WREG
ADDWF         REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG

```

AN617

```
                ADDWFC          REMB0
UOK13
                endm
;*****
;*****
;
;      32/32 Bit Signed Fixed Point Divide 32/32 -> 32.32
;      Input:  32 bit signed fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;      Use:    CALL    FXD3232S
;      Output: 32 bit signed fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;             32 bit fixed point remainder in REMB0, REMB1, REMB2, REMB3
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing: 13+573+3 = 589 clks          A > 0, B > 0
;                20+573+21 = 614 clks         A > 0, B < 0
;                20+573+21 = 614 clks         A < 0, B > 0
;                27+573+3 = 603 clks          A < 0, B < 0
;      Min Timing: 13+536+3 = 552 clks         A > 0, B > 0
;                20+536+21 = 577 clks         A > 0, B < 0
;                20+536+21 = 577 clks         A < 0, B > 0
;                27+536+3 = 566 clks          A < 0, B < 0
;      PM: 27+753+20 = 800          DM: 13
FXD3232S      MOVFP          AARGB0,WREG
              XORWF          BARGB0,W
              MOVWF          SIGN
              CLRF           REMB0
              CLRF           REMB1
              CLRF           REMB2
              CLRF           REMB3,W
              BTFS          BARGB0,MSB      ; if MSB set, negate BARG
              GOTO          CA3232S
              COMF          BARGB3
              COMF          BARGB2
              COMF          BARGB1
              COMF          BARGB0
              INCF          BARGB3
              ADDWFC        BARGB2
              ADDWFC        BARGB1
              ADDWFC        BARGB0
CA3232S      BTFS          AARGB0,MSB      ; if MSB set, negate AARG
              GOTO          C3232S
              COMF          AARGB3
              COMF          AARGB2
              COMF          AARGB1
              COMF          AARGB0
              INCF          AARGB3
              ADDWFC        AARGB2
              ADDWFC        AARGB1
              ADDWFC        AARGB0
C3232S      SDIV3232
              BTFS          SIGN,MSB
              RETLW         0x00
              COMF          AARGB3
              COMF          AARGB2
              COMF          AARGB1
              COMF          AARGB0
              CLRF          WREG
              INCF          AARGB3
              ADDWFC        AARGB2
              ADDWFC        AARGB1
              ADDWFC        AARGB0
              COMF          REMB3
              COMF          REMB2
              COMF          REMB1
              COMF          REMB0
              INCF          REMB3
```

```

                ADDWFC          REMB2
                ADDWFC          REMB1
                ADDWFC          REMB0
                RETLW           0x00
;*****
;*****
;       32/32 Bit Unsigned Fixed Point Divide 32/32 -> 32.32
;       Input:  32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               32 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2, BARGB3
;       Use:    CALL    FXD3232U
;       Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1AARGB2,AARGB3
;               32 bit unsigned fixed point remainder in REMB0, REMB1, REMB2, REMB3
;       Result: AARG, REM <-- AARG / BARG
;       Max Timing: 4+677+2 = 683 clks
;       Min Timing: 4+639+2 = 645 clks
;       PM: 4+925+1 = 930          DM: 13
FXD3232U      CLRF          REMB0
              CLRF          REMB1
              CLRF          REMB2
              CLRF          REMB3
              NDIV3232
              RETLW         0x00
;*****
;*****
;       32/31 Bit Unsigned Fixed Point Divide 32/31 -> 32.31
;       Input:  32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               31 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2, BARGB3
;       Use:    CALL    FXD3231U
;       Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;               31 bit unsigned fixed point remainder in REMB0, REMB1, REMB2, REMB3
;       Result: AARG, REM <-- AARG / BARG
;       Max Timing: 4+582+2 = 588 clks
;       Min Timing: 4+544+2 = 550 clks
;       PM: 4+768+1 = 773          DM: 12
FXD3231U      CLRF          REMB0
              CLRF          REMB1
              CLRF          REMB2
              CLRF          REMB3
              UDIV3231
              RETLW         0x00
;*****
;*****
;       31/31 Bit Unsigned Fixed Point Divide 31/31 -> 31.31
;       Input:  31 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               31 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2, BARGB3
;       Use:    CALL    FXD3131U
;       Output: 31 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;               31 bit unsigned fixed point remainder in REMB0, REMB1, REMB2, REMB3
;       Result: AARG, REM <-- AARG / BARG
;       Max Timing: 4+573+2 = 579 clks
;       Min Timing: 4+536+2 = 542 clks
;       PM: 4+753+1 = 758          DM: 12
FXD3131U      CLRF          REMB0
              CLRF          REMB1
              CLRF          REMB2
              CLRF          REMB3
              UDIV3131
              RETLW         0x00
;*****
;*****
;       32/24 Bit Signed Fixed Point Divide 32/24 -> 32.24
;       Input:  32 bit signed fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               24 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2

```

AN617

```
;      Use:      CALL      FXD3224S
;      Output: 32 bit signed fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;      24 bit fixed point remainder in REMB0, REMB1, REMB2
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:      12+476+3 = 491 clks           A > 0, B > 0
;                      17+476+19 = 512 clks          A > 0, B < 0
;                      19+476+19 = 514 clks          A < 0, B > 0
;                      24+476+3 = 503 clks           A < 0, B < 0
;      Min Timing:      12+441+3 = 456 clks          A > 0, B > 0
;                      17+441+19 = 477 clks          A > 0, B < 0
;                      19+441+19 = 479 clks          A < 0, B > 0
;                      24+441+3 = 468 clks           A < 0, B < 0
;      PM: 24+596+19 = 639          DM: 11
FXD3224S      MOVFP      AARGB0,WREG
              XORWF      BARGB0,W
              MOVWF      SIGN
              CLRF       REMB0
              CLRF       REMB1
              CLRF       REMB2,W
              BTFSS      BARGB0,MSB      ; if MSB set, negate BARG
              GOTO      CA3224S
              COMF       BARGB2
              COMF       BARGB1
              COMF       BARGB0
              INCF       BARGB2
              ADDWFC     BARGB1
              ADDWFC     BARGB0
CA3224S      BTFSS      AARGB0,MSB      ; if MSB set, negate AARG
              GOTO      C3224S
              COMF       AARGB3
              COMF       AARGB2
              COMF       AARGB1
              COMF       AARGB0
              INCF       AARGB3
              ADDWFC     AARGB2
              ADDWFC     AARGB1
              ADDWFC     AARGB0
C3224S      SDIV3224
              BTFSS      SIGN,MSB
              RETLW     0x00
              COMF       AARGB3
              COMF       AARGB2
              COMF       AARGB1
              COMF       AARGB0
              CLRF       WREG
              INCF       AARGB3
              ADDWFC     AARGB2
              ADDWFC     AARGB1
              ADDWFC     AARGB0
              COMF       REMB2
              COMF       REMB1
              COMF       REMB0
              INCF       REMB2
              ADDWFC     REMB1
              ADDWFC     REMB0
              RETLW     0x00
;*****
;*****
;      32/24 Bit Unsigned Fixed Point Divide 32/24 -> 32.24
;      Input: 32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;      24 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
;      Use:      CALL      FXD3224U
;      Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1AARGB2,AARGB3
;      24 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:      3+579+2 = 584 clks
```

```

;      Min Timing:      3+543+2 = 548 clks
;      PM: 3+765+1 = 769          DM: 11
FXD3224U      CLRF          REMB0
              CLRF          REMB1
              CLRF          REMB2
              NDIV3224
              RETLW        0x00
;*****
;*****
;      32/23 Bit Unsigned Fixed Point Divide 32/23 -> 32.23
;      Input: 32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;             23 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
;      Use:  CALL      FXD3223U
;      Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;             23 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      3+484+2 = 489 clks
;      Min Timing:      3+448+2 = 453 clks
;      PM: 3+608+1 = 612          DM: 10
FXD3223U      CLRF          REMB0
              CLRF          REMB1
              CLRF          REMB2
              UDIV3223
              RETLW        0x00
;*****
;*****
;      31/23 Bit Unsigned Fixed Point Divide 31/23 -> 31.23
;      Input: 31 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;             23 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
;      Use:  CALL      FXD3123U
;      Output: 31 bit unsigned fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;             23 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      3+476+2 = 481 clks
;      Min Timing:      3+441+2 = 446 clks
;      PM: 3+596+1 = 600          DM: 10
FXD3123U      CLRF          REMB0
              CLRF          REMB1
              CLRF          REMB2
              UDIV3123
              RETLW        0x00
;*****
;*****
;      END

```

F.2 PIC17CXX Fixed Point Divide Routines B

```

;      PIC17 FIXED POINT DIVIDE ROUTINES B      VERSION 1.8
;      Input: fixed point arguments in AARG and BARG
;      Output: quotient AARG/BARG followed by remainder in REM
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed divide application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;      Routine      Clocks      Function
;      FXD2416S    314 24 bit/16 bit -> 24.16 signed fixed point divide
;      FXD2416U    365 24 bit/16 bit -> 24.16 unsigned fixed point divide
;      FXD2415U    294 24 bit/15 bit -> 24.15 unsigned fixed point divide
;      FXD2315U    287 23 bit/15 bit -> 23.15 unsigned fixed point divide
;      FXD1616S    214 16 bit/16 bit -> 16.16 signed fixed point divide
;      FXD1616U    244 16 bit/16 bit -> 16.16 unsigned fixed point divide
;      FXD1615U    197 16 bit/15 bit -> 16.15 unsigned fixed point divide
;      FXD1515U    191 15 bit/15 bit -> 15.15 unsigned fixed point divide
;      FXD1608S    146 16 bit/08 bit -> 16.08 signed fixed point divide

```

AN617

```
;      FXD1608U    196 16 bit/08 bit -> 16.08 unsigned fixed point divide
;      FXD1607U    130 16 bit/07 bit -> 16.07 unsigned fixed point divide
;      FXD1507U    125 15 bit/07 bit -> 15.07 unsigned fixed point divide
;      FXD0808S    77  08 bit/08 bit -> 08.08 signed fixed point divide
;      FXD0808U    75  08 bit/08 bit -> 08.08 unsigned fixed point divide
;      FXD0807U    66  08 bit/07 bit -> 08.07 unsigned fixed point divide
;      FXD0707U    61  07 bit/07 bit -> 07.07 unsigned fixed point divide
;
;      list      r=dec,x=on,t=off,p=17C42
;      include  <PIC17.INC>      ; general PIC17 definitions
;
;      include  <MATH17.INC>     ; PIC17 math library definitions
;*****
;*****
;      Test suite storage
RANDHI    equ      0x2B      ; random number senerator registers
RANDLO    equ      0x2C
TESTCODE  equ      0x2D      ; integer code labeling test contained in following data
NUMTESTS  equ      0x2E      ; number of tests contained in following data
TESTCOUNT equ     0x2F      ; counter
DATA      equ      0x30      ; beginning of test data
;*****
;*****
;      Test suite for fixed point divide algorithms
;
;      org      0x0021
MAIN      MOVLW    RAMSTART
;
;      MOVPF    WREG,FSR0
MEMLOOP   CLRF     INDF0
;
;      INCFSZ   FSR0
;
;      GOTO     MEMLOOP
;
;      BSF     RTCSTA,5
;
;      MOVPF    RTCCH,WREG
;
;      MOVLW   0x45          ; seed for random numbers
;
;      MOVPF    WREG,RANDLO
;
;      MOVPF    RTCCCL,WREG
;
;      MOVLW   0x30
;
;      MOVPF    WREG,RANDHI
;
;      MOVLW   0x30
;
;      MOVPF    WREG,FSR0
;
;      BCF     _FS1
;
;      BSF     _FS0
;
;      CALL    TFXD0808
;
;      CALL    TFXD1608
;
;      CALL    TFXD1616
;
;      CALL    TFXD2416
;
;      MOVLW   0xFF
;
;      MOVPF    WREG,AARGB0
;
;      MOVLW   0xFF
;
;      MOVPF    WREG,AARGB1
;
;      MOVLW   0xFF
;
;      MOVPF    WREG,AARGB2
;
;      MOVLW   0xFF
;
;      MOVPF    WREG,AARGB3
;
;      MOVLW   0xFF
;
;      MOVPF    WREG,BARGB0
;
;      MOVLW   0xFF
;
;      MOVPF    WREG,BARGB1
;
;      CALL    FXD1616U
SELF      GOTO     SELF
RANDOM16  RLCF    RANDHI,W      ; random number generator
;
;      XORWF   RANDHI,W
;
;      RLCF    WREG
;
;      SWAPF   RANDHI
;
;      SWAPF   RANDLO,W
;
;      RLNCF   WREG
;
;      XORWF   RANDHI,W
;
;      SWAPF   RANDHI
```



```

        ANDLW      0x01
        RLCF      RANDLO
        XORWF     RANDLO
        RLCF      RANDHI

        RETLW     0
;       Test suite for FXD2416
TFXD2416  MOVLW     20
          MOVFPF   WREG,TESTCOUNT
          MOVFPF   WREG,NUMTESTS
          MOVLW    1
          MOVFPF   WREG,TESTCODE

D2416LOOP
          CALL     RANDOM16
          MOVFPF   RANDHI,WREG
          MOVFPF   WREG,BARGB0
          MOVFPF   RANDLO,WREG
          MOVFPF   WREG,BARGB1
;          BCF     BARGB0,MSB
          MOVFPF   BARGB0,INDF0
          MOVFPF   BARGB1,INDF0
          CALL     RANDOM16
          MOVFPF   RANDHI,WREG
          MOVFPF   WREG,AARGB0
          MOVFPF   RANDLO,WREG
          MOVFPF   WREG,AARGB1
          CALL     RANDOM16
          MOVFPF   RANDHI,WREG
          MOVFPF   WREG,AARGB2
;          BCF     AARGB0,MSB
          MOVFPF   AARGB0,INDF0
          MOVFPF   AARGB1,INDF0
          MOVFPF   AARGB2,INDF0
          CALL     FXD2416S
;          CALL    FXD2416U
;          CALL    FXD2415U
;          CALL    FXD2315U
          MOVFPF   AARGB0,INDF0
          MOVFPF   AARGB1,INDF0
          MOVFPF   AARGB2,INDF0
          MOVFPF   REMB0,INDF0
          MOVFPF   REMB1,INDF0
          DECFSZ   TESTCOUNT
          GOTO     D2416LOOP
          RETLW    0x00
;       Test suite for FXD1616
TFXD1616  MOVLW     26
          MOVFPF   WREG,TESTCOUNT
          MOVFPF   WREG,NUMTESTS
          MOVLW    2
          MOVFPF   WREG,TESTCODE

D1616LOOP
          CALL     RANDOM16
;          SWAPF   RANDHI
;          SWAPF   RANDLO
          MOVFPF   RANDHI,WREG
          MOVFPF   WREG,BARGB0
          MOVFPF   RANDLO,WREG
          MOVFPF   WREG,BARGB1
;          BCF     BARGB0,MSB
          MOVFPF   BARGB0,INDF0
          MOVFPF   BARGB1,INDF0
          CALL     RANDOM16

;          SWAPF   RANDHI
;          SWAPF   RANDLO

```

AN617

```
MOVFP      RANDHI , WREG
MOVFP      WREG , AARGB0
MOVFP      RANDLO , WREG
MOVFP      WREG , AARGB1
;
BCF        AARGB0 , MSB
MOVFP      AARGB0 , INDF0
MOVFP      AARGB1 , INDF0
CALL       FXD1616U
MOVFP      AARGB0 , INDF0
MOVFP      AARGB1 , INDF0
MOVFP      REMB0 , INDF0
MOVFP      REMB1 , INDF0
DECFSZ    TESTCOUNT
GOTO      D1616LOOP
RETLW     0x00
;          Test suite for FXD1608
TFXD1608   MOVLW     34
           MOVFP    WREG , TESTCOUNT
           MOVFP    WREG , NUMTESTS
           MOVLW   3
           MOVFP    WREG , TESTCODE
D1608LOOP  CALL     RANDOM16
;
;          SWAPF    RANDHI
;          SWAPF    RANDLO
MOVFP      RANDHI , WREG
MOVFP      WREG , BARGB0
BCF        BARGB0 , MSB
MOVFP      BARGB0 , INDF0
CALL       RANDOM16
;
;          SWAPF    RANDHI
;          SWAPF    RANDLO
MOVFP      RANDHI , WREG
MOVFP      WREG , AARGB0
MOVFP      RANDLO , WREG
MOVFP      WREG , AARGB1
;
BCF        AARGB0 , MSB
MOVFP      AARGB0 , INDF0
MOVFP      AARGB1 , INDF0
CALL       FXD1608S
MOVFP      AARGB0 , INDF0
MOVFP      AARGB1 , INDF0
MOVFP      REMB0 , INDF0
DECFSZ    TESTCOUNT
GOTO      D1608LOOP
RETLW     0x00
;          Test suite for FXD0808
TFXD0808   MOVLW     52
           MOVFP    WREG , TESTCOUNT
           MOVFP    WREG , NUMTESTS
           MOVLW   4
           MOVFP    WREG , TESTCODE
D0808LOOP  CALL     RANDOM16
           MOVFP    RANDHI , WREG
           MOVFP    WREG , BARGB0
;          BCF      BARGB0 , MSB
           MOVFP    BARGB0 , INDF0
           MOVFP    RANDLO , WREG
           MOVFP    WREG , AARGB0
;          BCF      AARGB0 , MSB
           MOVFP    AARGB0 , INDF0
           CALL     FXD0808S
           MOVFP    AARGB0 , INDF0
           MOVFP    REMB0 , INDF0
```

```

                DECFSZ          TESTCOUNT
                GOTO           D0808LOOP
                RETLW          0x00
;*****
;*****
;      24/16 Bit Division Macros
SDIV2416      macro
;      Max Timing:      5+8+22*12+6 = 283 clks
;      Min Timing:      5+8+22*11+3 = 258 clks
;      PM: 5+8+22*14+6 = 327          DM: 8
                variable i
                MOVFP          BARGB1,WREG
                SUBWF          REMB1
                MOVFP          BARGB0,WREG
                SUBWFB         REMB0
                RLCF          ACCB0
                RLCF          ACCB0,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB1,WREG
                ADDWF          REMB1
                MOVFP          BARGB0,WREG
                ADDWFC         REMB0
                RLCF          ACCB0
                i = 2
                while i < 8
                RLCF          ACCB0,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB1,WREG
                BTFSS          ACCB0,LSB
                GOTO          SADD46#v(i)
                SUBWF          REMB1
                MOVFP          BARGB0,WREG
                SUBWFB         REMB0
                GOTO          SOK46#v(i)
SADD46#v(i)    ADDWF          REMB1
                MOVFP          BARGB0,WREG
                ADDWFC         REMB0
SOK46#v(i)    RLCF          ACCB0
                i=i+1
                endw
                RLCF          ACCB1,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB1,WREG
                BTFSS          ACCB0,LSB
                GOTO          SADD468
                SUBWF          REMB1
                MOVFP          BARGB0,WREG
                SUBWFB         REMB0
                GOTO          SOK468
SADD468      ADDWF          REMB1
                MOVFP          BARGB0,WREG
                ADDWFC         REMB0
SOK468      RLCF          ACCB1
                i = 9
                while i < 16
                RLCF          ACCB1,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB1,WREG
                BTFSS          ACCB1,LSB
                GOTO          SADD46#v(i)
                SUBWF          REMB1
                MOVFP          BARGB0,WREG

```

AN617

```
SUBWFB          REMB0
GOTO            SOK46#v(i)
SADD46#v(i)    ADDWF          REMB1
               MOVFP         BARGB0,WREG
               ADDWFC        REMB0
SOK46#v(i)    RLCF           ACCB1
               i=i+1
               endw
               RLCF          ACCB2,W
               RLCF          REMB1
               RLCF          REMB0
               MOVFP         BARGB1,WREG
               BTFSS         ACCB1,LSB
               GOTO          SADD4616
               SUBWF         REMB1
               MOVFP         BARGB0,WREG
               SUBWFB        REMB0
               GOTO          SOK4616
SADD4616      ADDWF          REMB1
               MOVFP         BARGB0,WREG
               ADDWFC        REMB0
SOK4616      RLCF           ACCB2
               i = 17
               while i < 24
               RLCF          ACCB2,W
               RLCF          REMB1
               RLCF          REMB0
               MOVFP         BARGB1,WREG
               BTFSS         ACCB2,LSB
               GOTO          SADD46#v(i)
               SUBWF         REMB1
               MOVFP         BARGB0,WREG
               SUBWFB        REMB0
               GOTO          SOK46#v(i)
SADD46#v(i)    ADDWF          REMB1
               MOVFP         BARGB0,WREG
               ADDWFC        REMB0
SOK46#v(i)    RLCF           ACCB2
               i=i+1
               endw
               BTFSC         ACCB2,LSB
               GOTO          SOK46
               MOVFP         BARGB1,WREG
               ADDWF         REMB1
               MOVFP         BARGB0,WREG
               ADDWFC        REMB0
SOK46
               endm
UDIV2416      macro
;           restore = 15/20 clks, nonrestore = 11/14 clks
;           Max Timing: 16*15+1+8*20 = 401 clks
;           Min Timing: 16*11+1+8*14 = 289 clks
;           PM: 16*15+1+8*20 = 401          DM: 8
               variable      i
               i = 0
               while i < 8
               RLCF          ACCB0,W
               RLCF          REMB1
               RLCF          REMB0
               MOVFP         BARGB1,WREG
               SUBWF         REMB1
               MOVFP         BARGB0,WREG
               SUBWFB        REMB0
               BTFSC         _C
               GOTO          UOK46#v(i)
               MOVFP         BARGB1,WREG
```

```

                ADDWF          REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
                BCF           _C
UOK46#v(i)    RLCF           ACCB0
                i=i+1
                endw
                i = 8
                while i < 16
                RLCF          ACCB1,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB1,WREG
                SUBWF         REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                BTFSC        _C
                GOTO          UOK46#v(i)
                MOVFP         BARGB1,WREG
                ADDWF         REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
                BCF           _C
UOK46#v(i)    RLCF           ACCB1
                i=i+1
                endw
                CLRF          TEMP
                i = 16
                while i < 24
                RLCF          ACCB2,W
                RLCF          REMB1
                RLCF          REMB0
                RLCF          TEMP
                MOVFP         BARGB1,WREG
                SUBWF         REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                CLRF          WREG
                SUBWFB        TEMP
                BTFSC        _C
                GOTO          UOK46#v(i)
                MOVFP         BARGB1,WREG
                ADDWF         REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
                CLRF          WREG
                ADDWFC        TEMP
                BCF           _C
UOK46#v(i)    RLCF           ACCB2
                i=i+1
                endw
                endm
NDIV2416      macro
;           Max Timing:      10+23*15+6 = 361 clks
;           Min Timing:      10+23*14+3 = 335 clks
;           PM: 10+23*19+6 = 450           DM: 8
                variable i
                RLCF          ACCB0,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB1,WREG
                SUBWF         REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                CLRF          TEMP,W
                SUBWFB        TEMP

```

AN617

```

                                RLCF          ACCB0
                                i = 1
                                while i < 8
                                RLCF          ACCB0,W
                                RLCF          REMB1
                                RLCF          REMB0
                                RLCF          TEMP
                                MOVFP        BARGB1,WREG
                                BTFSS        ACCB0,LSB
                                GOTO         NADD46#v(i)
                                SUBWF        REMB1
                                MOVFP        BARGB0,WREG
                                SUBWFB       REMB0
                                CLRF         WREG
                                SUBWFB       TEMP
                                GOTO         NOK46#v(i)
NADD46#v(i)  ADDWF        REMB1
                                MOVFP        BARGB0,WREG
                                ADDWFC       REMB0
                                CLRF         WREG
                                ADDWFC       TEMP

NOK46#v(i)   RLCF          ACCB0
                                i=i+1
                                endw
                                RLCF          ACCB1,W
                                RLCF          REMB1
                                RLCF          REMB0
                                RLCF          TEMP
                                MOVFP        BARGB1,WREG
                                BTFSS        ACCB0,LSB
                                GOTO         NADD468
                                SUBWF        REMB1
                                MOVFP        BARGB0,WREG
                                SUBWFB       REMB0
                                CLRF         WREG
                                SUBWFB       TEMP
                                GOTO         NOK468
NADD468     ADDWF        REMB1
                                MOVFP        BARGB0,WREG
                                ADDWFC       REMB0
                                CLRF         WREG
                                ADDWFC       TEMP

NOK468      RLCF          ACCB1
                                i = 9
                                while i < 16
                                RLCF          ACCB1,W
                                RLCF          REMB1
                                RLCF          REMB0
                                RLCF          TEMP
                                MOVFP        BARGB1,WREG
                                BTFSS        ACCB1,LSB
                                GOTO         NADD46#v(i)
                                SUBWF        REMB1
                                MOVFP        BARGB0,WREG
                                SUBWFB       REMB0
                                CLRF         WREG
                                SUBWFB       TEMP
                                GOTO         NOK46#v(i)
NADD46#v(i)  ADDWF        REMB1
                                MOVFP        BARGB0,WREG
                                ADDWFC       REMB0
                                CLRF         WREG
                                ADDWFC       TEMP
```

```

NOK46#v(i)    RLCF          ACCB1
              i=i+1
              endw
              RLCF          ACCB2,W
              RLCF          REMB1
              RLCF          REMB0
              RLCF          TEMP
              MOVFP        BARGB1,WREG
              BTFSS        ACCB1,LSB
              GOTO         NADD4616
              SUBWF        REMB1
              MOVFP        BARGB0,WREG
              SUBWFB       REMB0
              CLRF         WREG
              SUBWFB       TEMP
              GOTO         NOK4616
NADD4616      ADDWF        REMB1
              MOVFP        BARGB0,WREG
              ADDWFC       REMB0
              CLRF         WREG
              ADDWFC       TEMP

NOK4616      RLCF          ACCB2
              i = 17
              while i < 24
              RLCF          ACCB2,W
              RLCF          REMB1
              RLCF          REMB0
              RLCF          TEMP
              MOVFP        BARGB1,WREG
              BTFSS        ACCB2,LSB
              GOTO         NADD46#v(i)
              SUBWF        REMB1
              MOVFP        BARGB0,WREG
              SUBWFB       REMB0
              CLRF         WREG
              SUBWFB       TEMP
              GOTO         NOK46#v(i)
NADD46#v(i)  ADDWF        REMB1
              MOVFP        BARGB0,WREG
              ADDWFC       REMB0
              CLRF         WREG
              ADDWFC       TEMP

NOK46#v(i)   RLCF          ACCB2
              i=i+1
              endw
              BTFSC        ACCB2,LSB
              GOTO         NOK46
              MOVFP        BARGB1,WREG
              ADDWF        REMB1
              MOVFP        BARGB0,WREG
              ADDWFC       REMB0

NOK46
              endm
UDIV2415     macro
;           Max Timing:      8+23*12+6 = 290 clks
;           Min Timing:      8+23*11+3 = 264 clks
;           PM: 8+23*14+6 = 336
              variable i
              RLCF          ACCB0,W
              RLCF          REMB1
              RLCF          REMB0
              MOVFP        BARGB1,WREG
              SUBWF        REMB1
              MOVFP        BARGB0,WREG

```

AN617

```

SUBWFB          REMB0
RLCF            ACCB0
i = 1
while i < 8
RLCF            ACCB0,W
RLCF            REMB1
RLCF            REMB0
MOVFP          BARGB1,WREG
BTFSS          ACCB0,LSB
GOTO           UADD45#v(i)
SUBWF          REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO           UOK45#v(i)
UADD45#v(i)    ADDWF          REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK45#v(i)    RLCF            ACCB0
i=i+1
endw
RLCF            ACCB1,W
RLCF            REMB1
RLCF            REMB0
MOVFP          BARGB1,WREG
BTFSS          ACCB0,LSB
GOTO           UADD458
SUBWF          REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO           UOK458
UADD458       ADDWF          REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK458       RLCF            ACCB1
i = 9
while i < 16
RLCF            ACCB1,W
RLCF            REMB1
RLCF            REMB0
MOVFP          BARGB1,WREG
BTFSS          ACCB1,LSB
GOTO           UADD45#v(i)
SUBWF          REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO           UOK45#v(i)
UADD45#v(i)    ADDWF          REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK45#v(i)    RLCF            ACCB1
i=i+1
endw
RLCF            ACCB2,W
RLCF            REMB1
RLCF            REMB0
MOVFP          BARGB1,WREG
BTFSS          ACCB1,LSB
GOTO           UADD4516
SUBWF          REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO           UOK4516
UADD4516     ADDWF          REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK4516     RLCF            ACCB2
```



```

        i = 17
        while i < 24
            RLCF          ACCB2,W
            RLCF          REMB1
            RLCF          REMB0
            MOVFP        BARGB1,WREG
            BTFSS        ACCB2,LSB
            GOTO         UADD45#v(i)
            SUBWF        REMB1
            MOVFP        BARGB0,WREG
            SUBWFB       REMB0
            GOTO         UOK45#v(i)
UADD45#v(i)  ADDWF        REMB1
            MOVFP        BARGB0,WREG
            ADDWFC       REMB0
UOK45#v(i)  RLCF          ACCB2
            i=i+1
            endw
            BTFSC        ACCB2,LSB
            GOTO         UOK45
            MOVFP        BARGB1,WREG
            ADDWF        REMB1
            MOVFP        BARGB0,WREG
            ADDWFC       REMB0
UOK45
        endm
UDIV2315    macro
;           Max Timing:    5+8+22*12+6 = 283 clks
;           Min Timing:    5+8+22*11+3 = 258 clks
;           PM: 5+8+22*14+6 = 327          DM: 8
        variable i
            MOVFP        BARGB1,WREG
            SUBWF        REMB1
            MOVFP        BARGB0,WREG
            SUBWFB       REMB0
            RLCF          ACCB0
            RLCF          ACCB0,W
            RLCF          REMB1
            RLCF          REMB0
            MOVFP        BARGB1,WREG
            ADDWF        REMB1
            MOVFP        BARGB0,WREG
            ADDWFC       REMB0
            RLCF          ACCB0
            i = 2
            while i < 8
                RLCF          ACCB0,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP        BARGB1,WREG
                BTFSS        ACCB0,LSB
                GOTO         UADD35#v(i)
                SUBWF        REMB1
                MOVFP        BARGB0,WREG
                SUBWFB       REMB0
            GOTO         UOK35#v(i)
UADD35#v(i)  ADDWF        REMB1
            MOVFP        BARGB0,WREG
            ADDWFC       REMB0
UOK35#v(i)  RLCF          ACCB0
            i=i+1
            endw
            RLCF          ACCB1,W
            RLCF          REMB1
            RLCF          REMB0
            MOVFP        BARGB1,WREG

```

AN617

```

                BTFSS          ACCB0, LSB
                GOTO          UADD358
                SUBWF         REMB1
                MOVFP         BARGB0, WREG
                SUBWFB        REMB0
                GOTO          UOK358
UADD358        ADDWF         REMB1
                MOVFP         BARGB0, WREG
                ADDWFC        REMB0
UOK358        RLCF          ACCB1
                i = 9
                while i < 16
                RLCF          ACCB1, W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB1, WREG
                BTFSS          ACCB1, LSB
                GOTO          UADD35#v(i)
                SUBWF         REMB1
                MOVFP         BARGB0, WREG
                SUBWFB        REMB0
                GOTO          UOK35#v(i)
UADD35#v(i)   ADDWF         REMB1
                MOVFP         BARGB0, WREG
                ADDWFC        REMB0
UOK35#v(i)   RLCF          ACCB1
                i=i+1
                endw
                RLCF          ACCB2, W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB1, WREG
                BTFSS          ACCB1, LSB
                GOTO          UADD3516
                SUBWF         REMB1
                MOVFP         BARGB0, WREG
                SUBWFB        REMB0
                GOTO          UOK3516
UADD3516     ADDWF         REMB1
                MOVFP         BARGB0, WREG
                ADDWFC        REMB0
UOK3516     RLCF          ACCB2
                i = 17
                while i < 24
                RLCF          ACCB2, W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB1, WREG
                BTFSS          ACCB2, LSB
                GOTO          UADD35#v(i)
                SUBWF         REMB1
                MOVFP         BARGB0, WREG
                SUBWFB        REMB0
                GOTO          UOK35#v(i)
UADD35#v(i)   ADDWF         REMB1
                MOVFP         BARGB0, WREG
                ADDWFC        REMB0
UOK35#v(i)   RLCF          ACCB2
                i=i+1
                endw
                BTFSC          ACCB2, LSB
                GOTO          UOK35
                MOVFP         BARGB1, WREG
                ADDWF         REMB1
                MOVFP         BARGB0, WREG
                ADDWFC        REMB0
```

```

UOK35
        endm
;*****
;*****
;    16/16 Bit Division Macros
SDIV1616    macro
;    Max Timing:    5+8+14*12+6 = 187 clks
;    Min Timing:    5+8+14*11+6 = 173 clks
;    PM: 5+8+14*14+6 = 215          DM: 6
        variable i
                MOVFP        BARGB1,WREG
                SUBWF        REMB1
                MOVFP        BARGB0,WREG
                SUBWFB       REMB0
                RLCF        ACCB0
                RLCF        ACCB0,W
                RLCF        REMB1
                RLCF        REMB0
                MOVFP        BARGB1,WREG
                ADDWF        REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0
                RLCF        ACCB0
                i = 2
                while i < 8
                        RLCF        ACCB0,W
                        RLCF        REMB1
                        RLCF        REMB0
                        MOVFP        BARGB1,WREG
                        BTFSS       ACCB0,LSB
                        GOTO        SADD66#v(i)
                        SUBWF        REMB1
                        MOVFP        BARGB0,WREG
                        SUBWFB       REMB0
                        GOTO        SOK66#v(i)
SADD66#v(i)    ADDWF        REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0

SOK66#v(i)    RLCF        ACCB0
                i=i+1
                endw
                RLCF        ACCB1,W
                RLCF        REMB1
                RLCF        REMB0
                MOVFP        BARGB1,WREG
                BTFSS       ACCB0,LSB
                GOTO        SADD668
                SUBWF        REMB1
                MOVFP        BARGB0,WREG
                SUBWFB       REMB0
                GOTO        SOK668
SADD668      ADDWF        REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0

SOK668      RLCF        ACCB1
                i = 9
                while i < 16
                        RLCF        ACCB1,W
                        RLCF        REMB1
                        RLCF        REMB0
                        MOVFP        BARGB1,WREG
                        BTFSS       ACCB1,LSB
                        GOTO        SADD66#v(i)
                        SUBWF        REMB1

```

AN617

```

                MOVFP          BARGB0,WREG
                SUBWFB         REMB0
SADD66#v(i)    GOTO          SOK66#v(i)
                ADDWF          REMB1
                MOVFP          BARGB0,WREG
                ADDWFC         REMB0

SOK66#v(i)     RLCF           ACCB1
                i=i+1
                endw
                BTFSC         ACCB1,LSB
                GOTO          SOK66
                MOVFP          BARGB1,WREG
                ADDWF          REMB1
                MOVFP          BARGB0,WREG
                ADDWFC         REMB0

SOK66
                endm
UDIV1616      macro
;             restore = 15 clks, nonrestore = 11 clks
;             Max Timing: 8*15+8*15 = 240 clks
;             Min Timing: 8*11+8*11 = 176 clks
;             PM: 8*15+8*15 = 240           DM: 6
                variable      i
                i = 0
                while i < 8
                RLCF           ACCB0,W
                RLCF           REMB1
                RLCF           REMB0
                MOVFP          BARGB1,WREG
                SUBWF          REMB1
                MOVFP          BARGB0,WREG
                SUBWFB         REMB0
                BTFSC         _C
                GOTO          UOK66#v(i)
                MOVFP          BARGB1,WREG
                ADDWF          REMB1
                MOVFP          BARGB0,WREG
                ADDWFC         REMB0
                BCF           _C
UOK66#v(i)    RLCF           ACCB0
                i=i+1
                endw
                i = 8
                while i < 16
                RLCF           ACCB1,W
                RLCF           REMB1
                RLCF           REMB0
                MOVFP          BARGB1,WREG
                SUBWF          REMB1
                MOVFP          BARGB0,WREG
                SUBWFB         REMB0
                BTFSC         _C
                GOTO          UOK66#v(i)
                MOVFP          BARGB1,WREG
                ADDWF          REMB1
                MOVFP          BARGB0,WREG
                ADDWFC         REMB0
                BCF           _C
UOK66#v(i)    RLCF           ACCB1
                i=i+1
                endw
                endm
NDIV1616      macro
;             Max Timing:      9+15*15+6 = 240 clks
;             Min Timing:      9+15*14+6 = 225 clks
```

```

;      PM: 9+15*19+6 = 300          DM: 7
      variable i
      RLCF          ACCB0,W
      RLCF          REMB1
      MOVFP        BARGB1,WREG
      SUBWF        REMB1
      MOVFP        BARGB0,WREG
      SUBWFB       REMB0
      CLRF         TEMP,W
      SUBWFB       TEMP
      RLCF         ACCB0
      i = 1
      while i < 8
      RLCF         ACCB0,W
      RLCF         REMB1
      RLCF         REMB0
      RLCF         TEMP
      MOVFP        BARGB1,WREG
      BTFSS        ACCB0,LSB
      GOTO         NADD66#v(i)
      SUBWF        REMB1
      MOVFP        BARGB0,WREG
      SUBWFB       REMB0
      CLRF         WREG
      SUBWFB       TEMP
      GOTO         NOK66#v(i)
NADD66#v(i)      ADDWF        REMB1
                  MOVFP        BARGB0,WREG
                  ADDWFC       REMB0
                  CLRF         WREG
                  ADDWFC       TEMP

NOK66#v(i)      RLCF         ACCB0
                  i=i+1
                  endw
                  RLCF         ACCB1,W
                  RLCF         REMB1
                  RLCF         REMB0
                  RLCF         TEMP
                  MOVFP        BARGB1,WREG
                  BTFSS        ACCB0,LSB
                  GOTO         NADD668
                  SUBWF        REMB1
                  MOVFP        BARGB0,WREG
                  SUBWFB       REMB0
                  CLRF         WREG
                  SUBWFB       TEMP
                  GOTO         NOK668
NADD668        ADDWF        REMB1
                  MOVFP        BARGB0,WREG
                  ADDWFC       REMB0
                  CLRF         WREG
                  ADDWFC       TEMP

NOK668        RLCF         ACCB1
                  i = 9
                  while i < 16
                  RLCF         ACCB1,W
                  RLCF         REMB1
                  RLCF         REMB0
                  RLCF         TEMP
                  MOVFP        BARGB1,WREG
                  BTFSS        ACCB1,LSB
                  GOTO         NADD66#v(i)
                  SUBWF        REMB1
                  MOVFP        BARGB0,WREG

```

AN617

```

                SUBWFB      REMB0
                CLRF        WREG
                SUBWFB      TEMP
NADD66#v(i)    GOTO        NOK66#v(i)
                ADDWF       REMB1
                MOVFP       BARGB0,WREG
                ADDWFC      REMB0
                CLRF        WREG
                ADDWFC      TEMP

NOK66#v(i)    RLCF         ACCB1
                i=i+1
                endw
                BTFSC      ACCB1,LSB
                GOTO        NOK66
                MOVFP       BARGB1,WREG
                ADDWF       REMB1
                MOVFP       BARGB0,WREG
                ADDWFC      REMB0

NOK66
                endm
UDIV1615      macro
;           Max Timing:    7+15*12+6 = 193 clks
;           Min Timing:    7+15*11+6 = 178 clks
;           PM: 7+15*14+6 = 213           DM: 6
                variable i
                RLCF        ACCB0,W
                RLCF        REMB1
                MOVFP       BARGB1,WREG
                SUBWF       REMB1
                MOVFP       BARGB0,WREG
                SUBWFB      REMB0
                RLCF        ACCB0
                i = 1
                while i < 8
                RLCF        ACCB0,W
                RLCF        REMB1
                RLCF        REMB0
                MOVFP       BARGB1,WREG
                BTFSS      ACCB0,LSB
                GOTO        UADD65#v(i)
                SUBWF       REMB1
                MOVFP       BARGB0,WREG
                SUBWFB      REMB0
UADD65#v(i)    GOTO        UOK65#v(i)
                ADDWF       REMB1
                MOVFP       BARGB0,WREG
                ADDWFC      REMB0

UOK65#v(i)    RLCF         ACCB0
                i=i+1
                endw
                RLCF        ACCB1,W
                RLCF        REMB1
                RLCF        REMB0
                MOVFP       BARGB1,WREG
                BTFSS      ACCB0,LSB
                GOTO        UADD658
                SUBWF       REMB1
                MOVFP       BARGB0,WREG
                SUBWFB      REMB0
                GOTO        UOK658
UADD658      ADDWF       REMB1
                MOVFP       BARGB0,WREG
                ADDWFC      REMB0
```

```

UOK658      RLCF      ACCB1
             i = 9
             while i < 16
RLCF        ACCB1,W
RLCF        REMB1
RLCF        REMB0
MOVFP      BARGB1,WREG
BTFSS     ACCB1,LSB
GOTO      UADD65#v(i)
SUBWF     REMB1
MOVFP      BARGB0,WREG
SUBWFB    REMB0
GOTO      UOK65#v(i)
UADD65#v(i) ADDWF     REMB1
           MOVFP    BARGB0,WREG
           ADDWFC   REMB0

UOK65#v(i)  RLCF      ACCB1
             i=i+1
             endw
BTFSC     ACCB1,LSB
GOTO      UOK65
MOVFP      BARGB1,WREG
ADDWF     REMB1
MOVFP      BARGB0,WREG
ADDWFC    REMB0

UOK65      endm
UDIV1515   macro
;          Max Timing:      5+8+14*12+6 = 187 clks
;          Min Timing:      5+8+14*11+6 = 173 clks
;          PM: 5+8+14*14+6 = 215          DM: 6
             variable i
MOVFP      BARGB1,WREG
SUBWF     REMB1
MOVFP      BARGB0,WREG
SUBWFB    REMB0
RLCF      ACCB0
RLCF      ACCB0,W
RLCF      REMB1
RLCF      REMB0
MOVFP      BARGB1,WREG
ADDWF     REMB1
MOVFP      BARGB0,WREG
ADDWFC    REMB0
RLCF      ACCB0
             i = 2
             while i < 8
RLCF      ACCB0,W
RLCF      REMB1
RLCF      REMB0
MOVFP      BARGB1,WREG
BTFSS     ACCB0,LSB
GOTO      UADD55#v(i)
SUBWF     REMB1
MOVFP      BARGB0,WREG
SUBWFB    REMB0
GOTO      UOK55#v(i)
UADD55#v(i) ADDWF     REMB1
           MOVFP    BARGB0,WREG
           ADDWFC   REMB0

UOK55#v(i)  RLCF      ACCB0
             i=i+1
             endw
RLCF      ACCB1,W

```

AN617

```

                RLCF          REMB1
                RLCF          REMB0
                MOVFP        BARGB1,WREG
                BTFSS        ACCB0,LSB
                GOTO         UADD558
                SUBWF        REMB1
                MOVFP        BARGB0,WREG
                SUBWFB       REMB0
                GOTO         UOK558
UADD558        ADDWF        REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0

UOK558        RLCF          ACCB1
                i = 9
                while i < 16
                RLCF          ACCB1,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP        BARGB1,WREG
                BTFSS        ACCB1,LSB
                GOTO         UADD55#v(i)
                SUBWF        REMB1
                MOVFP        BARGB0,WREG
                SUBWFB       REMB0
UADD55#v(i)   ADDWF        REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0

UOK55#v(i)   RLCF          ACCB1
                i=i+1
                endw
                BTFSC        ACCB1,LSB
                GOTO         UOK55
                MOVFP        BARGB1,WREG
                ADDWF        REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0
UOK55        endm

;
; Extra 16 Bit Divide Macros
;
DIV1616      macro
; Timing: restore = 16 clks, nonrestore = 13 clks          16*16 = 256 clks
                variable i
                i = 0
                while i < 16
                RLCF          AARGB1
                RLCF          AARGB0
                RLCF          REMB1
                RLCF          REMB0
                MOVFP        BARGB1,WREG
                SUBWF        REMB1
                MOVFP        BARGB0,WREG
                SUBWFB       REMB0
                BTFSS        _C
                GOTO         RS1616_#v( i )
                BSF          AARGB1,LSB
                GOTO         OK1616_#v( i )
RS1616_#v( i ) MOVFP        BARGB1,WREG
                ADDWF        REMB1
                MOVFP        BARGB0,WREG
                ADDWFC       REMB0
                BCF          AARGB1,LSB
```



```

OK1616_#v(i)
    i=i+1
endw
endm

DIVMAC
macro
;      Timing: restore = 19 clks, nonrestore = 14 clks      16*19 = 304 clks
    variable i
    i = 0
    while i < 16
        RLCF      AARGB1
        RLCF      AARGB0
        RLCF      REMB1
        RLCF      REMB0
        MOVFP     BARGB0,WREG
        SUBWF     REMB0,W
        BTFSS     _Z
        GOTO     notz#v( i )
        MOVFP     BARGB1,WREG
        SUBWF     REMB1,W
notz#v( i )
        BTFSS     _C
        GOTO     nosub#v( i )
        MOVFP     BARGB1,WREG
        SUBWF     REMB1
        MOVFP     BARGB0,WREG
        SUBWFB    REMB0
        BSF      AARGB1,LSB
        GOTO     ok#v(i)
nosub#v(i)
        BCF      AARGB1,LSB
ok#v(i)

        i=i+1
    endw
endm

;*****
;*****
;      16/08 BIT Division Macros
SDIV1608
macro
;      Max Timing:      3+5+14*8+2 = 122 clks
;      Min Timing:      3+5+14*8+2 = 122 clks
;      PM: 3+5+14*8+2 = 122      DM: 4
    variable i
        MOVFP     BARGB0,WREG
        SUBWF     REMB0
        RLCF      ACCB0
        RLCF      ACCB0,W
        RLCF      REMB0
        MOVFP     BARGB0,WREG
        ADDWF     REMB0
        RLCF      ACCB0
        i = 2
        while i < 8
            RLCF      ACCB0,W
            RLCF      REMB0
            MOVFP     BARGB0,WREG
            BTFSC     ACCB0,LSB
            SUBWF     REMB0
            BTFSS     ACCB0,LSB
            ADDWF     REMB0
            RLCF      ACCB0
            i=i+1
        endw
        RLCF      ACCB1,W
        RLCF      REMB0
        MOVFP     BARGB0,WREG
        BTFSC     ACCB0,LSB
        SUBWF     REMB0

```

AN617

```

        BTFSS          ACCB0,LSB
        ADDWF          REMB0
        RLCF           ACCB1
        i = 9
        while i < 16
        RLCF           ACCB1,W
        RLCF           REMB0
        MOVFP          BARGB0,WREG
        BTFSC          ACCB1,LSB
        SUBWF          REMB0
        BTFSS          ACCB1,LSB
        ADDWF          REMB0
        RLCF           ACCB1
        i=i+1
        endw
        BTFSS          ACCB1,LSB
        ADDWF          REMB0
        endm
UDIV1608 macro
;       restore = 9/15 clks, nonrestore = 8/11 clks
;       Max Timing: 8*9+1+8*15 = 193 clks          max
;       Min Timing: 8*8+1+8*11 = 153 clks          min
;       PM: 8*9+1+8*15 = 193          DM: 4
        variable      i
        i = 0
        while i < 8
        RLCF           ACCB0,W
        RLCF           REMB0
        MOVFP          BARGB0,WREG
        SUBWF          REMB0
        BTFSC          _C
        GOTO           UOK68#v(i)
        ADDWF          REMB0
        BCF            _C
UOK68#v(i) RLCF           ACCB0
        i=i+1
        endw
        CLRF           TEMP
        i = 8
        while i < 16
        RLCF           ACCB1,W
        RLCF           REMB0
        RLCF           TEMP
        MOVFP          BARGB0,WREG
        SUBWF          REMB0
        CLRF           WREG
        SUBWFB         TEMP
        BTFSC          _C
        GOTO           UOK68#v(i)
        MOVFP          BARGB0,WREG
        ADDWF          REMB0
        CLRF           WREG
        ADDWFC         TEMP
        BCF            _C
UOK68#v(i) RLCF           ACCB1
        i=i+1
        endw
        endm
NDIV1608 macro
;       Max Timing:      7+15*12+3 = 190 clks
;       Min Timing:     7+15*11+3 = 175 clks
;       PM: 7+15*14+3 = 220          DM: 5
        variable      i
        RLCF           ACCB0,W
        RLCF           REMB0
        MOVFP          BARGB0,WREG
```

```

SUBWF          REMB0
CLRF           TEMP,W
SUBWFB        TEMP
RLCF          ACCB0
i = 1
while i < 8
RLCF          ACCB0,W
RLCF          REMB0
RLCF          TEMP
MOVFP        BARGB0,WREG
BTFSS        ACCB0,LSB
GOTO         NADD68#v(i)
SUBWF        REMB0
CLRF         WREG
SUBWFB       TEMP
GOTO         NOK68#v(i)
NADD68#v(i)  ADDWF        REMB0
CLRF         WREG
ADDWFC       TEMP
NOK68#v(i)  RLCF          ACCB0
i=i+1
endw
RLCF          ACCB1,W
RLCF          REMB0
RLCF          TEMP
MOVFP        BARGB0,WREG
BTFSS        ACCB0,LSB
GOTO         NADD688
SUBWF        REMB0
CLRF         WREG
SUBWFB       TEMP
GOTO         NOK688
NADD688     ADDWF        REMB0
CLRF         WREG
ADDWFC       TEMP
NOK688     RLCF          ACCB1
i = 9
while i < 16
RLCF          ACCB1,W
RLCF          REMB0
RLCF          TEMP
MOVFP        BARGB0,WREG
BTFSS        ACCB1,LSB
GOTO         NADD68#v(i)
SUBWF        REMB0
CLRF         WREG
SUBWFB       TEMP
GOTO         NOK68#v(i)
NADD68#v(i)  ADDWF        REMB0
CLRF         WREG
ADDWFC       TEMP
NOK68#v(i)  RLCF          ACCB1
i=i+1
endw
BTFSS        ACCB1,LSB
MOVFP        BARGB0,WREG
ADDWF        REMB0
endm
UDIV1607    macro
;      Max Timing:      5+15*8+2 = 127 clks
;      Min Timing:      5+15*8+2 = 127 clks
;      PM: 5+15*8+2 = 127          DM: 4
      variable i
      RLCF          ACCB0,W
      RLCF          REMB0
      MOVFP        BARGB0,WREG

```

AN617

```
SUBWF          REMB0
RLCF          ACCB0
i = 1
while i < 8
RLCF          ACCB0,W
RLCF          REMB0
MOVFP        BARGB0,WREG
BTFSC        ACCB0,LSB
SUBWF        REMB0
BTFSS        ACCB0,LSB
ADDWF        REMB0
RLCF          ACCB0
i=i+1
endw
RLCF          ACCB1,W
RLCF          REMB0
MOVFP        BARGB0,WREG
BTFSC        ACCB0,LSB
SUBWF        REMB0
BTFSS        ACCB0,LSB
ADDWF        REMB0
RLCF          ACCB1
i = 9
while i < 16
RLCF          ACCB1,W
RLCF          REMB0
MOVFP        BARGB0,WREG
BTFSC        ACCB1,LSB
SUBWF        REMB0
BTFSS        ACCB1,LSB
ADDWF        REMB0
RLCF          ACCB1
i=i+1
endw
BTFSS        ACCB1,LSB
ADDWF        REMB0
endm
UDIV1507
macro
;      Max Timing:      3+5+14*8+2 = 122 clks
;      Min Timing:      3+5+14*8+2 = 122 clks
;      PM: 3+5+14*8+2 = 122          DM: 4
    variable i
    MOVFP        BARGB0,WREG
    SUBWF        REMB0
    RLCF          ACCB0
    RLCF          ACCB0,W
    RLCF          REMB0
    MOVFP        BARGB0,WREG
    ADDWF        REMB0
    RLCF          ACCB0
    i = 2
    while i < 8
    RLCF          ACCB0,W
    RLCF          REMB0
    MOVFP        BARGB0,WREG
    BTFSC        ACCB0,LSB
    SUBWF        REMB0
    BTFSS        ACCB0,LSB
    ADDWF        REMB0
    RLCF          ACCB0
    i=i+1
    endw
    RLCF          ACCB1,W
    RLCF          REMB0
    MOVFP        BARGB0,WREG
    BTFSC        ACCB0,LSB
```

```

SUBWF          REMB0
BTFSS          ACCB0,LSB
ADDWF          REMB0
RLCF           ACCB1
i = 9
while i < 16
RLCF           ACCB1,W
RLCF           REMB0
MOVFP         BARGB0,WREG
BTFSC        ACCB1,LSB
SUBWF        REMB0
BTFSS        ACCB1,LSB
ADDWF        REMB0
RLCF         ACCB1
i=i+1
endw
BTFSS        ACCB1,LSB
ADDWF        REMB0
endm

;*****
;*****
;      08/08 BIT Division Macros
SDIV0808      macro
;      Max Timing:      3+5+6*8+2 = 58 clks
;      Min Timing:      3+5+6*8+2 = 58 clks
;      PM: 3+5+6*8+2 = 58          DM: 3
      variable i
      MOVFP         BARGB0,WREG
      SUBWF        REMB0
      RLCF         ACCB0
      RLCF         ACCB0,W
      RLCF         REMB0
      MOVFP         BARGB0,WREG
      ADDWF        REMB0
      RLCF         ACCB0
      i = 2
      while i < 8
      RLCF         ACCB0,W
      RLCF         REMB0
      MOVFP         BARGB0,WREG
      BTFSC        ACCB0,LSB
      SUBWF        REMB0
      BTFSS        ACCB0,LSB
      ADDWF        REMB0
      RLCF         ACCB0
      i=i+1
      endw
      BTFSS        ACCB0,LSB
      ADDWF        REMB0
      endm
UDIV0808      macro
;      restore = 9 clks, nonrestore = 8 clks
;      Max Timing: 8*9 = 72 clks      max
;      Min Timing: 8*8 = 64 clks      min
;      PM: 8*9 = 72          DM: 3
      variable      i
      i = 0
      while i < 8
      RLCF          ACCB0,W
      RLCF          REMB0
      MOVFP         BARGB0,WREG
      SUBWF        REMB0
      BTFSC        _C
      GOTO         UOK88#v(i)
      ADDWF        REMB0

```

AN617

```

        BCF          _C
UOK88#v(i)  RLCF          ACCB0
            i=i+1
            endw
            endm
UDIV0807  macro
;          Max Timing:      5+7*8+2 = 63 clks
;          Min Timing:      5+7*8+2 = 63 clks
;          PM: 5+7*8+2 = 63          DM: 3
            variable i
            RLCF          ACCB0,W
            RLCF          REMB0
            MOVFP         BARGB0,WREG
            SUBWF         REMB0
            RLCF          ACCB0
            i = 1
            while i < 8
            RLCF          ACCB0,W
            RLCF          REMB0
            MOVFP         BARGB0,WREG
            BTFSC         ACCB0,LSB
            SUBWF         REMB0
            BTFSS         ACCB0,LSB
            ADDWF         REMB0
            RLCF          ACCB0
            i=i+1
            endw
            BTFSS         ACCB0,LSB
            ADDWF         REMB0
            endm
UDIV0707  macro
;          Max Timing:      3+5+6*8+2 = 58 clks
;          Min Timing:      3+5+6*8+2 = 58 clks
;          PM: 3+5+6*8+2 = 58          DM: 3
            variable i
            MOVFP         BARGB0,WREG
            SUBWF         REMB0
            RLCF          ACCB0
            RLCF          ACCB0,W
            RLCF          REMB0
            MOVFP         BARGB0,WREG
            ADDWF         REMB0
            RLCF          ACCB0
            i = 2
            while i < 8
            RLCF          ACCB0,W
            RLCF          REMB0
            MOVFP         BARGB0,WREG
            BTFSC         ACCB0,LSB
            SUBWF         REMB0
            BTFSS         ACCB0,LSB
            ADDWF         REMB0
            RLCF          ACCB0
            i=i+1
            endw
            BTFSS         ACCB0,LSB
            ADDWF         REMB0
            endm
;*****
;*****

;          24/16 Bit Signed Fixed Point Divide 24/16 -> 24.16
;          Input:  24 bit fixed point dividend in AARGB0, AARGB1, AARGB2
;          16 bit fixed point divisor in BARGB0, BARGB1
;          Use:    CALL    FXD2416S
;          Output: 24 bit fixed point quotient in AARGB0, AARGB1, AARGB2
```

```

;          16 bit fixed point remainder in REMB0, REMB1
;          Result: AARG, REM <-- AARG / BARG
;          Max Timing:    11+283+3 = 297 clks          A > 0, B > 0
;                        14+283+15 = 312 clks         A > 0, B < 0
;                        16+283+15 = 314 clks         A < 0, B > 0
;                        19+283+3 = 305 clks         A < 0, B < 0
;          Min Timing:    11+258+3 = 272 clks         A > 0, B > 0
;                        14+258+15 = 287 clks         A > 0, B < 0
;                        16+258+15 = 289 clks         A < 0, B > 0
;                        19+258+3 = 280 clks         A < 0, B < 0
;          PM: 14+327+12 = 353          DM: 8
FXD2416S    MOVFP          AARGB0,WREG
            XORWF          BARGB0,W
            MOVWF          SIGN
            CLRF           REMB0
            CLRF           REMB1,W
            BTFFS          BARGB0,MSB          ; if MSB set go & negate BARG
            GOTO           CA2416S
            COMF           BARGB1
            COMF           BARGB0
            INCF           BARGB1
            ADDWFC          BARGB0
CA2416S    BTFFS          AARGB0,MSB          ; if MSB set go & negate ACCa
            GOTO           C2416S
            COMF           AARGB2
            COMF           AARGB1
            COMF           AARGB0
            INCF           AARGB2
            ADDWFC          AARGB1
            ADDWFC          AARGB0
C2416S    SDIV2416
            BTFFS          SIGN,MSB          ; negate (ACCc,ACCd)
            RETLW          0x00
            COMF           AARGB2
            COMF           AARGB1
            COMF           AARGB0
            CLRF           WREG
            INCF           AARGB2
            ADDWFC          AARGB1
            ADDWFC          AARGB0
            COMF           REMB1
            COMF           REMB0
            INCF           REMB1
            ADDWFC          REMB0
            RETLW          0x00
;*****
;*****

;          24/16 Bit Unsigned Fixed Point Divide 24/16 -> 24.16
;          Input:  24 bit unsigned fixed point dividend in AARGB0, AARGB1, AARGB2
;                  16 bit unsigned fixed point divisor in BARGB0, BARGB1
;          Use:    CALL    FXD2416U
;          Output: 24 bit unsigned fixed point quotient in AARGB0, AARGB1, AARGB2
;                  16 bit unsigned fixed point remainder in REMB0, REMB1
;          Result: AARG, REM <-- AARG / BARG
;          Max Timing:    2+361+2 = 365 clks
;          Min Timing:    2+335+2 = 339 clks
;          PM: 2+450+1 = 453          DM: 8
FXD2416U    CLRF           REMB0
            CLRF           REMB1
            NDIV2416
            RETLW          0x00
;*****
;*****

;          24/15 Bit Unsigned Fixed Point Divide 24/15 -> 24.15

```

AN617

```
;      Input:  24 bit unsigned fixed point dividend in AARGB0, AARGB1, AARGB2
;      15 bit unsigned fixed point divisor in BARGB0, BARGB1
;      Use:    CALL    FXD2415U
;      Output: 24 bit unsigned fixed point quotient in AARGB0, AARGB1, AARGB2
;      15 bit unsigned fixed point remainder in REMB0, REMB1
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:    2+290+2 = 294 clks
;      Min Timing:    2+264+2 = 268 clks
;      PM: 2+336+1 = 339          DM: 8
FXD2415U      CLRF          REMB0
              CLRF          REMB1
              UDIV2415
              RETLW         0x00
;*****
;*****

;      23/15 Bit Unsigned Fixed Point Divide 23/15 -> 23.15
;      Input:  23 bit unsigned fixed point dividend in AARGB0, AARGB1, AARGB2
;      15 bit unsigned fixed point divisor in BARGB0, BARGB1
;      Use:    CALL    FXD2315U
;      Output: 23 bit unsigned fixed point quotient in AARGB0, AARGB1, AARGB2
;      15 bit unsigned fixed point remainder in REMB0, REMB1
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:    2+283+2 = 287 clks
;      Min Timing:    2+258+2 = 262 clks
;      PM: 2+327+1 = 330          DM: 8
FXD2315U      CLRF          REMB0
              CLRF          REMB1
              UDIV2315
              RETLW         0x00
;*****
;*****

;      16/16 Bit Signed Fixed Point Divide 16/16 -> 16.16
;      Input:  16 bit fixed point dividend in AARGB0, AARGB1
;      16 bit fixed point divisor in BARGB0, BARGB1
;      Use:    CALL    FXD1616S
;      Output: 16 bit fixed point quotient in AARGB0, AARGB1
;      16 bit fixed point remainder in REMB0, REMB1
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:    11+187+3 = 201 clks          A > 0, B > 0
;      14+187+13 = 214 clks          A > 0, B < 0
;      14+187+13 = 214 clks          A < 0, B > 0
;      17+187+3 = 207 clks          A < 0, B < 0
;      Min Timing:    11+173+3 = 187 clks          A > 0, B > 0
;      14+173+13 = 200 clks          A > 0, B < 0
;      14+173+13 = 200 clks          A < 0, B > 0
;      17+173+3 = 193 clks          A < 0, B < 0
;      PM: 14+215+12 = 241          DM: 7
FXD1616S      MOVFP        AARGB0,WREG
              XORWF        BARGB0,W
              MOVWF        SIGN
              CLRF         REMB0
              CLRF         REMB1,W
              BTFSS        BARGB0,MSB          ; if MSB set go & negate BARG
              GOTO         CA1616S
              COMF         BARGB1
              COMF         BARGB0
              INCF         BARGB1
              ADDWFC        BARGB0
CA1616S      BTFSS        AARGB0,MSB          ; if MSB set go & negate ACCa
              GOTO         C1616S
              COMF         AARGB1
              COMF         AARGB0
              INCF         AARGB1
              ADDWFC        AARGB0
```



```

C1616S          SDIV1616
                BTFFS          SIGN,MSB          ; negate (ACCc,ACCd)
                RETLW          0x00
                COMF           AARGB1
                COMF           AARGB0
                CLRF           WREG
                INCF           AARGB1
                ADDWFC         AARGB0
                COMF           REMB1
                COMF           REMB0
                INCF           REMB1
                ADDWFC         REMB0
                RETLW          0x00
;*****
;*****

;          16/16 Bit Unsigned Fixed Point Divide 16/16 -> 16.16
;          Input:  16 bit unsigned fixed point dividend in AARGB0, AARGB1
;                  16 bit unsigned fixed point divisor in BARGB0, BARGB1
;          Use:    CALL      FXD1616U
;          Output: 16 bit unsigned fixed point quotient in AARGB0, AARGB1
;                  16 bit unsigned fixed point remainder in REMB0, REMB1
;          Result: AARG, REM <-- AARG / BARG
;          Max Timing:  2+240+2 = 244 clks
;          Min Timing:  2+176+2 = 180 clks
;          PM: 2+240+1 = 243          DM: 6
FXD1616U        CLRF           REMB0
                CLRF           REMB1
                UDIV1616
                RETLW          0x00
;*****
;*****

;          16/15 Bit Unsigned Fixed Point Divide 16/15 -> 16.15
;          Input:  16 bit unsigned fixed point dividend in AARGB0, AARGB1
;                  15 bit unsigned fixed point divisor in BARGB0, BARGB1
;          Use:    CALL      FXD1615U
;          Output: 16 bit unsigned fixed point quotient in AARGB0, AARGB1
;                  15 bit unsigned fixed point remainder in REMB0, REMB1
;          Result: AARG, REM <-- AARG / BARG
;          Max Timing:  2+193+2 = 197 clks
;          Min Timing:  2+178+2 = 182 clks
;          PM: 2+213+1 = 216          DM: 6
FXD1615U        CLRF           REMB0
                CLRF           REMB1
                UDIV1615
                RETLW          0x00
;*****
;*****

;          15/15 Bit Unsigned Fixed Point Divide 15/15 -> 15.15
;          Input:  15 bit unsigned fixed point dividend in AARGB0, AARGB1
;                  15 bit unsigned fixed point divisor in BARGB0, BARGB1
;          Use:    CALL      FXD1515U
;          Output: 15 bit unsigned fixed point quotient in AARGB0, AARGB1
;                  15 bit unsigned fixed point remainder in REMB0, REMB1
;          Result: AARG, REM <-- AARG / BARG
;          Max Timing:  2+187+2 = 191 clks
;          Min Timing:  2+173+2 = 177 clks
;          PM: 2+215+1 = 218          DM: 6
FXD1515U        CLRF           REMB0
                CLRF           REMB1
                UDIV1515
                RETLW          0x00
;*****
;*****

```

AN617

```
;      16/8 Bit Signed Fixed Point Divide 16/08 -> 16.08
;      Input:  16 bit fixed point dividend in AARGB0, AARGB1
;              8 bit fixed point divisor in BARGB0
;      Use:    CALL    FXD1608S
;      Output: 16 bit fixed point quotient in AARGB0, AARGB1
;              8 bit fixed point remainder in REMB0
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      10+122+3 = 135 clks          A > 0, B > 0
;                      11+122+11 = 144 clks         A > 0, B < 0
;                      13+122+11 = 146 clks         A < 0, B > 0
;                      14+122+3 = 139 clks          A < 0, B < 0
;      Min Timing:      10+122+3 = 135 clks          A > 0, B > 0
;                      11+122+11 = 144 clks         A > 0, B < 0
;                      13+122+11 = 146 clks         A < 0, B > 0
;                      14+122+3 = 139 clks          A < 0, B < 0
;      PM: 14+122+10 = 146          DM: 5
FXD1608S      MOVFP      AARGB0,WREG
              XORWF     BARGB0,W
              MOVWF     SIGN
              CLRF      REMB0,W
              BTFSS     BARGB0,MSB          ; if MSB set go & negate BARG
              GOTO      CA1608S
              COMF      BARGB0
              INCF      BARGB0
CA1608S      BTFSS     AARGB0,MSB          ; if MSB set go & negate ACCa
              GOTO      C1608S
              COMF      AARGB1
              COMF      AARGB0
              INCF      AARGB1
              ADDWFC    AARGB0
C1608S      SDIV1608
              BTFSS     SIGN,MSB          ; negate (ACCC,ACCD)
              RETLW     0x00
              COMF      AARGB1
              COMF      AARGB0
              CLRF      WREG
              INCF      AARGB1
              ADDWFC    AARGB0
              COMF      REMB0
              INCF      REMB0
              RETLW     0x00
;*****
;*****

;      16/8 Bit Unsigned Fixed Point Divide 16/08 -> 16.08
;      Input:  16 bit unsigned fixed point dividend in AARGB0, AARGB1
;              8 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL    FXD1608U
;      Output: 16 bit unsigned fixed point quotient in AARGB0, AARGB1
;              8 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      1+193+2 = 196 clks
;      Min Timing:      1+153+2 = 156 clks
;      PM: 1+193+1 = 195          DM: 4
FXD1608U      CLRF      REMB0
              UDIV1608
              RETLW     0x00
;*****
;*****

;      16/7 Bit Unsigned Fixed Point Divide 16/07 -> 16.07
;      Input:  16 bit unsigned fixed point dividend in AARGB0, AARGB1
;              7 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL    FXD1607U
;      Output: 16 bit unsigned fixed point quotient in AARGB0, AARGB1
```

```

;          7 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      1+127+2 = 130 clks
;      Min Timing:      1+127+2 = 130 clks
;      PM: 1+127+1 = 129          DM: 4
FXD1607U      CLRF          REMB0
              UDIV1607
              RETLW        0x00
;*****
;*****
;      15/7 Bit Unsigned Fixed Point Divide 15/07 -> 15.07
;      Input:  15 bit unsigned fixed point dividend in AARGB0, AARGB1
;              7 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL      FXD1507U
;      Output: 15 bit unsigned fixed point quotient in AARGB0, AARGB1
;              7 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      1+122+2 = 125 clks
;      Min Timing:      1+122+2 = 125 clks
;      PM: 1+122+1 = 124          DM: 4
FXD1507U      CLRF          REMB0
              UDIV1507
              RETLW        0x00
;*****
;*****
;      8/8 Bit Signed Fixed Point Divide 08/08 -> 08.08
;      Input:  8 bit fixed point dividend in AARGB0
;              8 bit fixed point divisor in BARGB0
;      Use:    CALL      FXD0808S
;      Output: 8 bit fixed point quotient in AARGB0
;              8 bit fixed point remainder in REMB0
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing:      10+58+3 = 71 clks          A > 0, B > 0
;              11+58+8 = 77 clks          A > 0, B < 0
;              11+58+8 = 77 clks          A < 0, B > 0
;              12+58+3 = 73 clks          A < 0, B < 0
;      Min Timing:      10+58+3 = 71 clks          A > 0, B > 0
;              11+58+8 = 77 clks          A > 0, B < 0
;              11+58+8 = 77 clks          A < 0, B > 0
;              12+58+3 = 71 clks          A < 0, B < 0
;      PM: 12+58+7 = 77          DM: 4
FXD0808S      MOVFP        AARGB0,WREG
              XORWF        BARGB0,W
              MOVWF        SIGN
              CLRF         REMB0,W
              BTFSS        BARGB0,MSB
              GOTO         CA0808S
              COMF         BARGB0
              INCF         BARGB0
CA0808S      BTFSS        AARGB0,MSB
              GOTO         C0808S
              COMF         AARGB0
              INCF         AARGB0
C0808S      SDIV0808
              BTFSS        SIGN,MSB
              RETLW        0x00
              COMF         AARGB0
              INCF         AARGB0
              COMF         REMB0
              INCF         REMB0
              RETLW        0x00
;*****
;*****
;      8/8 Bit Unsigned Fixed Point Divide 08/08 -> 08.08

```

AN617

```
;      Input:  8 bit unsigned fixed point dividend in AARGB0
;      Input:  8 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL    FXD0808U
;      Output: 8 bit unsigned fixed point quotient in AARGB0
;      Output: 8 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing: 1+72+2 = 75 clks
;      Min Timing: 1+64+2 = 67 clks
;      PM: 1+72+1 = 74      DM: 3
FXD0808U      CLRF      REMB0
              UDIV0808
              RETLW     0x00
;*****
;*****

;      8/7 Bit Unsigned Fixed Point Divide 08/07 -> 08.07
;      Input:  8 bit unsigned fixed point dividend in AARGB0
;      Input:  7 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL    FXD0807U
;      Output: 8 bit unsigned fixed point quotient in AARGB0
;      Output: 7 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing: 1+63+2 = 66 clks
;      Min Timing: 1+63+2 = 66 clks
;      PM: 1+63+1 = 65      DM: 3
FXD0807U      CLRF      REMB0
              UDIV0807
              RETLW     0x00
;*****
;*****

;      7/7 Bit Unsigned Fixed Point Divide 07/07 -> 07.07
;      Input:  7 bit unsigned fixed point dividend in AARGB0
;      Input:  7 bit unsigned fixed point divisor in BARGB0
;      Use:    CALL    FXD0707U
;      Output: 7 bit unsigned fixed point quotient in AARGB0
;      Output: 7 bit unsigned fixed point remainder in REMB0
;      Result: AARG, REM <-- AARG / BARG
;      Max Timing: 1+58+2 = 61 clks
;      Min Timing: 1+58+2 = 61 clks
;      PM: 1+58+1 = 60      DM: 3
FXD0707U      CLRF      REMB0
              UDIV0707
              RETLW     0x00
;*****
;*****

END
```

F.3 PIC17CXX Fixed Point Divide Routines C

```
;      PIC17 FIXED POINT DIVIDE ROUTINES C      VERSION 1.8
;      Input:  fixed point arguments in AARG and BARG
;      Output: quotient AARG/BARG followed by remainder in REM
;      All timings are worst case cycle counts
;      It is useful to note that the additional unsigned routines requiring a non-power of two
;      argument can be called in a signed divide application where it is known that the
;      respective argument is nonnegative, thereby offering some improvement in
;      performance.
;      Routine      Clocks      Function
;      FXD3216S     414 32 bit/16 bit -> 32.16 signed fixed point divide
;      FXD3216U     485 32 bit/16 bit -> 32.16 unsigned fixed point divide
;      FXD3215U     390 32 bit/15 bit -> 32.15 unsigned fixed point divide
;      FXD3115U     383 31 bit/15 bit -> 31.15 unsigned fixed point divide
;      FXD2424S     390 24 bit/24 bit -> 24.24 signed fixed point divide
;      FXD2424U     440 24 bit/24 bit -> 24.24 unsigned fixed point divide
;      FXD2423U     369 24 bit/23 bit -> 24.23 unsigned fixed point divide
;      FXD2323U     361 23 bit/23 bit -> 23.23 unsigned fixed point divide
list      r=dec,x=on,t=off,p=17C42
```

```

include <PIC17.INC>      ; general PIC17 definitions

include <MATH17.INC>    ; PIC17 math library definitions
;*****
;*****
;
;   Test suite storage
RANDHI      equ      0x2B      ; random number senerator registers
RANDLO equ      0x2C
TESTCODE    equ      0x2D      ; integer code labeling test contained in following data
NUMTESTS    equ      0x2E      ; number of tests contained in following data
TESTCOUNT  equ      0x2F      ; counter
DATA        equ      0x30      ; beginning of test data
;*****
;*****
;
;   Test suite for fixed point divide algorithms
org          0x0021
MAIN         MOVLW          RAMSTART
            MOVPF          WREG,FSR0
MEMLOOP      CLRF          INDF0
            INCFSZ         FSR0
            GOTO          MEMLOOP
            BSF           RTCSTA,5
;           MOVPF          RTCCH,WREG
            MOVLW          0x45          ; seed for random numbers
            MOVPF          WREG,RANDLO
;           MOVPF          RTCCL,WREG
            MOVLW          0x30
            MOVPF          WREG,RANDHI
            MOVLW          0x30
            MOVPF          WREG,FSR0
            BCF           _FS1
            BSF           _FS0
;           CALL          TFXD3216
            CALL          TFXD2424
SELF        GOTO          SELF
RANDOM16     RLCF          RANDHI,W      ; random number generator
            XORWF         RANDHI,W
            RLCF          WREG
            SWAPF        RANDHI
            SWAPF        RANDLO,W
            RLNCF        WREG
            XORWF        RANDHI,W
            SWAPF        RANDHI
            ANDLW        0x01
            RLCF          RANDLO
            XORWF        RANDLO
            RLCF          RANDHI

            RETLW         0
;
;   Test suite for FXD3216
TFXD3216    MOVLW          17
            MOVPF          WREG,TESTCOUNT
            MOVPF          WREG,NUMTESTS
            MOVLW          1
            MOVPF          WREG,TESTCODE

D3216LOOP   CALL          RANDOM16
;           SWAPF        RANDHI
;           SWAPF        RANDLO
            MOVFP        RANDHI,WREG
            MOVFP        WREG,BARG0
            MOVFP        RANDLO,WREG
            MOVFP        WREG,BARG1
;           BCF          BARG0,MSB
            MOVFP        BARG0,INDF0
            MOVFP        BARG1,INDF0

```

AN617

```
CALL          RANDOM16

;            SWAPF          RANDHI
;            SWAPF          RANDLO
MOVFP         RANDHI , WREG
MOVFP         WREG , AARGB0
MOVFP         RANDLO , WREG
MOVFP         WREG , AARGB1
;            BCF           AARGB0 , MSB
CALL         RANDOM16
MOVFP         RANDHI , WREG
MOVFP         WREG , AARGB2
MOVFP         RANDLO , WREG
MOVFP         WREG , AARGB3

MOVFP         AARGB0 , INDF0
MOVFP         AARGB1 , INDF0
MOVFP         AARGB2 , INDF0
MOVFP         AARGB3 , INDF0
CALL         FXD3216U
MOVFP         AARGB0 , INDF0
MOVFP         AARGB1 , INDF0
MOVFP         AARGB2 , INDF0
MOVFP         AARGB3 , INDF0
MOVFP         REMB0 , INDF0
MOVFP         REMB1 , INDF0
DECFSZ       TESTCOUNT
GOTO         D3216LOOP
RETLW        0x00

;            Test suite for FXD2424
TFXD2424     MOVLW          17
MOVFP         WREG , TESTCOUNT
MOVFP         WREG , NUMTESTS
MOVLW        6
MOVFP         WREG , TESTCODE

D2424LOOP    CALL         RANDOM16
MOVFP         RANDHI , WREG
MOVFP         WREG , BARGB0
MOVFP         RANDLO , WREG
MOVFP         WREG , BARGB1
CALL         RANDOM16
MOVFP         RANDHI , WREG
MOVFP         WREG , BARGB2
;            BCF           BARGB0 , MSB
MOVFP         BARGB0 , INDF0
MOVFP         BARGB1 , INDF0
MOVFP         BARGB2 , INDF0
CALL         RANDOM16
MOVFP         RANDHI , WREG
MOVFP         WREG , AARGB0
MOVFP         RANDLO , WREG
MOVFP         WREG , AARGB1
CALL         RANDOM16
MOVFP         RANDHI , WREG
MOVFP         WREG , AARGB2

;            BCF           AARGB0 , MSB
MOVFP         AARGB0 , INDF0
MOVFP         AARGB1 , INDF0
MOVFP         AARGB2 , INDF0
CALL         FXD2424S
MOVFP         AARGB0 , INDF0
MOVFP         AARGB1 , INDF0
MOVFP         AARGB2 , INDF0
MOVFP         REMB0 , INDF0
```

```

MOVFP          REMB1, INDF0
MOVFP          REMB2, INDF0
DECFSZ        TESTCOUNT
GOTO          D2424LOOP
RETLW         0x00
;*****
;*****
;
; 32/16 Bit Division Macros
SDIV3216      macro
; Max Timing:   5+8+30*12+6 = 379 clks
; Min Timing:   5+8+30*11+6 = 349 clks
; PM: 5+8+30*14+6 = 439          DM: 8
    variable i
    MOVFP      BARGB1, WREG
    SUBWF      REMB1
    MOVFP      BARGB0, WREG
    SUBWFB     REMB0
    RLCF      ACCB0
    RLCF      ACCB0, W
    RLCF      REMB1
    RLCF      REMB0
    MOVFP      BARGB1, WREG
    ADDWF      REMB1
    MOVFP      BARGB0, WREG
    ADDWFC     REMB0
    RLCF      ACCB0
    i = 2
    while i < 8
    RLCF      ACCB0, W
    RLCF      REMB1
    RLCF      REMB0
    MOVFP      BARGB1, WREG
    BTFSS     ACCB0, LSB
    GOTO      SADD26#v(i)
    SUBWF      REMB1
    MOVFP      BARGB0, WREG
    SUBWFB     REMB0
SADD26#v(i)   ADDWF      REMB1
    MOVFP      BARGB0, WREG
    ADDWFC     REMB0
SOK26#v(i)   RLCF      ACCB0
    i=i+1
    endw
    RLCF      ACCB1, W
    RLCF      REMB1
    RLCF      REMB0
    MOVFP      BARGB1, WREG
    BTFSS     ACCB0, LSB
    GOTO      SADD268
    SUBWF      REMB1
    MOVFP      BARGB0, WREG
    SUBWFB     REMB0
    GOTO      SOK268
SADD268      ADDWF      REMB1
    MOVFP      BARGB0, WREG
    ADDWFC     REMB0
SOK268      RLCF      ACCB1
    i = 9
    while i < 16
    RLCF      ACCB1, W
    RLCF      REMB1
    RLCF      REMB0
    MOVFP      BARGB1, WREG
    BTFSS     ACCB1, LSB
    GOTO      SADD26#v(i)

```

AN617

```

SUBWF          REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO          SOK26#v(i)
SADD26#v(i)   ADDWF          REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
SOK26#v(i)   RLCF          ACCB1
              i=i+1
              endw
              RLCF          ACCB2,W
              RLCF          REMB1
              RLCF          REMB0
              MOVFP        BARGB1,WREG
              BTFSS        ACCB1,LSB
              GOTO          SADD2616
              SUBWF        REMB1
              MOVFP        BARGB0,WREG
              SUBWFB       REMB0
SADD2616     GOTO          SOK2616
              ADDWF        REMB1
              MOVFP        BARGB0,WREG
              ADDWFC       REMB0
SOK2616     RLCF          ACCB2
              i = 17
              while i < 24
              RLCF          ACCB2,W
              RLCF          REMB1
              RLCF          REMB0
              MOVFP        BARGB1,WREG
              BTFSS        ACCB2,LSB
              GOTO          SADD26#v(i)
              SUBWF        REMB1
              MOVFP        BARGB0,WREG
              SUBWFB       REMB0
              GOTO          SOK26#v(i)
SADD26#v(i)  ADDWF          REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
SOK26#v(i)  RLCF          ACCB2
              i=i+1
              endw
              RLCF          ACCB3,W
              RLCF          REMB1
              RLCF          REMB0
              MOVFP        BARGB1,WREG
              BTFSS        ACCB2,LSB
              GOTO          SADD2624
              SUBWF        REMB1
              MOVFP        BARGB0,WREG
              SUBWFB       REMB0
              GOTO          SOK2624
SADD2624    ADDWF          REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
SOK2624    RLCF          ACCB3
              i = 25
              while i < 32
              RLCF          ACCB3,W
              RLCF          REMB1
              RLCF          REMB0
              MOVFP        BARGB1,WREG
              BTFSS        ACCB3,LSB
              GOTO          SADD26#v(i)
              SUBWF        REMB1
              MOVFP        BARGB0,WREG
```



```

SUBWFB          REMB0
SADD26#v(i)    GOTO          SOK26#v(i)
               ADDWF         REMB1
               MOVFP        BARGB0,WREG
               ADDWFC       REMB0
SOK26#v(i)    RLCF          ACCB3
               i=i+1
               endw
               BTFSC        ACCB3,LSB
               GOTO          SOK26
               MOVFP        BARGB1,WREG
               ADDWF         REMB1
               MOVFP        BARGB0,WREG
               ADDWFC       REMB0
SOK26
               endm
UDIV3216      macro
;               restore = 15/20 clks, nonrestore = 11/14 clks
;               Max Timing: 16*15+1+16*20 = 561 clks
;               Min Timing: 16*11+1+16*14 = 401 clks
;               PM: 16*15+1+16*20 = 561          DM: 9
               variable      i
               i = 0
               while i < 8
RLCF          ACCB0,W
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB1,WREG
SUBWF        REMB1
MOVFP        BARGB0,WREG
SUBWFB       REMB0
BTFSC        _C
GOTO          UOK26#v(i)
MOVFP        BARGB1,WREG
ADDWF        REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
BCF          _C
UOK26#v(i)  RLCF          ACCB0
               i=i+1
               endw
               i = 8
               while i < 16
RLCF          ACCB1,W
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB1,WREG
SUBWF        REMB1
MOVFP        BARGB0,WREG
SUBWFB       REMB0
BTFSC        _C
GOTO          UOK26#v(i)
MOVFP        BARGB1,WREG
ADDWF        REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
BCF          _C
UOK26#v(i)  RLCF          ACCB1
               i=i+1
               endw
               CLR          TEMP
               i = 16
               while i < 24
RLCF          ACCB2,W
RLCF          REMB1
RLCF          REMB0

```

AN617

```

        RLCF          TEMP
        MOVFP        BARGB1,WREG
        SUBWF        REMB1
        MOVFP        BARGB0,WREG
        SUBWFB       REMB0
        CLRF         WREG
        SUBWFB       TEMP
        BTFSC        _C
        GOTO         UOK26#v(i)
        MOVFP        BARGB1,WREG
        ADDWF        REMB1
        MOVFP        BARGB0,WREG
        ADDWFC       REMB0
        CLRF         WREG
        ADDWFC       TEMP
        BCF          _C
UOK26#v(i) RLCF          ACCB2
           i=i+1
           endw
           i = 24
           while i < 32
RLCF          ACCB3,W
RLCF          REMB1
RLCF          REMB0
RLCF          TEMP
MOVFP        BARGB1,WREG
SUBWF        REMB1
MOVFP        BARGB0,WREG
SUBWFB       REMB0
CLRF         WREG
SUBWFB       TEMP
BTFSC        _C
GOTO         UOK26#v(i)
MOVFP        BARGB1,WREG
ADDWF        REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
CLRF         WREG
ADDWFC       TEMP
        BCF          _C
UOK26#v(i) RLCF          ACCB3
           i=i+1
           endw
           endm
NDIV3216 macro
;           Max Timing:      10+31*15+6 = 481 clks
;           Min Timing: 10+31*14+6 = 450 clks
;           PM: 10+31*19+6 = 605           DM: 9
           variable i
           RLCF          ACCB0,W
           RLCF          REMB1
           RLCF          REMB0
           MOVFP        BARGB1,WREG
           SUBWF        REMB1
           MOVFP        BARGB0,WREG
           SUBWFB       REMB0
           CLRF         TEMP,W
           SUBWFB       TEMP
           RLCF          ACCB0
           i = 1
           while i < 8
           RLCF          ACCB0,W
           RLCF          REMB1
           RLCF          REMB0
           RLCF          TEMP
           MOVFP        BARGB1,WREG
```

	BTFSS	ACCB0, LSB
	GOTO	NADD26#v(i)
	SUBWF	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	CLRF	WREG
	SUBWFB	TEMP
NADD26#v(i)	GOTO	NOK26#v(i)
	ADDWF	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
	CLRF	WREG
	ADDWFC	TEMP
NOK26#v(i)	RLCF	ACCB0
	i=i+1	
	endw	
	RLCF	ACCB1, W
	RLCF	REMB1
	RLCF	REMB0
	RLCF	TEMP
	MOVFP	BARGB1, WREG
	BTFSS	ACCB0, LSB
	GOTO	NADD268
	SUBWF	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	CLRF	WREG
	SUBWFB	TEMP
NADD268	GOTO	NOK268
	ADDWF	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
	CLRF	WREG
	ADDWFC	TEMP
NOK268	RLCF	ACCB1
	i = 9	
	while i < 16	
	RLCF	ACCB1, W
	RLCF	REMB1
	RLCF	REMB0
	RLCF	TEMP
	MOVFP	BARGB1, WREG
	BTFSS	ACCB1, LSB
	GOTO	NADD26#v(i)
	SUBWF	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	CLRF	WREG
	SUBWFB	TEMP
NADD26#v(i)	GOTO	NOK26#v(i)
	ADDWF	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
	CLRF	WREG
	ADDWFC	TEMP
NOK26#v(i)	RLCF	ACCB1
	i=i+1	
	endw	
	RLCF	ACCB2, W
	RLCF	REMB1
	RLCF	REMB0
	RLCF	TEMP
	MOVFP	BARGB1, WREG

AN617

```

                                BTFSS      ACCB1, LSB
                                GOTO      NADD2616
                                SUBWF     REMB1
                                MOVFP    BARGB0, WREG
                                SUBWFB   REMB0
                                CLRF     WREG
                                SUBWFB   TEMP
                                GOTO     NOK2616
NADD2616                        ADDWF     REMB1
                                MOVFP    BARGB0, WREG
                                ADDWFC   REMB0
                                CLRF     WREG
                                ADDWFC   TEMP

NOK2616                        RLCF      ACCB2
                                i = 17
                                while i < 24
                                RLCF     ACCB2, W
                                RLCF     REMB1
                                RLCF     REMB0
                                RLCF     TEMP
                                MOVFP    BARGB1, WREG
                                BTFSS    ACCB2, LSB
                                GOTO     NADD26#v(i)
                                SUBWF    REMB1
                                MOVFP    BARGB0, WREG
                                SUBWFB   REMB0
                                CLRF     WREG
                                SUBWFB   TEMP
                                GOTO     NOK26#v(i)
NADD26#v(i)                    ADDWF     REMB1
                                MOVFP    BARGB0, WREG
                                ADDWFC   REMB0
                                CLRF     WREG
                                ADDWFC   TEMP

NOK26#v(i)                     RLCF      ACCB2
                                i=i+1
                                endw
                                RLCF     ACCB3, W
                                RLCF     REMB1
                                RLCF     REMB0
                                RLCF     TEMP
                                MOVFP    BARGB1, WREG
                                BTFSS    ACCB2, LSB
                                GOTO     NADD2624
                                SUBWF    REMB1
                                MOVFP    BARGB0, WREG
                                SUBWFB   REMB0
                                CLRF     WREG
                                SUBWFB   TEMP
                                GOTO     NOK2624
NADD2624                        ADDWF     REMB1
                                MOVFP    BARGB0, WREG
                                ADDWFC   REMB0
                                CLRF     WREG
                                ADDWFC   TEMP

NOK2624                        RLCF      ACCB3
                                i = 25
                                while i < 32
                                RLCF     ACCB3, W
                                RLCF     REMB1
                                RLCF     REMB0
                                RLCF     TEMP
                                MOVFP    BARGB1, WREG
```

```

        BTFSS          ACCB3,LSB
        GOTO          NADD26#v(i)
        SUBWF         REMB1
        MOVFP         BARGB0,WREG
        SUBWFB        REMB0
        CLRF          WREG
        SUBWFB        TEMP
        GOTO          NOK26#v(i)
NADD26#v(i)  ADDWF         REMB1
             MOVFP         BARGB0,WREG
             ADDWFC        REMB0
             CLRF          WREG
             ADDWFC        TEMP

NOK26#v(i)   RLCF          ACCB3
             i=i+1
             endw
             BTFSC        ACCB3,LSB
             GOTO          NOK26
             MOVFP         BARGB1,WREG
             ADDWF         REMB1
             MOVFP         BARGB0,WREG
             ADDWFC        REMB0

NOK26
             endm
UDIV3215    macro
;           Max Timing:      8+31*12+6 = 386 clks
;           Min Timing:      8+31*11+6 = 355 clks
;           PM: 8+31*14+6 = 448           DM: 8
             variable i
             RLCF          ACCB0,W
             RLCF          REMB1
             RLCF          REMB0
             MOVFP         BARGB1,WREG
             SUBWF         REMB1
             MOVFP         BARGB0,WREG
             SUBWFB        REMB0
             RLCF          ACCB0
             i = 1
             while i < 8
             RLCF          ACCB0,W
             RLCF          REMB1
             RLCF          REMB0
             MOVFP         BARGB1,WREG
             BTFSS        ACCB0,LSB
             GOTO          UADD25#v(i)
             SUBWF         REMB1
             MOVFP         BARGB0,WREG
             SUBWFB        REMB0
             GOTO          UOK25#v(i)
UADD25#v(i)  ADDWF         REMB1
             MOVFP         BARGB0,WREG
             ADDWFC        REMB0

UOK25#v(i)   RLCF          ACCB0
             i=i+1
             endw
             RLCF          ACCB1,W
             RLCF          REMB1
             RLCF          REMB0
             MOVFP         BARGB1,WREG
             BTFSS        ACCB0,LSB
             GOTO          UADD258
             SUBWF         REMB1
             MOVFP         BARGB0,WREG
             SUBWFB        REMB0
             GOTO          UOK258

```

AN617

```
UADD258      ADDWF      REMB1
              MOVFP      BARGB0 , WREG
              ADDWFC
UOK258       RLCF      REMB0
              RLCF      ACCB1
              i = 9
              while i < 16
              RLCF      ACCB1 , W
              RLCF      REMB1
              RLCF      REMB0
              MOVFP      BARGB1 , WREG
              BTFSS     ACCB1 , LSB
              GOTO      UADD25#v(i)
              SUBWF     REMB1
              MOVFP      BARGB0 , WREG
              SUBWFB    REMB0
              GOTO      UOK25#v(i)
UADD25#v(i)  ADDWF      REMB1
              MOVFP      BARGB0 , WREG
              ADDWFC    REMB0
UOK25#v(i)   RLCF      ACCB1
              i=i+1
              endw
              RLCF      ACCB2 , W
              RLCF      REMB1
              RLCF      REMB0
              MOVFP      BARGB1 , WREG
              BTFSS     ACCB1 , LSB
              GOTO      UADD2516
              SUBWF     REMB1
              MOVFP      BARGB0 , WREG
              SUBWFB    REMB0
              GOTO      UOK2516
UADD2516     ADDWF      REMB1
              MOVFP      BARGB0 , WREG
              ADDWFC    REMB0
UOK2516      RLCF      ACCB2
              i = 17
              while i < 24
              RLCF      ACCB2 , W
              RLCF      REMB1
              RLCF      REMB0
              MOVFP      BARGB1 , WREG
              BTFSS     ACCB2 , LSB
              GOTO      UADD25#v(i)
              SUBWF     REMB1
              MOVFP      BARGB0 , WREG
              SUBWFB    REMB0
              GOTO      UOK25#v(i)
UADD25#v(i)  ADDWF      REMB1
              MOVFP      BARGB0 , WREG
              ADDWFC    REMB0
UOK25#v(i)   RLCF      ACCB2
              i=i+1
              endw
              RLCF      ACCB3 , W
              RLCF      REMB1
              RLCF      REMB0
              MOVFP      BARGB1 , WREG
              BTFSS     ACCB2 , LSB
              GOTO      UADD2524
              SUBWF     REMB1
              MOVFP      BARGB0 , WREG
              SUBWFB    REMB0
              GOTO      UOK2524
UADD2524     ADDWF      REMB1
              MOVFP      BARGB0 , WREG
```

```

UOK2524      ADDWFC      REMB0
              RLCF       ACCB3
              i = 25
              while i < 32
                RLCF      ACCB3,W
                RLCF      REMB1
                RLCF      REMB0
                MOVFP     BARGB1,WREG
                BTFSS     ACCB3,LSB
                GOTO      UADD25#v(i)
                SUBWF     REMB1
                MOVFP     BARGB0,WREG
                SUBWFB    REMB0
              GOTO      UOK25#v(i)
UADD25#v(i)  ADDWF      REMB1
              MOVFP     BARGB0,WREG
              ADDWFC     REMB0
UOK25#v(i)   RLCF       ACCB3
              i=i+1
              endw
              BTFSC     ACCB3,LSB
              GOTO      UOK25
              MOVFP     BARGB1,WREG
              ADDWF     REMB1
              MOVFP     BARGB0,WREG
              ADDWFC     REMB0
UOK25
              endm
UDIV3115     macro
;           Max Timing:      5+8+30*12+6 = 379 clks
;           Min Timing:      5+8+30*11+6 = 349 clks
;           PM: 5+8+30*14+6 = 439           DM: 8
              variable i
              MOVFP     BARGB1,WREG
              SUBWF     REMB1
              MOVFP     BARGB0,WREG
              SUBWFB    REMB0
              RLCF      ACCB0
              RLCF      ACCB0,W
              RLCF      REMB1
              RLCF      REMB0
              MOVFP     BARGB1,WREG
              ADDWF     REMB1
              MOVFP     BARGB0,WREG
              ADDWFC     REMB0
              RLCF      ACCB0
              i = 2
              while i < 8
                RLCF      ACCB0,W
                RLCF      REMB1
                RLCF      REMB0
                MOVFP     BARGB1,WREG
                BTFSS     ACCB0,LSB
                GOTO      UADD15#v(i)
                SUBWF     REMB1
                MOVFP     BARGB0,WREG
                SUBWFB    REMB0
              GOTO      UOK15#v(i)
UADD15#v(i)  ADDWF      REMB1
              MOVFP     BARGB0,WREG
              ADDWFC     REMB0
UOK15#v(i)   RLCF       ACCB0
              i=i+1
              endw
              RLCF      ACCB1,W
              RLCF      REMB1

```

AN617

	RLCF	REMB0
	MOVFP	BARGB1, WREG
	BTFSS	ACCB0, LSB
	GOTO	UADD158
	SUBWF	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	UOK158
UADD158	ADDWF	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
UOK158	RLCF	ACCB1
	i = 9	
	while i < 16	
	RLCF	ACCB1, W
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB1, WREG
	BTFSS	ACCB1, LSB
	GOTO	UADD15#v(i)
	SUBWF	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	UOK15#v(i)
UADD15#v(i)	ADDWF	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
UOK15#v(i)	RLCF	ACCB1
	i=i+1	
	endw	
	RLCF	ACCB2, W
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB1, WREG
	BTFSS	ACCB1, LSB
	GOTO	UADD1516
	SUBWF	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	UOK1516
UADD1516	ADDWF	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
UOK1516	RLCF	ACCB2
	i = 17	
	while i < 24	
	RLCF	ACCB2, W
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB1, WREG
	BTFSS	ACCB2, LSB
	GOTO	UADD15#v(i)
	SUBWF	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	UOK15#v(i)
UADD15#v(i)	ADDWF	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
UOK15#v(i)	RLCF	ACCB2
	i=i+1	
	endw	
	RLCF	ACCB3, W
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB1, WREG


```

                BTFSS          ACCB2,LSB
                GOTO          UADD1524
                SUBWF         REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                GOTO          UOK1524
UADD1524      ADDWF          REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
UOK1524      RLCF          ACCB3
                i = 25
                while i < 32
                RLCF          ACCB3,W
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB1,WREG
                BTFSS        ACCB3,LSB
                GOTO          UADD15#v(i)
                SUBWF         REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                GOTO          UOK15#v(i)
UADD15#v(i)  ADDWF          REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
UOK15#v(i)  RLCF          ACCB3
                i=i+1
                endw
                BTFSC        ACCB3,LSB
                GOTO          UOK15
                MOVFP         BARGB1,WREG
                ADDWF         REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
UOK15
                endm
;*****
;*****
;          24/24 Bit Division Macros
SDIV2424    macro
;          Max Timing:      7+11+22*15+8 = 356 clks
;          Min Timing:      7+11+22*14+3 = 329 clks
;          PM: 7+11+22*19+8 = 444          DM: 9
                variable i
                MOVFP         BARGB2,WREG
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                RLCF          ACCB0
                RLCF          ACCB0,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                ADDWF         REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
                RLCF          ACCB0
                i = 2
                while i < 8
                RLCF          ACCB0,W
                RLCF          REMB2

```

AN617

	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB2, WREG
	BTFSS	ACCB0, LSB
	GOTO	SADD44#v(i)
	SUBWF	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	SOK44#v(i)
SADD44#v(i)	ADDWF	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
SOK44#v(i)	RLCF	ACCB0
	i=i+1	
	endw	
	RLCF	ACCB1, W
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB2, WREG
	BTFSS	ACCB0, LSB
	GOTO	SADD448
	SUBWF	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	SOK448
SADD448	ADDWF	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
SOK448	RLCF	ACCB1
	i = 9	
	while i < 16	
	RLCF	ACCB1, W
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB2, WREG
	BTFSS	ACCB1, LSB
	GOTO	SADD44#v(i)
	SUBWF	REMB2
	MOVFP	BARGB1, WREG
	SUBWFB	REMB1
	MOVFP	BARGB0, WREG
	SUBWFB	REMB0
	GOTO	SOK44#v(i)
SADD44#v(i)	ADDWF	REMB2
	MOVFP	BARGB1, WREG
	ADDWFC	REMB1
	MOVFP	BARGB0, WREG
	ADDWFC	REMB0
SOK44#v(i)	RLCF	ACCB1
	i=i+1	
	endw	
	RLCF	ACCB2, W
	RLCF	REMB2
	RLCF	REMB1
	RLCF	REMB0
	MOVFP	BARGB2, WREG

```

                BTFSS          ACCB1,LSB
                GOTO          SADD4416
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                GOTO          SOK4416
SADD4416      ADDWF          REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
SOK4416      RLCF           ACCB2
                i = 17
                while i < 24
                RLCF          ACCB2,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                BTFSS        ACCB2,LSB
                GOTO          SADD44#v(i)
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                GOTO          SOK44#v(i)
SADD44#v(i)  ADDWF          REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
SOK44#v(i)  RLCF           ACCB2
                i=i+1
                endw
                BTFSC        ACCB2,LSB
                GOTO          SOK44
                MOVFP         BARGB2,WREG
                ADDWF         REMB2
                MOVFP         BARGB1,WREG
                ADDWFC        REMB1
                MOVFP         BARGB0,WREG
                ADDWFC        REMB0
SOK44      endm
UDIV2424    macro
;           restore = 20/25 clks, nonrestore = 14/17 clks
;           Max Timing: 16*20+1+8*25 = 521 clks
;           Min Timing: 16*14+1+8*17 = 361 clks
;           PM: 16*20+1+8*25 = 521          DM: 10
                variable      i
                i = 0
                while i < 8
                RLCF          ACCB0,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                BTFSC        _C

```

AN617

```

GOTO          UOK44#v(i)
MOVFP        BARGB2,WREG
ADDWF        REMB2
MOVFP        BARGB1,WREG
ADDWFC       REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
BCF          _C
UOK44#v(i)   RLCF          ACCB0
             i=i+1
             endw
             i = 8
             while i < 16
RLCF          ACCB1,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
MOVFP        BARGB2,WREG
SUBWF        REMB2
MOVFP        BARGB1,WREG
SUBWFB       REMB1
MOVFP        BARGB0,WREG
SUBWFB       REMB0
BTFSC       _C
GOTO          UOK44#v(i)
MOVFP        BARGB2,WREG
ADDWF        REMB2
MOVFP        BARGB1,WREG
ADDWFC       REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
BCF          _C
UOK44#v(i)   RLCF          ACCB1
             i=i+1
             endw
CLRF         TEMP
             i = 16
             while i < 24
RLCF          ACCB2,W
RLCF          REMB2
RLCF          REMB1
RLCF          REMB0
RLCF          TEMP
MOVFP        BARGB2,WREG
SUBWF        REMB2
MOVFP        BARGB1,WREG
SUBWFB       REMB1
MOVFP        BARGB0,WREG
SUBWFB       REMB0
CLRF         WREG
SUBWFB       TEMP
BTFSC       _C
GOTO          UOK44#v(i)
MOVFP        BARGB2,WREG
ADDWF        REMB2
MOVFP        BARGB1,WREG
ADDWFC       REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
CLRF         WREG
ADDWFC       TEMP
BCF          _C
UOK44#v(i)   RLCF          ACCB2
             i=i+1
             endw
             endm
```

```

NDIV2424      macro
;      Max Timing:      13+23*18+8 = 435 clks
;      Min Timing:      13+23*17+3 = 407 clks
;      PM: 13+23*24+8 = 573          DM: 10
      variable i
      RLCF          ACCB0,W
      RLCF          REMB2
      RLCF          REMB1
      RLCF          REMB0
      MOVFP         BARGB2,WREG
      SUBWF         REMB2
      MOVFP         BARGB1,WREG
      SUBWFB        REMB1
      MOVFP         BARGB0,WREG
      SUBWFB        REMB0
      CLRF          TEMP,W
      SUBWFB        TEMP
      RLCF          ACCB0
      i = 1
      while i < 8
      RLCF          ACCB0,W
      RLCF          REMB2
      RLCF          REMB1
      RLCF          REMB0
      RLCF          TEMP
      MOVFP         BARGB2,WREG
      BTFSS         ACCB0,LSB
      GOTO          NADD44#v(i)
      SUBWF         REMB2
      MOVFP         BARGB1,WREG
      SUBWFB        REMB1
      MOVFP         BARGB0,WREG
      SUBWFB        REMB0
      CLRF          WREG
      SUBWFB        TEMP
      GOTO          NOK44#v(i)
NADD44#v(i)    ADDWF          REMB2
      MOVFP         BARGB1,WREG
      ADDWFC        REMB1
      MOVFP         BARGB0,WREG
      ADDWFC        REMB0
      CLRF          WREG
      ADDWFC        TEMP

NOK44#v(i)    RLCF          ACCB0
      i=i+1
      endw
      RLCF          ACCB1,W
      RLCF          REMB2
      RLCF          REMB1
      RLCF          REMB0
      RLCF          TEMP
      MOVFP         BARGB2,WREG
      BTFSS         ACCB0,LSB
      GOTO          NADD448
      SUBWF         REMB2
      MOVFP         BARGB1,WREG
      SUBWFB        REMB1
      MOVFP         BARGB0,WREG
      SUBWFB        REMB0
      CLRF          WREG
      SUBWFB        TEMP
      GOTO          NOK448
NADD448      ADDWF          REMB2
      MOVFP         BARGB1,WREG
      ADDWFC        REMB1

```

AN617

```
MOVFP          BARGB0,WREG
ADDWFC         REMB0
CLRF           WREG
ADDWFC         TEMP

NOK448         RLCF          ACCB1
               i = 9
               while i < 16
RLCF           ACCB1,W
RLCF           REMB2
RLCF           REMB1
RLCF           REMB0
RLCF           TEMP
MOVFP         BARGB2,WREG
BTFSS        ACCB1,LSB
GOTO         NADD44#v(i)
SUBWF        REMB2
MOVFP        BARGB1,WREG
SUBWFB       REMB1
MOVFP        BARGB0,WREG
SUBWFB       REMB0
CLRF         WREG
SUBWFB       TEMP
GOTO         NOK44#v(i)
NADD44#v(i)   ADDWF         REMB2
MOVFP        BARGB1,WREG
ADDWFC       REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
CLRF         WREG
ADDWFC       TEMP

NOK44#v(i)    RLCF          ACCB1
               i=i+1
               endw
RLCF         ACCB2,W
RLCF         REMB2
RLCF         REMB1
RLCF         REMB0
RLCF         TEMP
MOVFP        BARGB2,WREG
BTFSS        ACCB1,LSB
GOTO         NADD4416
SUBWF        REMB2
MOVFP        BARGB1,WREG
SUBWFB       REMB1
MOVFP        BARGB0,WREG
SUBWFB       REMB0
CLRF         WREG
SUBWFB       TEMP
GOTO         NOK4416
NADD4416     ADDWF         REMB2
MOVFP        BARGB1,WREG
ADDWFC       REMB1
MOVFP        BARGB0,WREG
ADDWFC       REMB0
CLRF         WREG
ADDWFC       TEMP

NOK4416     RLCF          ACCB2
               i = 17
               while i < 24
RLCF         ACCB2,W
RLCF         REMB2
RLCF         REMB1
RLCF         REMB0
```

```

        RLCF          TEMP
        MOVFP        BARGB2,WREG
        BTFSS       ACCB2,LSB
        GOTO        NADD44#v(i)
        SUBWF       REMB2
        MOVFP        BARGB1,WREG
        SUBWFB      REMB1
        MOVFP        BARGB0,WREG
        SUBWFB      REMB0
        CLRF        WREG
        SUBWFB      TEMP
        GOTO        NOK44#v(i)
NADD44#v(i)  ADDWF       REMB2
        MOVFP        BARGB1,WREG
        ADDWFC      REMB1
        MOVFP        BARGB0,WREG
        ADDWFC      REMB0
        CLRF        WREG
        ADDWFC      TEMP

NOK44#v(i)  RLCF          ACCB2
        i=i+1
        endw
        BTFSC       ACCB2,LSB
        GOTO        NOK44
        MOVFP        BARGB2,WREG
        ADDWF       REMB2
        MOVFP        BARGB1,WREG
        ADDWFC      REMB1
        MOVFP        BARGB0,WREG
        ADDWFC      REMB0

NOK44
        endm
UDIV2423    macro
;           Max Timing:      11+23*15+8 = 364 clks
;           Min Timing:      11+23*14+3 = 336 clks
;           PM: 11+23*19+8 = 456
;
        variable i
        RLCF        ACCB0,W
        RLCF        REMB2
        RLCF        REMB1
        RLCF        REMB0
        MOVFP        BARGB2,WREG
        SUBWF       REMB2
        MOVFP        BARGB1,WREG
        SUBWFB      REMB1
        MOVFP        BARGB0,WREG
        SUBWFB      REMB0
        RLCF        ACCB0
        i = 1
        while i < 8
        RLCF        ACCB0,W
        RLCF        REMB2
        RLCF        REMB1
        RLCF        REMB0
        MOVFP        BARGB2,WREG
        BTFSS       ACCB0,LSB
        GOTO        UADD43#v(i)
        SUBWF       REMB2
        MOVFP        BARGB1,WREG
        SUBWFB      REMB1
        MOVFP        BARGB0,WREG
        SUBWFB      REMB0
        GOTO        UOK43#v(i)
UADD43#v(i)  ADDWF       REMB2
        MOVFP        BARGB1,WREG

```

DM: 9

AN617

```

                ADDWFC          REMB1
                MOVFP          BARGB0 , WREG
UOK43#v(i)     ADDWFC          REMB0
                RLCF          ACCB0
                i=i+1
                endw
                RLCF          ACCB1 , W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB2 , WREG
                BTFSS         ACCB0 , LSB
                GOTO          UADD438
                SUBWF         REMB2
                MOVFP          BARGB1 , WREG
                SUBWFB        REMB1
                MOVFP          BARGB0 , WREG
                SUBWFB        REMB0
UADD438       GOTO          UOK438
                ADDWF         REMB2
                MOVFP          BARGB1 , WREG
                ADDWFC        REMB1
                MOVFP          BARGB0 , WREG
                ADDWFC        REMB0
UOK438       RLCF          ACCB1
                i = 9
                while i < 16
                RLCF          ACCB1 , W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB2 , WREG
                BTFSS         ACCB1 , LSB
                GOTO          UADD43#v(i)
                SUBWF         REMB2
                MOVFP          BARGB1 , WREG
                SUBWFB        REMB1
                MOVFP          BARGB0 , WREG
                SUBWFB        REMB0
                GOTO          UOK43#v(i)
UADD43#v(i)  ADDWF         REMB2
                MOVFP          BARGB1 , WREG
                ADDWFC        REMB1
                MOVFP          BARGB0 , WREG
                ADDWFC        REMB0
UOK43#v(i)  RLCF          ACCB1
                i=i+1
                endw
                RLCF          ACCB2 , W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP          BARGB2 , WREG
                BTFSS         ACCB1 , LSB
                GOTO          UADD4316
                SUBWF         REMB2
                MOVFP          BARGB1 , WREG
                SUBWFB        REMB1
                MOVFP          BARGB0 , WREG
                SUBWFB        REMB0
                GOTO          UOK4316
UADD4316    ADDWF         REMB2
                MOVFP          BARGB1 , WREG
                ADDWFC        REMB1
                MOVFP          BARGB0 , WREG
                ADDWFC        REMB0
```



```

UOK4316      RLCF          ACCB2
              i = 17
              while i < 24
                RLCF          ACCB2,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                BTFSS         ACCB2,LSB
                GOTO          UADD43#v(i)
                SUBWF         REMB2
                MOVFP         BARGB1,WREG
                SUBWFB        REMB1
                MOVFP         BARGB0,WREG
                SUBWFB        REMB0
                GOTO          UOK43#v(i)
UADD43#v(i)  ADDWF         REMB2
              MOVFP         BARGB1,WREG
              ADDWFC        REMB1
              MOVFP         BARGB0,WREG
              ADDWFC        REMB0
UOK43#v(i)   RLCF          ACCB2
              i=i+1
              endw
              BTFSC         ACCB2,LSB
              GOTO          UOK43
              MOVFP         BARGB2,WREG
              ADDWF         REMB2
              MOVFP         BARGB1,WREG
              ADDWFC        REMB1
              MOVFP         BARGB0,WREG
              ADDWFC        REMB0

UOK43
              endm
UDIV2323    macro
;           Max Timing:      7+11+22*15+8 = 356 clks
;           Min Timing:      7+11+22*14+3 = 329 clks
;           PM: 7+11+22*19+8 = 444          DM: 9
              variable i
              MOVFP         BARGB2,WREG
              SUBWF         REMB2
              MOVFP         BARGB1,WREG
              SUBWFB        REMB1
              MOVFP         BARGB0,WREG
              SUBWFB        REMB0
              RLCF          ACCB0
              RLCF          ACCB0,W
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
              MOVFP         BARGB2,WREG
              ADDWF         REMB2
              MOVFP         BARGB1,WREG
              ADDWFC        REMB1
              MOVFP         BARGB0,WREG
              ADDWFC        REMB0
              RLCF          ACCB0
              i = 2
              while i < 8
                RLCF          ACCB0,W
                RLCF          REMB2
                RLCF          REMB1
                RLCF          REMB0
                MOVFP         BARGB2,WREG
                BTFSS         ACCB0,LSB
                GOTO          UADD33#v(i)

```

AN617

```

SUBWF          REMB2
MOVFP          BARGB1,WREG
SUBWFB        REMB1
MOVFP          BARGB0,WREG
SUBWFB        REMB0
GOTO          UOK33#v(i)
UADD33#v(i)   ADDWF          REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK33#v(i)    RLCF          ACCB0
              i=i+1
              endw
              RLCF          ACCB1,W
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
              MOVFP          BARGB2,WREG
              BTFSS         ACCB0,LSB
              GOTO          UADD338
              SUBWF          REMB2
              MOVFP          BARGB1,WREG
              SUBWFB        REMB1
              MOVFP          BARGB0,WREG
              SUBWFB        REMB0
              GOTO          UOK338
UADD338       ADDWF          REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK338        RLCF          ACCB1
              i = 9
              while i < 16
              RLCF          ACCB1,W
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
              MOVFP          BARGB2,WREG
              BTFSS         ACCB1,LSB
              GOTO          UADD33#v(i)
              SUBWF          REMB2
              MOVFP          BARGB1,WREG
              SUBWFB        REMB1
              MOVFP          BARGB0,WREG
              SUBWFB        REMB0
              GOTO          UOK33#v(i)
UADD33#v(i)   ADDWF          REMB2
MOVFP          BARGB1,WREG
ADDWFC        REMB1
MOVFP          BARGB0,WREG
ADDWFC        REMB0
UOK33#v(i)    RLCF          ACCB1
              i=i+1
              endw
              RLCF          ACCB2,W
              RLCF          REMB2
              RLCF          REMB1
              RLCF          REMB0
              MOVFP          BARGB2,WREG
              BTFSS         ACCB1,LSB
              GOTO          UADD3316
              SUBWF          REMB2
              MOVFP          BARGB1,WREG
              SUBWFB        REMB1
```

```

                MOVFP      BARGB0,WREG
                SUBWFB     REMB0
                GOTO      UOK3316
UADD3316      ADDWF      REMB2
                MOVFP      BARGB1,WREG
                ADDWFC     REMB1
                MOVFP      BARGB0,WREG
                ADDWFC     REMB0
UOK3316      RLCF      ACCB2
                i = 17
                while i < 24
                RLCF      ACCB2,W
                RLCF      REMB2
                RLCF      REMB1
                RLCF      REMB0
                MOVFP      BARGB2,WREG
                BTFSS     ACCB2,LSB
                GOTO      UADD33#v(i)
                SUBWF     REMB2
                MOVFP      BARGB1,WREG
                SUBWFB     REMB1
                MOVFP      BARGB0,WREG
                SUBWFB     REMB0
                GOTO      UOK33#v(i)
UADD33#v(i)  ADDWF      REMB2
                MOVFP      BARGB1,WREG
                ADDWFC     REMB1
                MOVFP      BARGB0,WREG
                ADDWFC     REMB0
UOK33#v(i)  RLCF      ACCB2
                i=i+1
                endw
                BTFSC     ACCB2,LSB
                GOTO      UOK33
                MOVFP      BARGB2,WREG
                ADDWF     REMB2
                MOVFP      BARGB1,WREG
                ADDWFC     REMB1
                MOVFP      BARGB0,WREG
                ADDWFC     REMB0
UOK33      endm
;*****
;*****
;
;   32/16 Bit Signed Fixed Point Divide 32/16 -> 32.16
;   Input:  32 bit signed fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;           16 bit unsigned fixed point divisor in BARGB0, BARGB1
;   Use:    CALL    FXD3216S
;   Output: 32 bit signed fixed point quotient in AARGB0, AARGB1,AARGB2,AARGB3
;           16 bit fixed point remainder in REMB0, REMB1
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:  11+379+3 = 393 clks           A > 0, B > 0
;               14+379+17 = 410 clks          A > 0, B < 0
;               18+379+17 = 414 clks          A < 0, B > 0
;               21+379+3 = 403 clks           A < 0, B < 0
;   Min Timing:  11+349+3 = 363 clks          A > 0, B > 0
;               14+349+17 = 380 clks          A > 0, B < 0
;               18+349+17 = 384 clks          A < 0, B > 0
;               21+349+3 = 373 clks           A < 0, B < 0
;   PM: 21+439+16 = 476           DM: 9
FXD3216S     MOVFP      AARGB0,WREG
                XORWF     BARGB0,W
                MOVWF     SIGN
                CLRF      REMB0
                CLRF      REMB1,W

```

AN617

```

        BTFSS          BARG0,MSB          ; if MSB set go & negate BARG
        GOTO          CA3216S
        COMF          BARGB1
        COMF          BARGB0
        INCF          BARGB1
        ADDWFC        BARGB0
CA3216S  BTFSS          AARGB0,MSB        ; if MSB set go & negate ACCa
        GOTO          C3216S
        COMF          AARGB3
        COMF          AARGB2
        COMF          AARGB1
        COMF          AARGB0
        INCF          AARGB3
        ADDWFC        AARGB2
        ADDWFC        AARGB1
        ADDWFC        AARGB0
C3216S   SDIV3216
        BTFSS          SIGN,MSB          ; negate (ACCc,ACCd)
        RETLW         0x00
        COMF          AARGB3
        COMF          AARGB2
        COMF          AARGB1
        COMF          AARGB0
        CLRF          WREG
        INCF          AARGB3
        ADDWFC        AARGB2
        ADDWFC        AARGB1
        ADDWFC        AARGB0
        COMF          REMB1
        COMF          REMB0
        INCF          REMB1
        ADDWFC        REMB0
        RETLW         0x00
;*****
;*****
;       32/16 Bit Unsigned Fixed Point Divide 32/16 -> 32.16
;       Input:  32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               16 bit unsigned fixed point divisor in BARGB0, BARGB1
;       Use:    CALL    FXD3216U
;       Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1AARGB2,AARGB3
;               16 bit unsigned fixed point remainder in REMB0, REMB1
;       Result: AARG, REM  <--  AARG / BARG
;       Max Timing:    2+481+2 = 485 clks
;       Min Timing:    2+450+2 = 459 clks
;       PM: 2+605+1 = 608          DM: 9
FXD3216U  CLRF          REMB0
          CLRF          REMB1
          NDIV3216
          RETLW         0x00
;*****
;*****
;       32/15 Bit Unsigned Fixed Point Divide 32/15 -> 32.15
;       Input:  32 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;               15 bit unsigned fixed point divisor in BARGB0, BARGB1
;       Use:    CALL    FXD3215U
;       Output: 32 bit unsigned fixed point quotient in AARGB0, AARGB1
;               15 bit unsigned fixed point remainder in REMB0, REMB1
;       Result: AARG, REM  <--  AARG / BARG
;       Max Timing:    2+386+2 = 390 clks
;       Min Timing:    2+355+2 = 359 clks
;       PM: 2+448+1 = 451          DM: 8
FXD3215U  CLRF          REMB0
          CLRF          REMB1
          UDIV3215
          RETLW         0x00
```

```

;*****
;*****
;
;   31/15 Bit Unsigned Fixed Point Divide 31/15 -> 31.15
;   Input:  31 bit unsigned fixed point dividend in AARGB0, AARGB1,AARGB2,AARGB3
;           15 bit unsigned fixed point divisor in BARGB0, BARGB1
;   Use:    CALL    FXD3115U
;   Output: 31 bit unsigned fixed point quotient in AARGB0, AARGB1
;           15 bit unsigned fixed point remainder in REMB0, REMB1
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:      2+379+2 = 383 clks
;   Min Timing:      2+349+2 = 353 clks
;   PM: 2+439+1 = 442          DM: 8
FXD3115U      CLRF          REMB0
              CLRF          REMB1
              UDIV3115
              RETLW         0x00
;*****
;*****
;
;   24/24 Bit Signed Fixed Point Divide 24/24 -> 24.24
;   Input:  24 bit signed fixed point dividend in AARGB0, AARGB1, AARGB2
;           24 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
;   Use:    CALL    FXD2424S
;   Output: 24 bit signed fixed point quotient in AARGB0, AARGB1, AARGB2
;           24 bit fixed point remainder in REMB0, REMB1, REMB2
;   Result: AARG, REM  <--  AARG / BARG
;   Max Timing:      12+356+3 = 371 clks          A > 0, B > 0
;               17+356+17 = 390 clks          A > 0, B < 0
;               17+356+17 = 390 clks          A < 0, B > 0
;               22+356+3 = 381 clks          A < 0, B < 0
;   Min Timing:      12+329+3 = 344 clks          A > 0, B > 0
;               17+329+17 = 363 clks          A > 0, B < 0
;               17+329+17 = 363 clks          A < 0, B > 0
;               22+329+3 = 354 clks          A < 0, B < 0
;   PM: 22+444+16 = 482          DM: 10
FXD2424S      MOVFP          AARGB0,WREG
              XORWF          BARGB0,W
              MOVWF          SIGN
              CLRF          REMB0
              CLRF          REMB1
              CLRF          REMB2,W
              BTFSS          BARGB0,MSB      ; if MSB set, negate BARG
              GOTO          CA2424S
              COMF          BARGB2
              COMF          BARGB1
              COMF          BARGB0
              INCF          BARGB2
              ADDWFC         BARGB1
              ADDWFC         BARGB0
CA2424S      BTFSS          AARGB0,MSB      ; if MSB set, negate AARG
              GOTO          C2424S
              COMF          AARGB2
              COMF          AARGB1
              COMF          AARGB0
              INCF          AARGB2
              ADDWFC         AARGB1
              ADDWFC         AARGB0
C2424S      SDIV2424
              BTFSS          SIGN,MSB
              RETLW         0x00
              COMF          AARGB2
              COMF          AARGB1
              COMF          AARGB0
              CLRF          WREG
              INCF          AARGB2

```

AN617

```
ADDWFC      AARGB1
ADDWFC      AARGB0
COMF        REMB2
COMF        REMB1
COMF        REMB0
INCF        REMB2
ADDWFC      REMB1
ADDWFC      REMB0
RETLW      0x00
;*****
;*****
;      24/24 Bit Unsigned Fixed Point Divide 24/24 -> 24.24
;      Input:  24 bit unsigned fixed point dividend in AARGB0, AARGB1, AARGB2
;      Use:    24 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
;      Use:    CALL    FXD2424U
;      Output: 24 bit unsigned fixed point quotient in AARGB0, AARGB1, AARGB2
;      Output: 24 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:    3+435+2 = 440 clks
;      Min Timing:    3+407+2 = 412 clks
;      PM: 3+573+1 = 577          DM: 10
FXD2424U    CLRF        REMB0
            CLRF        REMB1
            CLRF        REMB2
            NDIV2424
            RETLW      0x00
;*****
;*****
;      24/23 Bit Unsigned Fixed Point Divide 24/23 -> 24.23
;      Input:  24 bit unsigned fixed point dividend in AARGB0, AARGB1, AARGB2
;      Use:    23 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
;      Use:    CALL    FXD2423U
;      Output: 24 bit unsigned fixed point quotient in AARGB0, AARGB1, AARGB2
;      Output: 23 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:    3+364+2 = 369 clks
;      Min Timing:    3+336+2 = 341 clks
;      PM: 3+456+1 = 460          DM: 9
FXD2423U    CLRF        REMB0
            CLRF        REMB1
            CLRF        REMB2
            UDIV2423
            RETLW      0x00
;*****
;*****
;      23/23 Bit Unsigned Fixed Point Divide 23/23 -> 23.23
;      Input:  23 bit unsigned fixed point dividend in AARGB0, AARGB1, AARGB2
;      Use:    23 bit unsigned fixed point divisor in BARGB0, BARGB1, BARGB2
;      Use:    CALL    FXD2323U
;      Output: 23 bit unsigned fixed point quotient in AARGB0, AARGB1, AARGB2
;      Output: 23 bit unsigned fixed point remainder in REMB0, REMB1, REMB2
;      Result: AARG, REM  <--  AARG / BARG
;      Max Timing:    3+356+2 = 361 clks
;      Min Timing:    3+329+2 = 334 clks
;      PM: 3+444+1 = 448          DM: 9
FXD2323U    CLRF        REMB0
            CLRF        REMB1
            CLRF        REMB2
            UDIV2323
            RETLW      0x00
;*****
;*****
            END
```

SECTION 3

ASSP APPLICATION NOTES

AN599	Energy Management Control System	3-1
-------	--	-----

Energy Management Control System

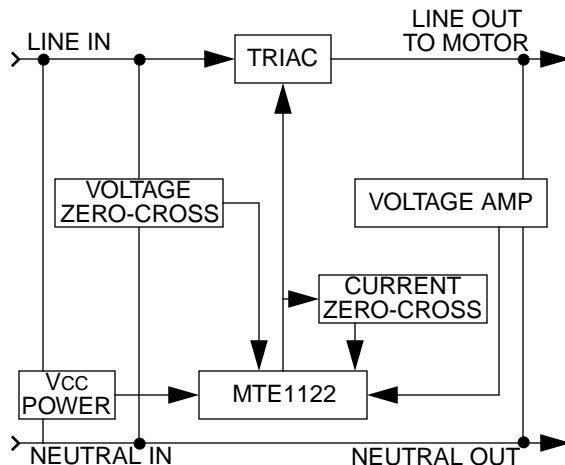
*Author: Michael Rosenfield
Memory Products Division*

INTRODUCTION

This application note describes an electronic system to improve the efficiency of certain types of single-phase induction motors.

The system is based around the MTE1122 - an energy management controller IC for induction motors. This CMOS device is based on Microchip's RISC processor core and proprietary firmware algorithms. When combined with some external analog components, the MTE1122 will provide an electronic system that economically reduces the operating costs of small induction motors by as much as 58%. It will also allow motors to run cooler and with less vibration. The system operates on single phase 110 or 240 VAC.

FIGURE 1: SYSTEM BLOCK DIAGRAM



RECOMMENDED APPLICATION

The schematic diagram shown in Figure 11 is a recommended circuit. See Table 3 for the parts list. It uses low cost, readily available components. Note that VCC is supplied directly from the AC line without the need for a transformer. Component values have been calculated to work on 110 or 240 VAC lines, with motor current draws of up to 15A RMS continuous. This translates to 1 to 1.5 HP at 110V, and 3 HP at 220V. Motor size can be increased by selecting a triac with a larger current rating.

This system will only work with rotating inductive loads (i.e., motors) that are not otherwise power-factor corrected -- capacitor-run motors will not function with this system, nor will fluorescent lighting. Universal motors (brush-type) will not benefit from the system, either. While use of this system will usually save energy, the greatest savings will be for lightly-loaded motors.

For best results, appliances or systems with other electrical devices in addition to motors should have those devices powered directly from the line, not through the energy management control system (EMC).

Also, note that each motor must have its own control circuit (unless the motors are never activated at the same time).

The circuit can be laid out on a single- or double-sided board, observing the standard layout techniques used with monolithic microcontrollers. LED D3 is lighted during normal operation of the MTE1122, and can be left out of the circuit, if desired.

Heatsinking of the triac will be required, the size depending on the triac rating, the motor current draws and ambient temperatures.

The distance between the motor and the MTE1122 circuit is not critical.

Standard electrical practices should be followed. Agency approvals may be required, depending on the implementation. While this circuit should not be connected to earth ground, any enclosure should be so connected. Also note that the low voltage power generated on this board should not be used to supply any other circuitry, particularly if that circuitry is off the board.

THEORY OF OPERATION

Power Consumption

In an induction motor, the current draw at no load is quite high because the stator windings must supply all the magnetic field energy. This means that, even when idling, the motor draws a major portion of its full-load current. The energy not converted into work is converted into heat and vibration. In addition to being wasted, the heat and vibration shortens the life of lubricants, bearings, and other components in the vicinity.

The torque produced by an AC motor is proportional to the square of the applied voltage. Thus, a motor producing part of its rated load only needs part of its rated voltage.

Power Factor

In an induction motor, the current in the windings lags the voltage, due to the inductive reactance in the windings (Figure 10). The cosine of the amount of lag in degrees is the power factor. Power factors are 1.0 for resistive loads (heaters, etc.) and vary from close to 1 for a fully loaded motor, to as low as 0.1 for an idling motor. The actual power being consumed by the motor is:

$$\text{(Voltage)} \bullet \text{(Current)} \bullet \text{(Power Factor)}$$

A lightly loaded induction motor has low power factor. As the motor reaches its rated load, its power factor gets closer to 1. How close it gets to 1 will depend on the motor's internal design. Values around 0.65 are typical of single-phase motors.

The MTE1122 calculates motor loading by measuring the time between current and voltage zero-crossings, in effect, power factor. When the load on the motor is low, the power consumed by the motor can be lowered by lowering the voltage applied to the motor, which is done by turning a triac on at the proper time during the voltage cycle. (Figure 10 for waveforms.) The resulting voltage across the motor, and the zero-crossing times, are monitored, and adjusted on a cycle-by-cycle basis, as determined by the proprietary algorithms in the MTE1122. At no load, the voltage to the motor can be as low as 85 VAC, instead of the usual 120 VAC. Power consumption can be cut by as much as 58%, depending on load, and operating temperature lowered by as much as 45°F. Refer to the system block diagram in Figure 1, and the graph in Figure 2 and Table 1.

A motor powered by the MTE1122 and this energy management control circuit will draw less current. Its power factor will also be improved; however, the power factor seen by the line will NOT be improved.

ENERGY MEASUREMENT

To measure the true power of an induction motor requires the use of a true-RMS power meter, one that will measure non-sinusoidal waveforms. Models of this type of instrument are available from Fluke and Tektronix, among others.

Measurements for this Application Note were made using the following equipment:

Hampden Engineering Corp:

CSM-100 1/3 HP motor

DYN-100A Dynamometer

RI-100A Load bank

HPT-100 Digital Photo Tachometer

Tektronix Corp:

THM560 Scopemeter

A622 Current probe

The voltage, current, and true-RMS power supplied to the Energy Management Control System driving a 1/3 HP motor were measured with the meter and current probe. The motor RPM was measured with the photo-tach. The torque supplied by the motor was measured with the dynamometer, which also supplied the adjustable load on the motor.

Motor Power Out in Watts is calculated by:

$$P = (\tau n) / 5.18$$

Power Out in Horsepower is calculated by:

$$P = (\tau n) / 7142.72$$

τ is in Newton-meters

n is RPM

Efficiency, in percent (%), is calculated by:

$$\text{(Power Out)} / \text{(Power In)} \bullet 100$$

CIRCUIT DETAILS

The MTE1122 consists of a high-performance 8-bit microcontroller (U3) with embedded proprietary algorithms, which monitors the voltage across the motor (U1), the voltage zero-crossing (Q2) and the current zero-crossing (by monitoring the signal on Q3). By measuring the time between voltage and current zero-crossings, it calculates the amount of load on the motor.

U1 and R10-R13 form a differential amplifier with a gain of 1/48. C4 limits noise sensitivity.

C2 through C6 and components in between rectify and filter line voltage to provide Vcc.

Q1 and associated components provide power-up reset for the MTE1122.

L1 and C8 form an LC filter for the 5V power supply.

U2 is an opto-triac to trigger the power triac. Q3 is the triac, which in this circuit is rated at 15A.

D3 and R7 are used to indicate "normal operation" of the circuit, and may be left out if desired.

As stated above, it turns out that the energy consumption of a motor running only partly loaded can be lowered by decreasing the current flowing into the motor windings. This can be accomplished by lowering the voltage across the motor windings. If the voltage is not increased when the motor load increases, the internal reactance of the motor decreases, and the windings will draw too much current, and could overheat and be damaged. Because the MTE1122 is an intelligent controller, it is able to monitor motor voltage and motor load, and make corrections within 8 ms, well before there is any potential for motor damage.

See Figure 10 for circuit waveforms.

ENERGY SAVINGS

Reducing the voltage to the motor cuts its power draw. By reviewing Table 1, it can be seen that at no load, the 1/3 HP test motor is dissipating 120W, much of it as heat. With the MTE1122 managing the power load, this drops to 50W, a savings of 58%. At full load, the figures are, respectively, 428W and 406W, for a savings of 5%. The degree of savings are presented in Table 2.

This data is presented graphically in the following figures.

Actual performance figures may vary based on motor size, motor load and motor construction.

ALTERNATIVE APPROACH

There is another way to increase motor efficiency. This approach is to add another winding to the motor, and phase-shift it with capacitance. This produces what is known as a capacitor-run motor, and results in a motor with a power factor close to 1.0 regardless of its load, and considerably lowered idle power consumption. It is a more efficient motor, and produces less vibration. This approach, however, is neither cost-effective in motors less than 1 hp, nor in motors for residential use. Thus, for lowest-cost approaches, use of the MTE1122 and associated circuitry is probably the best method of improving motor efficiency.

AN599

FIGURE 2: ENERGY SAVINGS

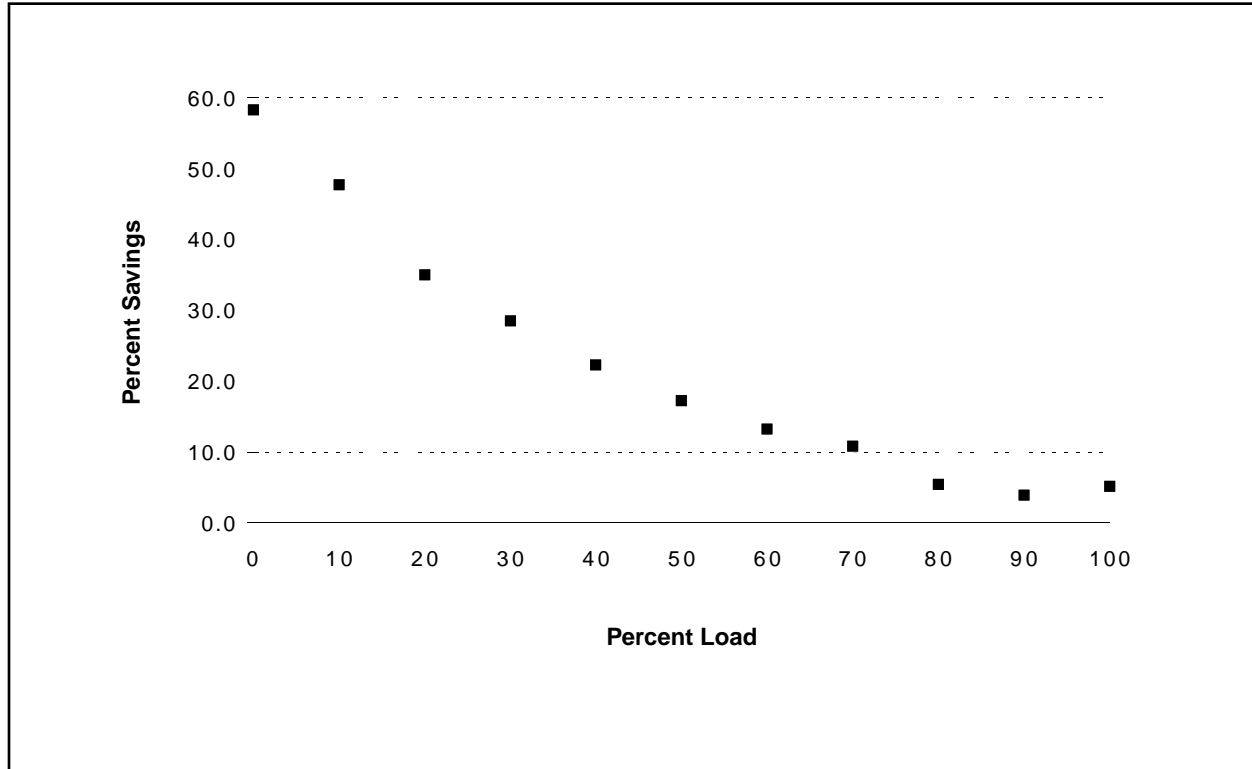


FIGURE 3: MOTOR SPEED CHANGE

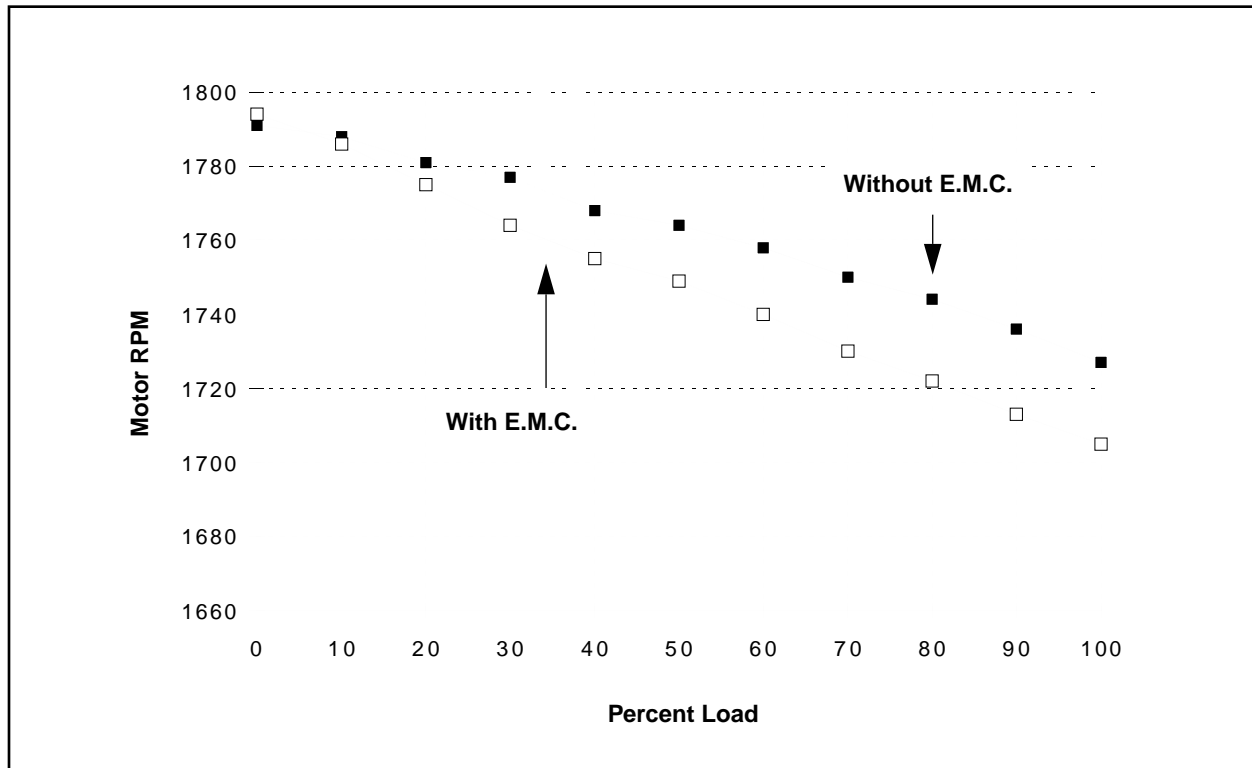


FIGURE 4: MOTOR EFFICIENCY

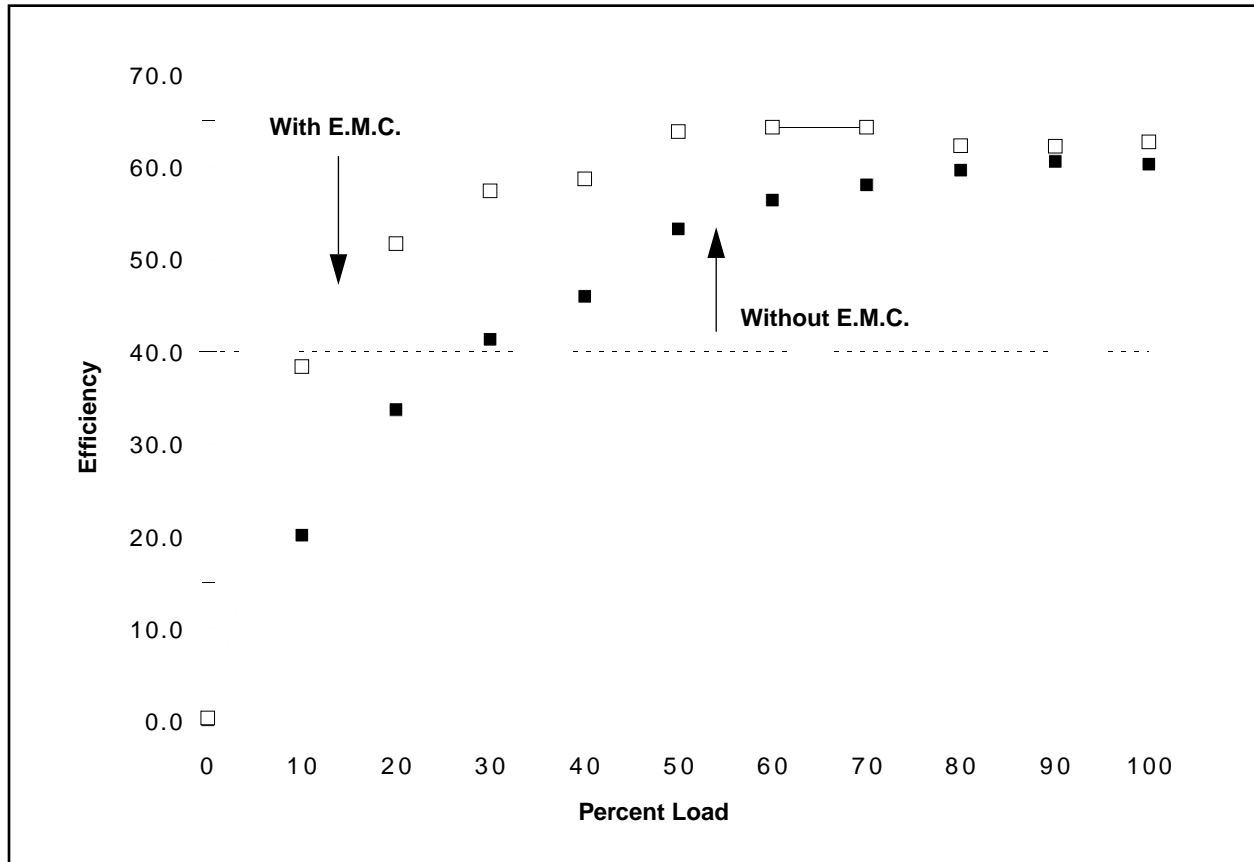
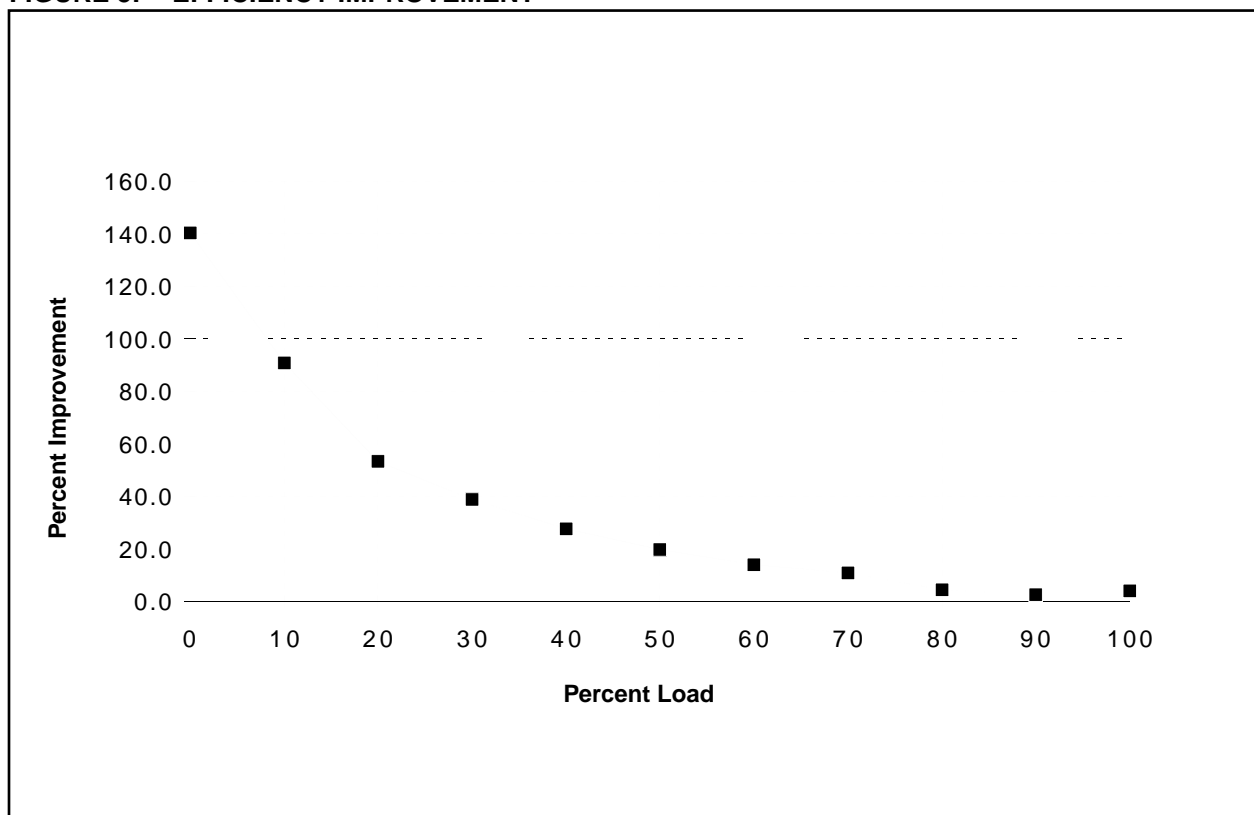


FIGURE 5: EFFICIENCY IMPROVEMENT



AN599

FIGURE 6: MOTOR CURRENT DRAW

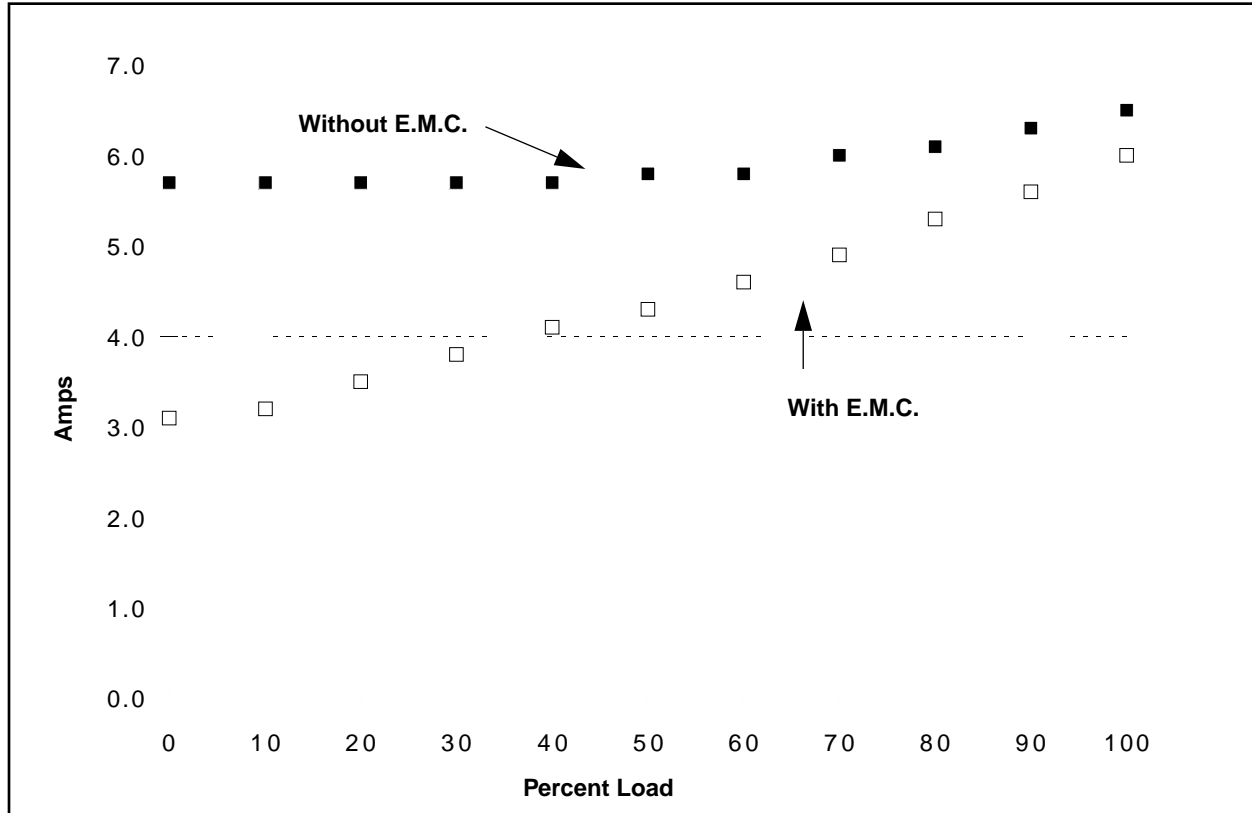


FIGURE 7: MOTOR POWER DRAW

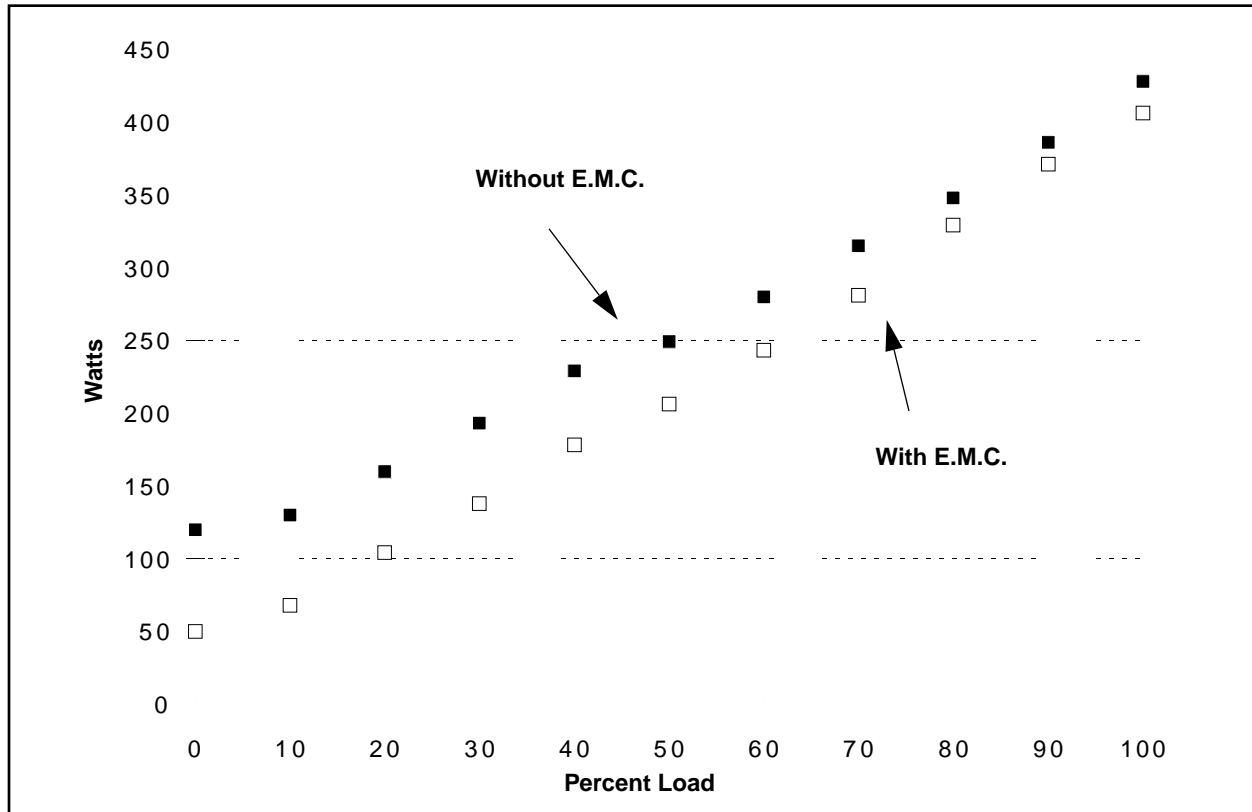
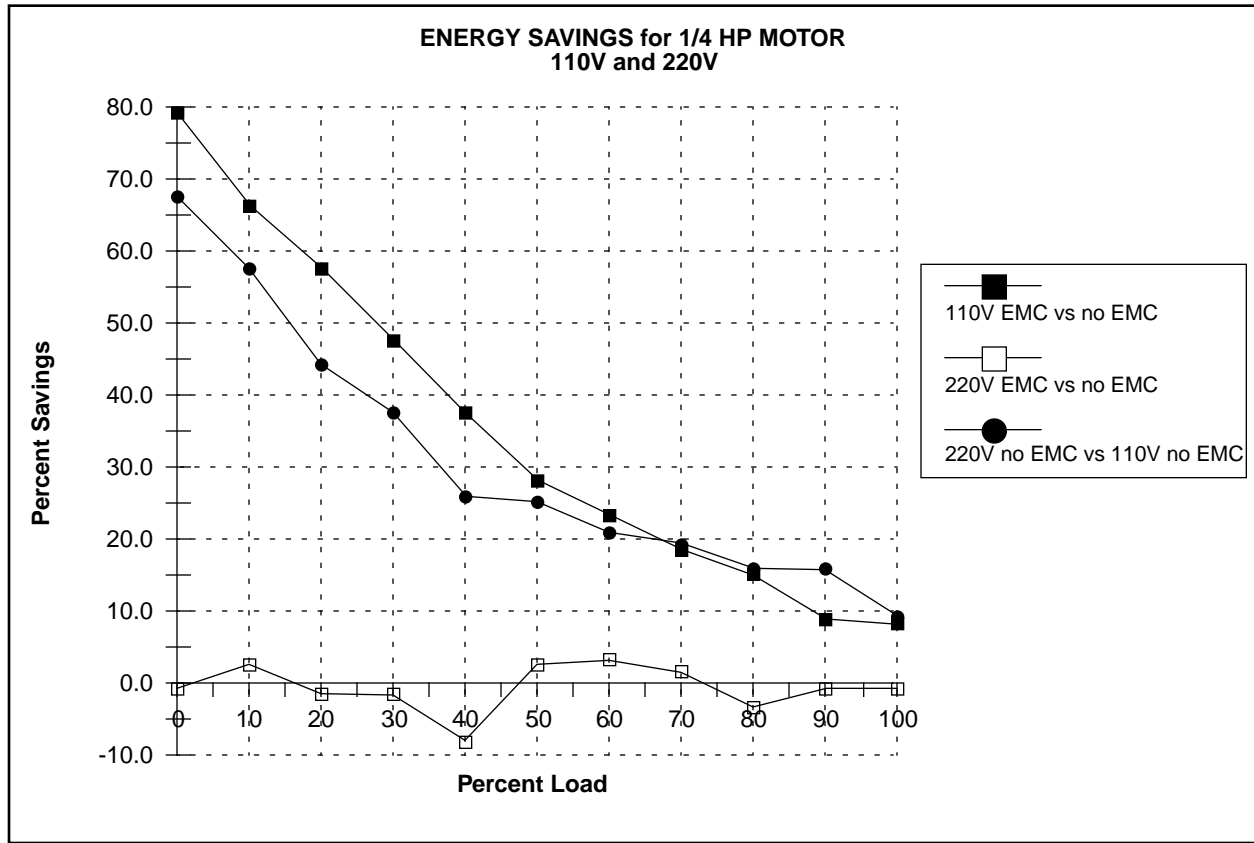


FIGURE 8: ENERGY SAVINGS FOR 1/4 HP MOTOR



AN599

TABLE 1: OPERATING PARAMETER COMPARISONS

1/3 HP Motor without E.M.C.									
Load (%)	Load (Nm)	Vrms	Irms (A)	Power Factor	Power In (W)	RPM	Power Out (W)	Power Out (HP)	Efficiency (%)
0	0.00	115	5.7	0.18	120	1791	0	0.00	0.2
10	0.14	115	5.7	0.20	130	1788	26	0.04	20.1
20	0.29	115	5.7	0.24	160	1781	54	0.07	33.7
30	0.43	115	5.7	0.29	193	1777	80	0.11	41.4
40	0.57	115	5.7	0.35	229	1768	105	0.14	46.0
50	0.72	115	5.8	0.37	249	1764	133	0.18	53.3
60	0.86	115	5.8	0.42	280	1758	158	0.21	56.4
70	1.00	115	6.0	0.46	315	1750	183	0.25	58.0
80	1.14	116	6.1	0.49	348	1744	208	0.28	59.7
90	1.29	115	6.3	0.53	386	1736	234	0.31	60.6
100	1.43	116	6.5	0.57	428	1727	258	0.35	60.3

1/3 HP Motor with E.M.C.									
Load (%)	Load (Nm)	Vrms	Irms (A)	Power Factor	Power In (W)	RPM	Power Out (W)	Power Out (HP)	Efficiency (%)
0	0.00	113	3.1	0.14	50	1794	0	0.00	0.4
10	0.14	113	3.2	0.19	68	1786	26	0.04	38.4
20	0.29	113	3.5	0.26	104	1775	54	0.07	51.7
30	0.43	113	3.8	0.32	138	1764	79	0.11	57.4
40	0.57	113	4.1	0.38	178	1755	104	0.14	58.7
50	0.72	113	4.3	0.42	206	1749	132	0.18	63.8
60	0.86	112	4.6	0.47	243	1740	156	0.21	64.3
70	1.00	112	4.9	0.51	281	1730	181	0.24	64.3
80	1.14	112	5.3	0.55	329	1722	205	0.27	62.3
90	1.29	112	5.6	0.59	371	1713	231	0.31	62.2
100	1.43	111	6.0	0.61	406	1705	255	0.34	62.7

TABLE 2: ENERGY SAVINGS

LOAD (%)	Improvement in Efficiency (%)	Energy Savings (%)
0	140.4	58.3
10	91.0	47.7
20	53.3	35.0
30	38.8	28.5
40	27.7	22.3
50	19.8	17.3
60	14.0	13.2
70	10.8	10.8
80	4.4	5.5
90	2.7	3.9
100	4.1	5.1

FIGURE 9: WAVEFORMS

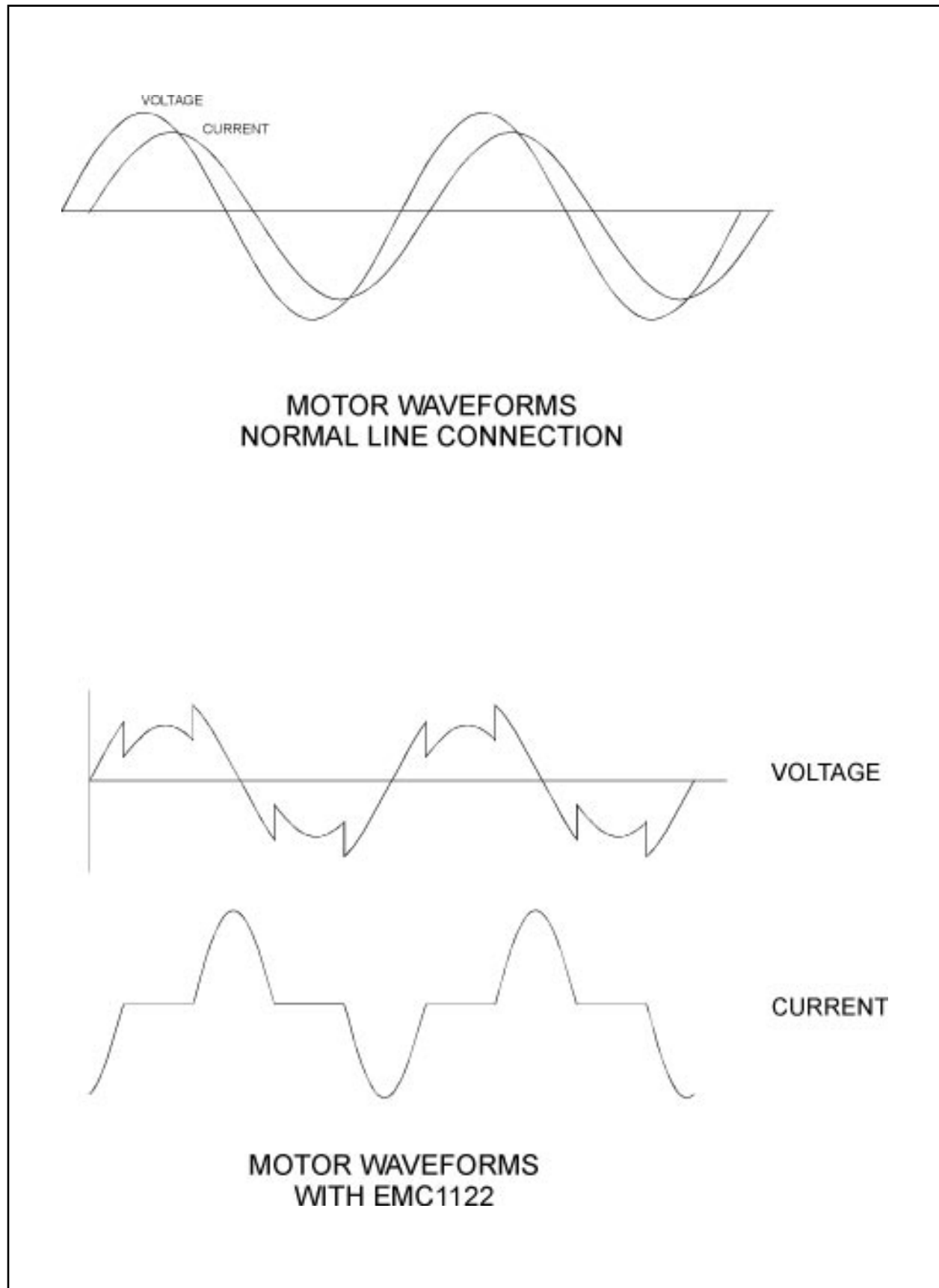
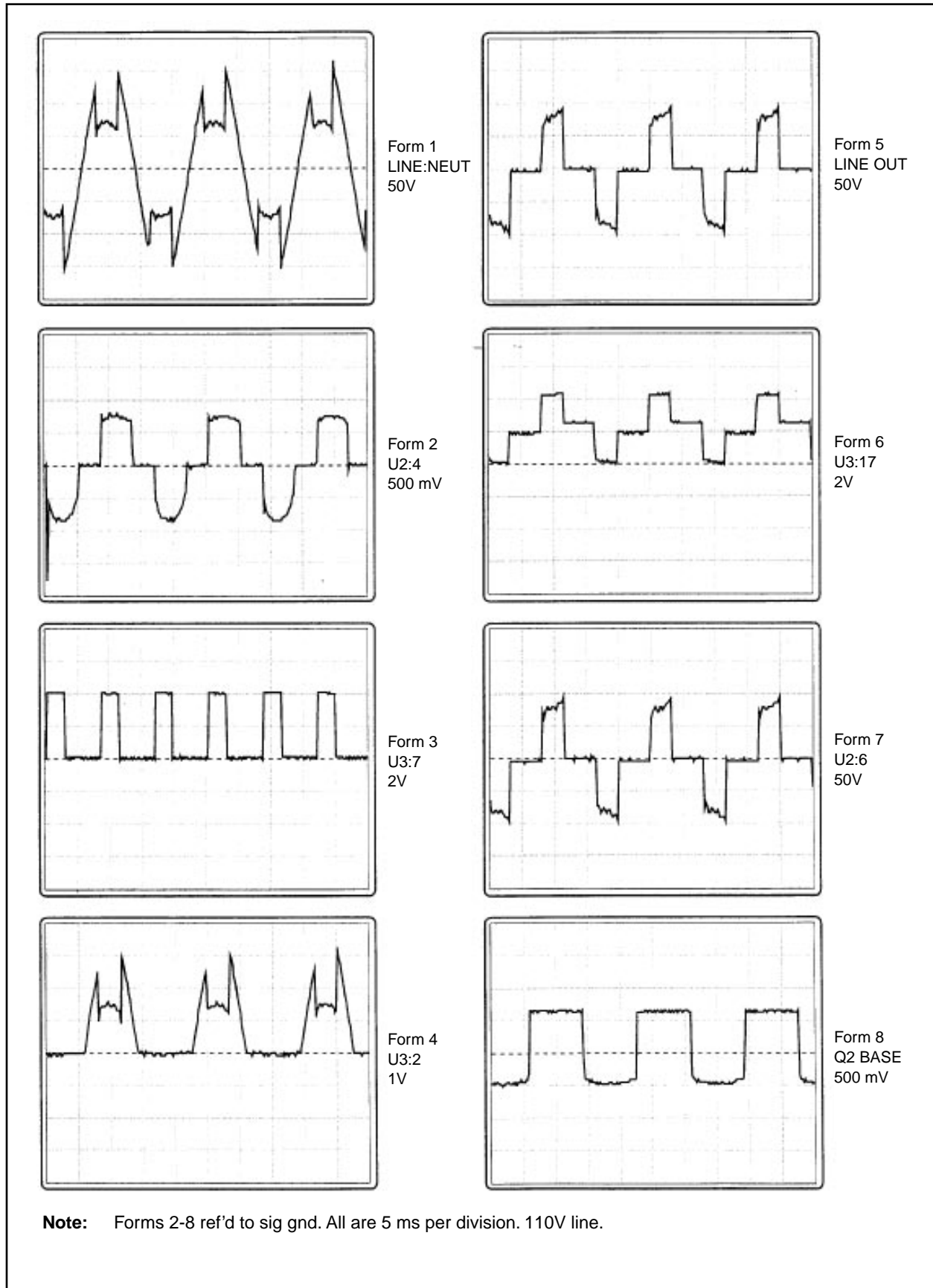


FIGURE 10: MTE1122 CIRCUIT WAVEFORMS



AN599

TABLE 3: BILL OF MATERIALS FOR MTE1122 ENERGY MANAGEMENT CONTROL DEMO BOARD

Item	Qty.	Reference	Value	Desc.ription		Mfg.
1	1	C1	100 μ F	16V	electrolytic	
2	1	C2	1 μ F	250VAC	film	
3	1	C3	220 μ F	16V	electrolytic	
4	2	C4,C5	100 pF		ceramic	
5	1	C6	0.1 μ F		ceramic	
6	1	C7	1 μ F	50V	electrolytic	
7	1	C8	2.2 μ F	250VAC	film	
8	2	C9,C10	0.1 μ F	250VAC	film	
9	3	D1,D2, D8	1N4007			
10	1	D3	LED	green		
11	1	D4	1N5226			
12	1	D5	1N4733			
13	2	D6,D7	1N5230			
14	1	L1	100 μ H	4632	RF Choke	JW Miller
15	1	Q1	TP2907			
16	1	Q2	TP2222A			
17	1	Q3	Q4015L5			Teccor
18	1	R1	470 1/2W			
19	1	R2	330 1/2W			
20	1	R3	560			
21	1	R4	270 1/2W			
22	2	R5, R15	30K			
23	1	R14	30K 1/2W			
24	1	R6	15K			
25	1	R7	560			
26	1	R8	2.4K			
27	1	R9	1M 1/2W			
28	2	R10,R13	562K 1%			
29	2	R11,R12	12.1K 1%			
30	1	R16	10K			
31	1	U1	TLC271CP	opamp		TI
32	1	U2	TLP3023	opto-triac		Seimens
33	1	U3	MTE1122			Microchip
34	1	Y1	4 MHz	ceramic resonator		
35	1			heatsink	as needed	

Note 1: C2,8,9,10 MUST be AC-rated capacitors. THIS IS CRUCIAL!

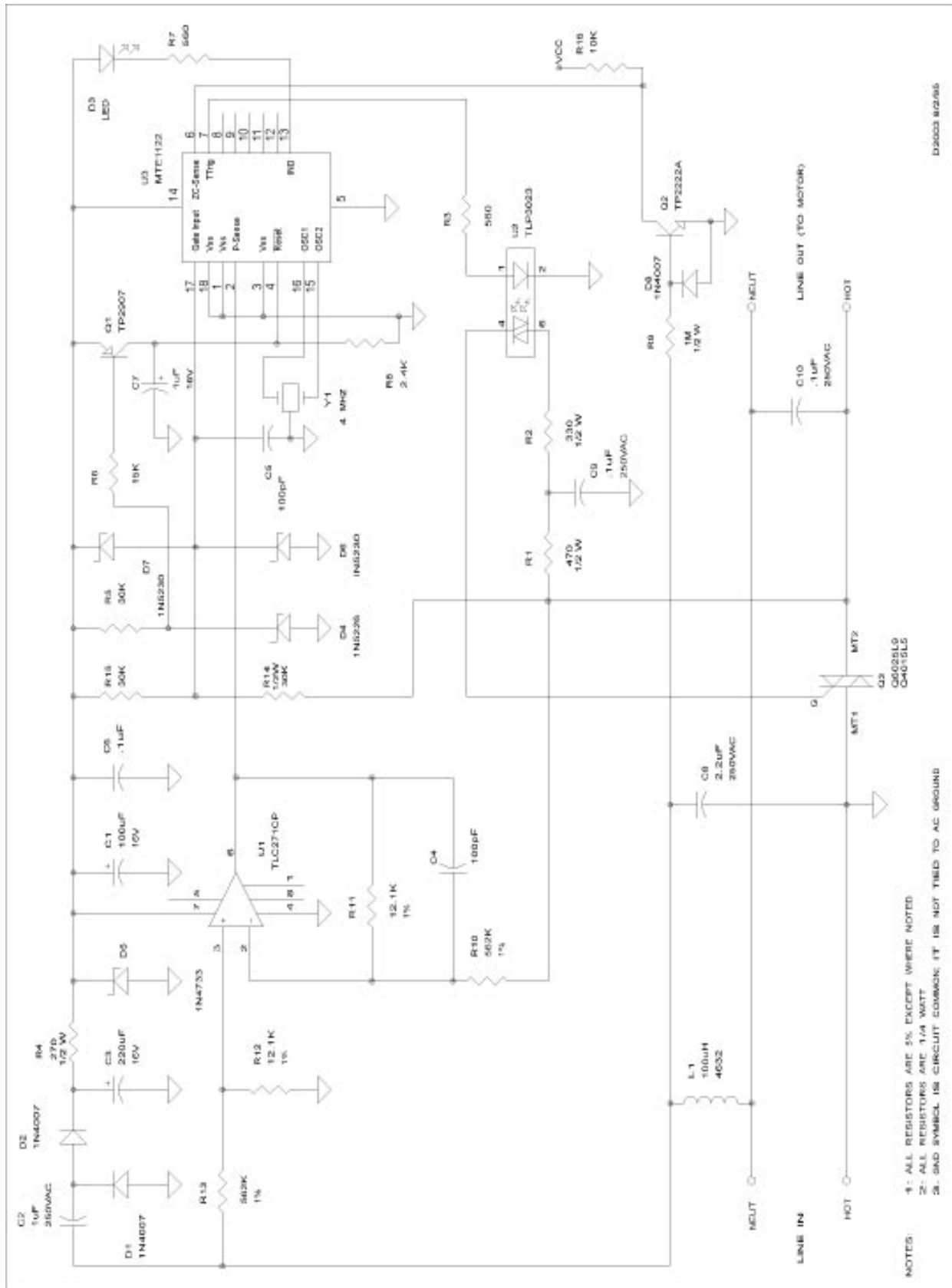
Note 2: Q3 can be sized to fit the load. A 400V 15A part is called out in the parts list and on the schematic; a 600V 25A part is also listed on the schematic for reference. Nearly any triac can be used here, as long as its trigger current does not exceed that supplied by U2 (typically 50 mA). For higher current applications, two SCR's back-to-back perform well. Performance is improved slightly if the device is NOT operated at its current limit.

Note 3: Any opto-triac meeting the current-transfer and current handling specs of the TLP3023 can be used.

Note 4: A heat sink is called out. Its size will depend on the particular Triac used, the operating temperature, and the load. Contact a heat sink manufacturer for specific information.

AN599

FIGURE 11: SCHEMATIC DIAGRAM



SECTION 4

SERIAL EEPROM APPLICATION NOTES

AN601	EEPROM Endurance Tutorial.....	4-1
AN602	How to get 10 Million Cycles out of your Microchip Serial EEPROM.....	4-5
AN608	Converting to 24LCXXB and 93LCXX Serial EEPROMs.....	4-7
AN609	Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller.....	4-9
AN610	Using the 24LC21 Dual Mode Serial EEPROM.....	4-17
AN613	Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports.....	4-23
AN614	Interfacing the 8051 with 2-wire Serial EEPROMs.....	4-37

EEPROM Endurance Tutorial

<i>Author: David Wilkie Reliability Engineer</i>
--

INTRODUCTION

The endurance of an EEPROM-based device will be quoted by a manufacturer in terms of the minimum number of erase/write cycles (write cycles) that the device is capable of sustaining before failure. A write cycle is generally considered to be the operation that changes data in a device from one value to the next.

There are several EEPROM-based devices available on the market. Microchip Technology Incorporated makes three general types of EEPROM-based product: Serial EEPROMs, Parallel EEPROMs, and EEPROM-based Microcontrollers. As a manufacturer of many EEPROM products, Microchip is concerned with endurance and continues to try to educate its customers on the importance of this topic.

There are many differences in the interpretation of “endurance” that can result in misleading or inaccurate information being used in design decisions. This paper hopes to clear up any questions that the customer may have in the subject of endurance, without becoming so technical that the information given is not helpful.

There is no widely used standard for any type of endurance test. Each manufacturer will use their own endurance testing methodology. This report will describe all the testing options, and which tests Microchip performs on its EEPROM-based products.

The MIL-STD cycling test (Method 1033) has not been updated since 1977 and is well out of date as applied to EEPROM non-volatile memories. The standard does not distinguish the difference between block cycling and byte cycling, and gives a very poor failure criteria. Microchip does not use this standard.

BASIC TERMS

The definition of “endurance” (as applied to EEPROMs) in the first part of this introduction contains various words and phrases that require clear definition and understanding. As shown in the following paragraphs, different manufacturers will use different standards.

“Endurance cycling” is a test performed by all manufacturers (and some customers) to determine how many “write cycles” the product will achieve before failing.

The “minimum number of write cycles” is the least number of times that you can expect to subject the product to a “write cycle” before it fails.

“Failure” is a somewhat arbitrary definition, since a device only truly fails when it no longer meets the customer expectation, and does not operate in his system. A failure can be defined in this, the loosest of definitions, or the most stringent of definitions (whereby a device would fail if it did not meet any of the data sheet parameters), as well as a wide range in between.

For example, if the device did not correctly store data into a particular address that the customer was not using, then the device would work correctly for the customer but would fail a functional test set by the manufacturer. Likewise, if the device drew more current than the data sheet specified after some time, but the customer application could supply the current needed, the device would work in the customer application but would fail a parametric test set by the manufacturer.

Microchip uses the most stringent definition: **A failure occurs when the device fails to meet any data sheet condition under any guaranteed operating condition of temperature and voltage.**

The number of devices that can fail before a particular endurance criteria is not met is also somewhat flexible. Even the most quality conscious manufacturer will occasionally have a failure, so a failure level is defined. The industry standard conditions for many types of reliability tests are set by JEDEC (the Joint Electronic Device Engineering Council). JEDEC defines that if 5% or less of a given sample fails at a given endurance goal, then that goal has been met. For example, if a sample of 100 units are endurance cycled to 1 million cycles and 3 have failed at 100,000 cycles and a further 7 have failed at 1 million cycles, then the sample would have an endurance of 100,000 cycles.

Microchip uses a more stringent criteria for endurance: no more than 2.5% of devices can have failed for the given endurance goal to have been met.

A “write cycle” is also a somewhat flexible definition since almost every customer will write the device in a different way. For example, if the customer application uses only the first three bytes of the array to store variable data, and the remainder of the array is used as a lookup table, then a write cycle will be complete when the three data bytes have been re-written to their new data state.

A write cycle is often described as an erase/write cycle, since almost all technologies employ an “auto-erase” before the data is actually written to the array. This is also used by Microchip, but we will use the term “write cycle” since the auto-erase is invisible to, and cannot be suppressed by, the customer.

The term “data changes” is occasionally used in place of “write cycle” or “erase/write cycle.” A data change will occur when an auto-erase cycle is initiated, and a second data change will occur upon the write cycle, therefore, one “erase/write cycle” is equivalent to two “data changes.” The term “data change” also implies that a different type of cycling is being used than “erase/write cycle.” This will be described later.

The term “write cycle” does not define under what conditions the cycling was done (unless explicitly stated) nor does it define the type of cycling that was done. The endurance cycling can be done at any number of conditions of voltage and temperature (e.g., 85°C and 5.5V, or 25°C and 5.0V) that may or may not meet with a customer’s application. The cycling mode used in endurance cycling can affect the endurance of the product. All these effects will be described later.

Microchip uses the most stringent conditions that are reasonable for endurance cycling. We use byte or page mode cycling at a temperature of 85°C at 5.5V. All data not explicitly defined at other conditions is taken at these conditions.

A “read cycle” is completed when any number of bytes of data have been read from the device. For the FLOTOX-design EEPROM-based devices made by Microchip a read cycle does not affect endurance, since the data in the EEPROM is not changed. Other technologies, such as Ferroelectric technology, may have a limited number of read cycles since data is corrupted during a read.

System Design Considerations

There are a number of design considerations that the system designer can use to maximize the endurance of an EEPROM-based device, if endurance is the application’s limiting factor.

As will be described in more detail later, if the designer has any control over certain environmental or operation conditions he should observe the following basic guidelines:

- Keep the application temperature as low as possible
- Keep the application voltage (or the VCC voltage on the EEPROM-based device) as low as possible
- Write as few bytes as possible
- Use page write features wherever possible
- Write data as infrequently as possible

With these basic guidelines applied to the fullest extent, the endurance of EEPROM-based devices can be extended well beyond the guaranteed minimum endurance. Under certain very specific conditions, Microchip Serial EEPROMs have been shown to last for well over 100 million cycles.

WRITE MODES IN EEPROMS

There are three ways that EEPROM-based devices can have the entire array data contents changed. These are: byte mode, page mode, and block (or bulk) mode. Some types of devices support all three modes, others may only support one or two modes. The mode that you use to write an EEPROM-based device will affect the long term endurance of the product.

Byte mode writing is used when the contents of the array are changed one byte at a time. For many devices this is the only user-accessible write mode available. To change the entire contents of a Serial EEPROM in this way would take up to 10 seconds (using 10 ms per page on a 64K Serial EEPROM). Parallel EEPROMs such as a 28C64, which have a faster byte write time (1 ms rather than 10 ms), but no page mode would also take up to 10 seconds.

Page mode writing is a popular feature on many new designs of EEPROM memory products. This feature allows up to 8 bytes of data to be written to the memory in the same time that one byte would normally take. In this mode, the write time for a 64K Serial EEPROM can be cut from eight seconds to one second.

Block cycle is generally a test mode used by EEPROM manufacturers to make it easier to test the products. Some types of EEPROM-based products have these modes as user options (such as the ERAL and WRAL mode in 93CXX products, or the Chip Clear mode in 28CXXX products) but generally this mode is not user accessible. A block write can be done in as little as 1 ms, allowing millions of write cycles to be completed in a few hours.

A general rule to follow in choosing write modes is that the larger the number of bytes being written in a single instruction, the longer the device will last. For example, in byte mode a device might start to fail after 300,000 cycles under a particular set of conditions, but the device may last 600,000 cycles in page mode under the same conditions. In block mode the device might last 1 million cycles, under the same conditions.

The reason for this is related to the internal design of any FLOTOX EEPROM-based product. In these devices, an internal “charge pump” takes the applied VCC voltage (typically 2.5 to 5.5V) and increases it to 15 to 25V. This voltage is required to induce the “Fowler-Nordheim Tunneling” or “Enhanced Emission” effects that are used to program and erase EEPROM-based devices. The specific effect that is used is manufacturer-dependent. Microchip EEPROMs program by Fowler-Nordheim Tunneling.

The charge pump voltage is used to program however many EEPROM-cells are being programmed. For example, in byte mode, all the cells in a byte (8 to 16) are biased with the charge pump voltage. In block mode, all the cells in the array (up to 100,000, depending on the device) are biased with the charge pump voltage. The charge pump is like a current source during conditions of high load, so the voltage put out by the charge pump will be reduced slightly if more bytes are being written. If the whole array is being programmed then the charge pump voltage will be significantly reduced, but the programming current I_{pp} will be very high.

Generally, the lower the charge pump voltage the better the endurance (there is a limit since the charge pump voltage needs to be high enough to program the cell) and so the best endurance is generally achieved by using block mode cycling. Page mode is worse than block mode, but better than byte mode. Block mode is generally not a very useful cycling mode to the end user, since the data contents in the whole array will be changed to the same value (generally 00 or FF).

When Microchip tests EEPROM-based products we use byte mode cycling on devices which do not have a page mode, and page mode cycling for those that do. We encourage our customers to use page mode writing on all products which have page mode, to ensure high endurance.

ENDURANCE TESTING METHODOLOGIES

Different manufacturers use different ways to both endurance cycle and test EEPROM-based products. There is no standard for endurance cycling, or testing devices after cycling.

There are two groups of testing that Microchip performs on all products: qualification and production. Qualification testing is done for all new products, and major changes to a product or manufacturing process. Production testing is done on all devices shipped to customers.

Qualification testing at Microchip is used to test the reliability of new products, and to guarantee that the device is reliable. A great deal of testing is done, including endurance testing on all EEPROM-based products. Endurance cycling is done at the maximum rated data book value, generally 85°C or 125°C. After the rated number of cycles (10,000, 100,000, or 1,000,000) the sample (around 300 from multiple wafer lots) is tested to a full production test program. After endurance, the units are subject to "data retention" to guarantee that the required 40 years of data retention will be achieved, after the maximum number of cycles has been completed.

Endurance cycling is done under the conditions previously described, and the data retention test is performed after this. After the data retention stress is

completed (which takes six weeks) the devices are tested again, to confirm the functionality of the device to all data sheet parameters.

No more than 2.5% failures are allowed after endurance, and a failure rate higher than 100 FITs after 1000 hours of data retention stress (equivalent to more than 10 years at 55°C) is unacceptable.

Production testing is done by Microchip on all devices shipped to a customer. Production testing begins immediately after a wafer lot is finished being processed, continuing in various stages until the devices are shipped to a customer.

The first tests that are done on EEPROM-based products at Microchip are called wafersort. They are done before the wafer is cut up into dice for assembly. There is a series of tests which include large numbers of write cycles (up to 5000) to ensure reliability by weeding out weak devices so they never get shipped.

After assembly full testing is done which includes further write cycles across the guaranteed temperature range, to ensure device functionality at the temperature extremes. After the normal production testing a sample of 128 units is taken from every wafer lot, and cycled to 10,000 cycles (Parallel EEPROMs) or 1,000,000 cycles (Serial EEPROMs and EEPROM microcontrollers) at 85°C using the conditions already described. After endurance testing the devices are tested for functionality at 85°C.

Any sample of Parallel EEPROMs which shows significant failures at the end of cycling causes the entire wafer lot to be pre-cycled prior to production testing. Serial EEPROMs and EEPROM-based microcontrollers do not receive the same disposition. Lots with significant failures come under scrutiny and full failure analysis with corrective actions is done. This is, however, a rare occurrence.

The testing that Microchip does is unique. Manufacturers will generally do different testing from each other. Microchip firmly believes that our testing ensures excellent quality and reliability.

THE EFFECT OF TEMPERATURE ON EEPROM ENDURANCE

The temperature at which cycling is done will affect the number of write cycles that can be executed before the device fails. The higher the temperature, the worse the endurance will be. Generally, and approximately, a device which fails at 10 million cycles at 25°C will fail at 2 million cycles at 85°C and 1 million cycles at 125°C. The reasons for this are not conclusive (although there is much technical literature supporting one theory or another) but it is apparent that the failure mode of EEPROM cells (generally considered to be electron trapping in the tunnel dielectric causing shielding and dielectric breakdown) is strongly dependent on temperature.

AN601

Data taken by Microchip suggests that if the typical failure of an EEPROM-based device is 10 million cycles at 25°C, the mean failure will occur at other temperatures according to the following table:

TABLE 1: TEMPERATURE MEAN FAILURE

Write Cycle Temperature	Mean Failure (Cycles)
-40°C	37.1 million
0°C	16.7 million
25°C	10.0 million
40°C	7.4 million
55°C	5.4 million
70°C	4.0 million
85°C	2.9 million
100°C	2.2 million
125°C	1.3 million

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance™ disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics.

As is clearly seen, any cycling done at 25°C can be misleading in the extreme if the application requires a device that can be cycled 10 million times at, say, 55°C.

THE EFFECT OF VOLTAGE ON EEPROM ENDURANCE

The voltage at which a device is written can also affect the endurance. This is simply because the charge pump (used to program and erase EEPROM cells) is more powerful at higher voltages. As has already been described, a higher programming voltage will reduce the endurance of an EEPROM cell, and a stronger charge pump will produce a higher voltage.

Data taken by Microchip suggests that if typical failure of an EEPROM-based device is 1 million cycles when endurance cycling is done at 5.5V, mean failure occurs at other temperatures according to the following table:

TABLE 2: VOLTAGE MEAN FAILURE

Endurance Cycling Voltage	Mean Failure (Cycles)
5.5V	1.0 Million
5.0V	1.2 Million
4.5V	1.4 Million
4.0V	1.7 Million
3.5V	2.0 Million
3.0V	2.4 Million
2.5V	2.8 Million
2.0V	3.3 Million

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics.

THE EFFECT OF WRITE MODE ON EEPROM ENDURANCE

As has been discussed there are three basic ways of writing data to an EEPROM-based device:

- Byte mode
- Page mode
- Block mode

This is related to the strength of the charge pump in applying the required programming voltages to the EEPROM cells.

Data taken by Microchip suggests that if the typical failure of an EEPROM-based device is 1 million cycles when the endurance cycling is done in byte mode, the mean failure will occur in other modes according to the following table:

TABLE 3: ENDURANCE MEAN FAILURE

Endurance Cycling Mode	Mean Failure (Cycles)
Byte	1.0 Million
Page	4.6 Million
Block	13.2 Million

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics. This data was taken from a Microchip 24LC16.

As you can see, the use of the block cycle data to guarantee endurance can be misleading.

THE TOTAL ENDURANCE PREDICTIVE SOFTWARE

Microchip has a Windows®-based software model called Total Endurance. This program, based on all of the customers endurance parameters, will predict the failure level at the expected end of application life. This tool is invaluable for system designers who would like to fine-tune their application in favor of endurance. It is available now from your local Microchip distributor.



AN602

How to get 10 Million Cycles out of your Microchip Serial EEPROM

<i>Author:</i> David Wilkie Reliability Engineer

INTRODUCTION

Microchip Technology Incorporated recently became the first manufacturer of Serial EEPROMs to rate the endurance of 1K to 16K 2-Wire protocols and 1K to 4K 3-Wire protocols at 10 million cycles. This gives the highest endurance available from any manufacturer on such a wide range of Serial EEPROMs. Microchip is also the world leader in endurance application support, including Total Endurance™ software, which allows customers to determine exactly what their endurance will be in their application. This application note will explain our 10 million cycle guarantee in the context of Total Endurance and the customer application.

ENDURANCE

Endurance is the term used to define the number of times an EEPROM-based device can be erased and re-written before an unacceptable failure rate has occurred. Microchip defines an unacceptable failure rate as 2.5%, which is half the "industry standard" (JEDEC) failure rate of 5%.

Microchip uses the industry standard cycling conditions to guarantee 10 million cycles. These are the same conditions used by the major EEPROM manufacturers, unless specifically stated otherwise. The conditions are 25°C block cycle at 5.0V. The data pattern that is written to the device is supplier-dependent. Microchip uses a checkerboard pattern to better simulate actual use.

10 MILLION CYCLE GUARENTEE

We take a great deal of data on the endurance of our devices. We run experiments over temperature, voltage, array size, data pattern, cycling type and other erase/write cycling conditions, to continually learn about our products. The data we take is continually fed back into our manufacturing process tracking, and is also used to periodically update our application predictive software, Total Endurance. The lessons that we learn are passed on to our customers.

Microchip has shown many times before that temperature has the strongest effect on endurance. The higher the temperature of your application is, the lower your endurance will be. For example, no EEPROM manufacturer can deliver a product that will achieve 1 million cycles at 125°C, but many can deliver 1 million cycles at room temperature. Microchip can provide 10 million cycles on all our Serial EEPROMs at room temperature, using the standard cycling conditions.

You can ensure the lowest possible failure rate at 10 million cycles by using the EEPROM in ways that are easiest on endurance. Use the lowest temperature possible. Use the lowest voltage possible. Write as few bytes as possible (the number of reads is unlimited), and use page modes if the device you are using has one (Typically all 2-wire protocols have a page mode, but 3-wire protocols do not).

The best way to determine the endurance of your application is to use the Total Endurance software. This will give accurate results over a very wide range of application-specific conditions, including array size, temperature, and voltage (the main effects on cycling).

The most important thing to remember is that endurance is application-specific. Each application will result in a different endurance level, based on the temperature, voltage, and other conditions under which the application operates. Currently Microchip is the only manufacturer to offer such a wide range of tools to help guide the user in his or her application of Serial EEPROMs.

AN602

NOTES:

Converting to 24LCXXB and 93LCXX Serial EEPROMs

*Author: Nathan John
Memory Products Division*

Microchip Technology Inc. is constantly striving to give you, our customer, greater value. In the Serial EEPROM market this means creating devices that exhibit greater reliability, that consume less power, have more features and are priced aggressively with respect to our competitors. We have created the 24LCXXB and 93LCXX families of Serial EEPROMs to meet all of these goals and more. We are proud to announce that these families of EEPROMs are the

world's first to be guaranteed to 10 Million Erase/Write cycles. Please see Table 1 below for a complete list of the benefits for converting from "C" designated parts from Microchip and other vendors.

For most applications, there are no hardware or software compatibility issues in converting to the "LC" parts, you can drop them into the socket and you will be off and running. A small percentage of designs could require some modification to your hardware or software. Please review Table 2 and Table 3 to ensure that the "LC" parts will work correctly in your systems.

TABLE 1: FEATURES AND BENEFITS OF CONVERSION

Feature	Benefit
Higher Endurance	Opens new applications for Serial EEPROMs Lower FIT rate at given Write Cycle point
Smaller Die	Lower Cost
Lower Voltage / Lower Power	Enables new battery powered uses Allows for lower overall system VCC
Smaller Process Geometry	Avoids obsolescence of older technology
Schmitt Trigger Inputs (24LCXXB)	Filtering for better noise immunity
Rotated Pinout Versions (93LCXX/SN)	Conforms to alternate industry pinout
Hardware Write Protect (24LCXXB)	Enhanced data protection

TABLE 2: CONVERSION ISSUES FOR 24LCXXB

Conversion Issue	Detail
Addressing	The "LC" versions do not have active address pins and, therefore, you can only have one of these devices on the I ² C™ bus.
Write Speed	The "LC" parts have a maximum write time of 10ms, while the "C" parts have a write time of 1ms.
Write Disable Voltage	The "LC" versions will write down to 1.5V, while the "C" versions will disable writing below 4.0V.
Write Protection	The "C" parts have either no write protection or protection on one half of the array, the "LC" parts all have full array protection, implemented by connecting pin 7 to Vcc.

AN608

TABLE 3: CONVERSION ISSUES FOR 93LCXX

Conversion Issue	Detail
Write Speed	The "LC" versions have a maximum write time of 10ms, compared to 1ms for "C."
Write Disable Voltage	The "LC" parts will write down to 1.5V while the "C" parts will only write down to 4.0V.
Start of Write Cycle	The "LC" versions start the write cycle on the low going edge of chip select, the "C" versions start on the last data bit clock.

Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller

<i>Author:</i> Keith Pazul Memory and ASSP Division
--

INTRODUCTION

There are many different microcontrollers on the market today that are being used in embedded control applications. Many of these embedded control systems need non-volatile memory. Because of their small footprint, byte level flexibility, low I/O pin requirement, low power consumption and low cost, Serial EEPROMs are a popular choice for non-volatile storage.

Microchip Technology Incorporated addresses this need by offering a full line of Serial EEPROMs covering industry standard serial communication protocols for 2-wire and 3-wire communication. The theory, operation and differences of these two protocols are discussed in detail in Microchip's application note AN536. The reader should refer to AN536 if unfamiliar with 2-wire and/or 3-wire communication protocols. Serial EEPROM devices are available in a variety of densities, operational voltage ranges and packaging options.

Microchip realizes that its customer base is very broad, and because of this, different microcontrollers are used to interface to Serial EEPROMs. One of the microcontrollers used in these applications is the Motorola® 68HC11. In order to simplify the design process, Microchip has written a 68HC11 assembly code to communicate with our 2-wire and 3-wire parts that is verified and tested to function properly.

There are 13 programs written for inclusion in this application note. A listing of one of the programs is included with this application note as an example. The source code for all of the programs is available for downloading via Microchip's Bulletin Board (BBS). Users may consult the index of the *Microchip Embedded Control Handbook* for log-on instructions for the BBS. Once logged on to the Microchip BBS, select the FILE LIBRARY (download files) option from the main menu. Next, select a library. This application note is located in the MEM_APPS file library. Once in the MEM_APPS file library, the proper application note can be selected and downloaded by following the directions supplied by the BSS system. Only one file needs to be downloaded from the Microchip BBS to get all of the source code. All source code files are combined and compressed into one downloadable file called AN609.zip.

For Microchip 2-wire devices (24CXXA, 24LCXX, 24LCXXB, 24AAXX devices, excluding 24XX32 and 24XX65 devices), there are three programs to perform some of the basic communication functions.

- SRDMT2W.ASM - 2-Wire Sequential Read
- BW4MT2W.ASM - 2-Wire Sequential Write
- BW4PMT2W.ASM - 2-Wire Byte Write with Data Polling

Refer to the data sheets in the Microchip *Data Book* for the 2-wire devices listed above for explanations of sequential read, sequential write and data polling.

Microchip also offers a powerful and flexible family of products referred to as Smart Serial™. These devices use the same 2-wire interface as described above, but have added intelligence not found on Microchip's other 2-wire devices. These devices include the 24C32, 24LC32, 24AA32, 24C65, 24LC65, and 24AA65. Among Smart Serial features are: split erase/write cycle endurance (user selectable regions of the device with two different endurance ratings), a 64-byte write cache and the ability to permanently write protect part or all of the array.

- RRDMT65.ASM - Random Read from 64K Smart Serial
- SRDMT65.ASM - Sequential Read from 64K Smart Serial
- CACWMT65.ASM - Full Cache Write to 64K Smart Serial
- WR8MMT65.ASM - Sequential Byte Write to 64K Smart Serial
- WR8PMT65.ASM - Sequential Byte Write with Data Polling to 64K Smart Serial
- HEMT65.ASM - Setting of High Endurance block for 64K Smart Serial
- SECMT65.ASM - Setting Security Features for 64K Smart Serial

Refer to individual data sheets of the parts listed for the definition and explanation of the functions.

For Microchip 3-wire devices (93CXX, 93LCXX, 93LCXXB, and 93AAXX devices), there are three programs included to perform some of the basic communication functions.

- MT4BR3W.ASM - 3-Wire Multiple Word Read
- MT4BW3W.ASM - 3-Wire Multiple Word Write
- MT4BW3PW.ASM - 3-Wire Multiple Word Write with Data Polling

AN609

Refer to the individual data sheets of the 3-wire devices listed for explanations of multiple word read, multiple word write and data polling.

Figure 1 describes the hardware schematic for the interface between Microchip's 2-wire devices and the Motorola 68HC11E9. This schematic applies to the connection of the Smart Serial devices as well. Figure 2 describes the hardware schematic used to connect Microchip's 3-wire devices to the Motorola

microcontroller. The schematics show the connections necessary between the microcontroller and the serial EEPROM, and the software was written assuming these connections.

FIGURE 1: CIRCUIT DIAGRAM FOR 68HC11 TO 2-WIRE SERIAL EEPROM INTERFACE

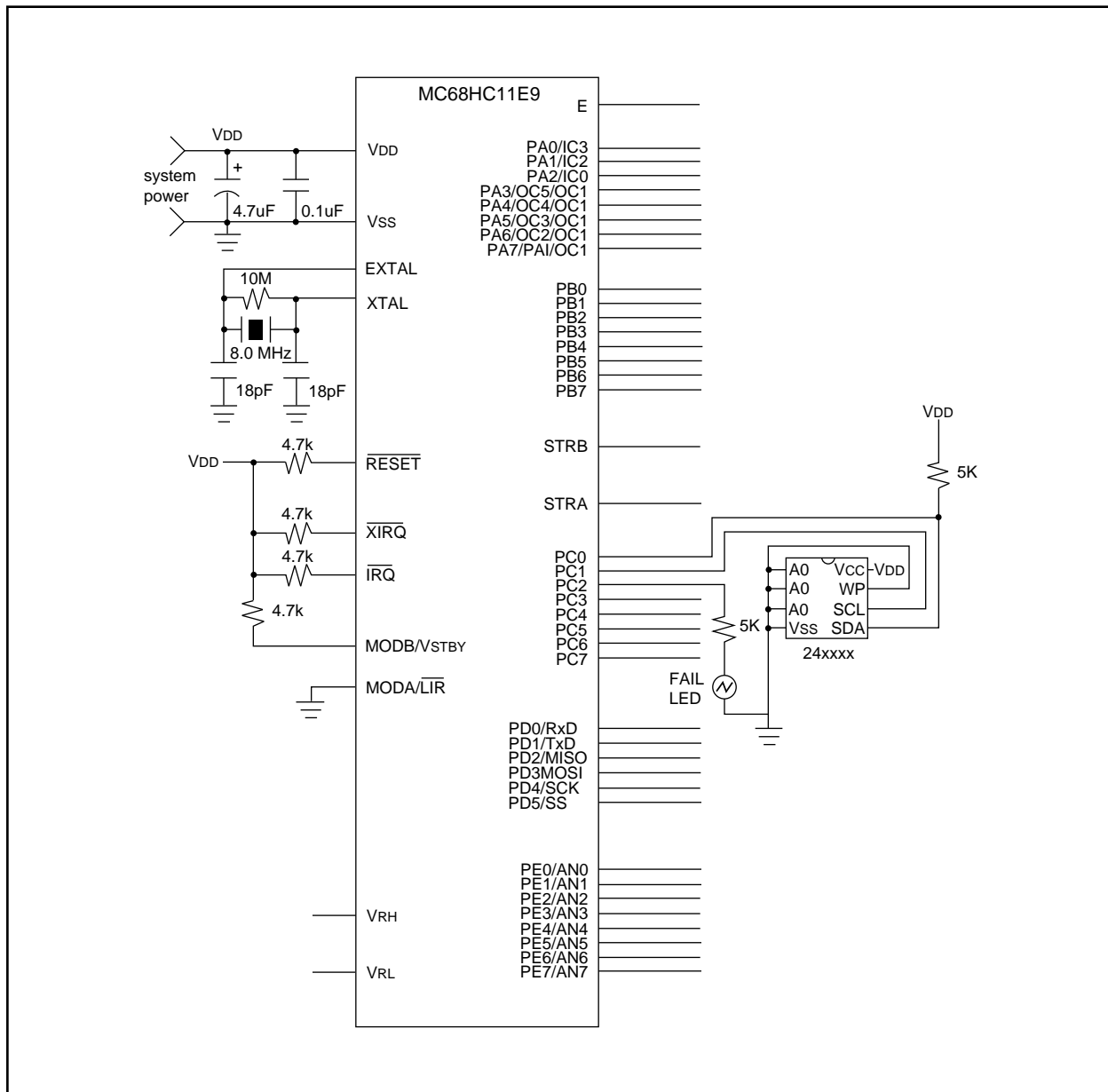
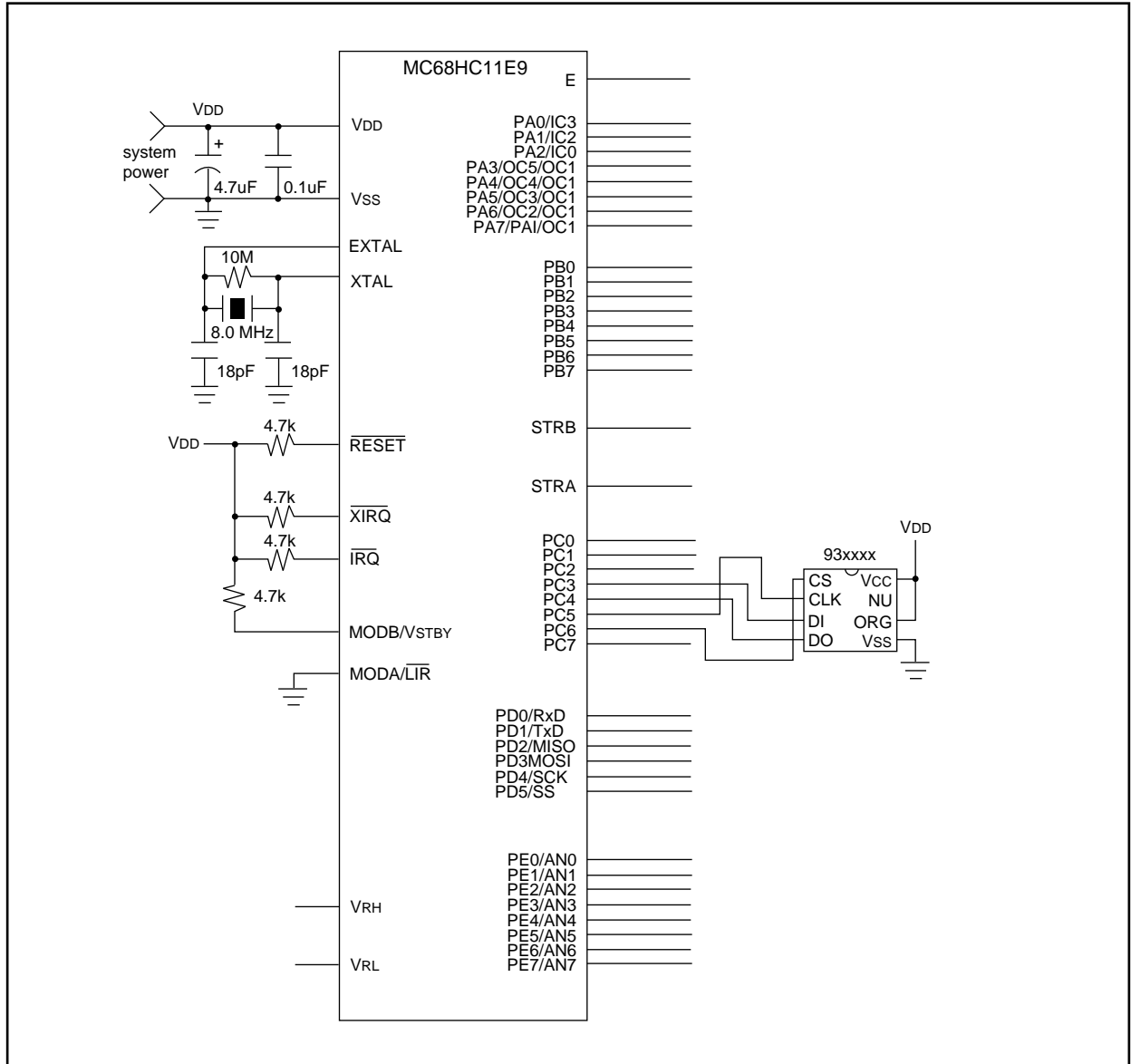


FIGURE 2: CIRCUIT DIAGRAM FOR 68HC11 TO 3-WIRE SERIAL EEPROM INTERFACE



Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: SOURCE CODE

```
; *****
; 2-Wire Sequential Write Program (225 bytes)
;
; This program (bwr4mt2w.*) writes 4 bytes of $A5 to a Microchip 24LCxx
; beginning at location $10 using a Motorola 68HC11 microcontroller.
; This code has been verified that it writes properly to a Microchip
; 24LCxx serial EEPROM.
;
; This program was written using a 68CH11EVBU evaluation board.
; This board has a monitor program in firmware of the 68HC11 which
; allows single stepping, register viewing, modifying, etc. Since this
; is the case, the program code will be loaded into the on-chip EEPROM.
; This EEPROM begins at $B600. The control registers are left to their
; default location of $1000 and the RAM is left to its default location.
; RAM locations of $48-$ff are used by the monitor program and are not
; available for program use. Therefore, the stack pointer is set a $47
; and will be able to use all of the RAM to $00, and the RAM variables
; begin at location $100 and go up from there. I cannot program the
; reset vector since it is ROM space (at $FFFE), so the way I run this
; program is to use the monitor program that comes with the evaluation
; board to set the program counter to the starting address of my user
; program ($B600) and begin from there. For users who do have access
; to the reset vector, the label of the beginning program (in my case, it
; is called START) should be loaded at location $FFFE. This program was
; not assembled using a Motorola assembler, but was assembled using
; Universal Cross-Assemblers Cross-32 Meta-Assembler. It has the
; ability to assemble just about any microcontroller code. There are
; certain commands that are unique to the cross-assembler. These
; commands will be commented differently than other comments to
; be recognizable. They will look like this:

;+++++
; Special cross-assembler command(s)
;+++++

; I do not know the exact assembler commands required to accomplish
; assembly using a Motorola assembler.
;
; The crystal that comes with the evaluation board is 8 Mhz.
; With this used as the clock input, this code will output a serial
; data stream at approximately 32khz. Although this does not
; meet I2C spec of 100 khz, it communicate with the Microchip
; 24LCxx parts properly. If a different frequency crystal is used,
; the code may need to be modified to meet timing specifications.

;*****

;+++++
0000 CPU "C:\WINC32\68HC11.TBL" ; LOAD TABLE
0000 HOF "MOT8" ; Hex output is Motorola S-records
0000 PAGE 60 ; Sets # of lines in list file
; to 60 before pagebreak
;+++++
;
;*****
; 68HC11 control register locations
;*****
1000 = REGBS EQU 1000H ; BEGINNING OF REGISTERS
```

```

0100 =      RAMBS      EQU    100H          ; BEGINNING OF RAM VARIABLES
1007 =      DDRC      EQU    REGBS+07H     ; DATA DIRECTION REG FOR PORT C
1003 =      PORTC     EQU    REGBS+03H     ; PORT C DATA REGISTER
0003 =      PCOFF     EQU    03H          ; OFFSET FROM CONTROL REG BEG.
;*****
;*****
; User defined constants
;*****
0003 =      CHIDHI    EQU    00000011B    ; SET BOTH CLK AND DATA HI
0002 =      CHIDLO    EQU    00000010B    ; SET CLK HI AND DATA LO
0001 =      CLODHI    EQU    00000001B    ; SET CLK LO AND DATA HI
0000 =      CLODLO    EQU    00000000B    ; SET CLK AND DATA BOTH LO
0001 =      DIMASK    EQU    00000001B    ; BIT MASK FOR DATA IN BIT
0080 =      DOMASK    EQU    10000000B    ; BIT MASK FOR DATA OUT BIT
0001 =      SDAMASK   EQU    00000001B    ; BIT MASK FOR SERIAL DATA
0002 =      SCKMASK   EQU    00000010B    ; BIT MASK FOR SERIAL CLOCK
0004 =      LEDMASK   EQU    00000100B    ; BIT MASK FOR ACK FAILED LED
;*****

0000 8E0047      LDS    #0047H              ; STACK POINTER BEGINS AT $47

0100              ORG    100H              ; RAM variables begin at 100h

;+++++
; DFS is a Universal Cross-Assembler directive that stands for define
; storage. 1 is for byte, 2 is for word, 4 is for long word. These are
; the user defined RAM variables.

0100      TXBUFF     DFS    1              ; 100H
0101      EE_IN      DFS    1              ; 101H
0102      ADDR       DFS    1              ; 102H
0103      RXBUFF     DFS    1              ; 103H
0104      BYTECNT    DFS    1              ; 104H
;+++++
;*****
; Program code cannot be placed in ROM for eval board because
; eval board firmware is loaded in ROM. Program code will be
; loaded in EEPROM (which is 512 bytes and begins at B600h).

; Serial data (SDA) is located on port c, pin 0
; Serial clock (SCK) is located on port c, pin 1
; An acknowledge fail LED is connected to port c, pin 2. When
; the ack is not low like it should be, the ack failed LED is
; illuminated.

B600      ORG    0B600H
;*****
;
; This is the main portion of the code. It is where the reset
; vector should set program counter to.
;
;*****

B600      START
B600 8610      LDAA   #00010000B          ; ADDRESS OF MEMORY TO BEGIN
B602 B70102    STAA  ADDR                ; WRITING DATA
B605 8604      LDAA   #4                 ; NUMBER OF BYTES TO WRITE OUT
B607 B70104    STAA  BYTECNT            ;
B60A CE1000    LDX   #REGBS             ; LOAD $1000 INTO X INDEX REG

B60D BDB639    JSR   STRTBIT            ; GOTO STRTBIT SUBROUTINE
B610 86A0      LDAA   #10100000B        ; LOAD CONTROL BYTE INTO
B612 B70100    STAA  TXBUFF            ; TXBUFF FOR OUTPUT TO SEEPROM

```

AN609

```
B615 BDB6A2          JSR   TXBYTE          ; OUTPUT 1 BYTE TO SEEPROM

B618 B60102          LDAA  ADDR              ; GET ADDRESS AND LOAD IN
B61B B70100          STAA  TXBUFF           ; TXBUFF FOR OUTPUT
B61E BDB6A2          JSR   TXBYTE          ; OUTPUT 1 BYTE TO SEEPROM

B621 F60104          LDAB  BYTECNT           ;
B624 86A5 NEXTWRDLDA #10100101B        ; DATA BYTE TO OUTPUT IS A5H
B626 B70100          STAA  TXBUFF           ;
B629 37              PSHB                    ; PUSH DATA BYTE COUNTER TO
;                          ; STACK
B62A BDB6A2          JSR   TXBYTE          ; OUTPUT 1 BYTE TO SEEPROM
B62D 33              PULB                    ; PULL DATA BYTE COUNTER FROM
;                          ; STACK
B62E 5A              DECB                    ; HAVE WE OUTPUT CORRECT # OF
;                          ; DATA BYTES?
B62F C100            CMPB  #00H              ;
B631 26F1            BNE  NEXTWR           ; NO, THEN SEND NEXT BIT

B633 BDB653          JSR   STOPBIT         ; SEND STOP BIT TO BEGIN
;                          ; INTERNAL WRITE CYCLE

B636 7EB600          JMP   START              ; START OVER AGAIN
;*****
;
;      Start bit output subroutine
;
;*****
B639                STRTBIT
B639 8607            LDAA  #00000111B        ;
B63B B71007          STAA  DDRC              ; PORT C ALL INPUTS EXCEPT BITS
;                          ; 0,1,2

B63E 8603            LDAA  #CHIDHI          ; SET SCLK AND SDATA HI
B640 B71003          STAA  PORTC           ;
B643 01              NOP                    ; OBEY PROPER START BIT SETUP
;                          ; TIME
B644 01              NOP                    ;
B645 01              NOP                    ;
B646 01              NOP                    ;
B647 01              NOP                    ;
B648 1D0301          BCLR  PCOFF,X,SDAMASK   ; SET DATA LOW FOR STOP BIT
B64B 01              NOP                    ; OBEY PROPER START BIT HOLD
;                          ; TIME
B64C 01              NOP                    ;
B64D 01              NOP                    ;
B64E 01              NOP                    ;
B64F 1D0302          BCLR  PCOFF,X,SCKMAS   ; SET CLK LO
B652 39              RTS                    ; END START BIT SUBROUTINE
;*****

;*****
;
;      Stop bit output subroutine
;
;*****

B653                STOPBIT
B653 8607            LDAA  #00000111B        ;

B655 B71007          STAA  DDRC              ; PORT C ALL INPUTS EXCEPT FOR
;                          ; BITS 0,1
B658 1D0301          BCLR  PCOFF,X,SDAMASK   ; MAKE SURE DATA BIT IS LOW
B65B 1C0302          BSET  PCOFF,X,SCKMASK   ; CLK BIT HI
```

```

B65E 01          NOP          ; OBEY PROPER STOP BIT SETUP
                                ; TIME
B65F 01          NOP          ;
B660 01          NOP          ;
B661 01          NOP          ;
B662 1C0301     BSET  PCOFF,X,SDAMASK ; DATA BIT HI CAUSES STOP BIT
B665 39          RTS          ; END STOP BIT SUBROUTINE
;*****
; Remember to wait internal write cycle time or acknowledge
; pole here until writing is complete before beginning next
; write to part.
;*****

;*****
;
; This routine reads in one bit from the data line
;
;*****

B666          INBIT
B666 8606       LDAA  #0000110B      ; SET SDATA AS INPUT AND KEEP
B668 B71007     STAA  DDRC          ; SCLK AS OUTPUT

B66B 7F0101     CLR   EE_IN         ; GUESS INPUT IS A 0

B66E 1C0302     BSET  PCOFF,X,SCKMASK ; SET CLK BIT HI
B671 01         NOP                ; WAIT TO READ INPUT
B672 B61003     LDAA  PORTC         ; GET INPUT FROM SDATA
B675 1D0302     BCLR  PCOFF,X,SCKMASK ; BRING CLK LO AFTER PORT READ
B678 8501       BITA  #DIMASK       ; SEE IF INPUT IS 1 OR 0
B67A 2705       BEQ   DONEIN        ; INPUT IS A ZERO

B67C 86FF       LDAA  #0FFH         ; INPUT BIT IS ACTUALLY A 1
B67E B70101     STAA  EE_IN         ; STORE BACK IN EE_IN

B681          DONEIN
B681 39         RTS
;*****

;*****
;
; This routine writes out one bit to the sdata line
;
;*****

B682          OUTBIT
B682 8607       LDAA  #0000111B      ;
B684 B71007     STAA  DDRC          ; BOTH CLK AND DATA ARE
                                ; OUTPUTS

B687 B60100     LDAA  TXBUFF         ; WHAT ARE WE TRYING TO
                                ; OUTPUT,
B68A 8580       BITA  #DOMASK       ; A 1 OR 0?
B68C 2705       BEQ   LOWOUT        ; EE_OUT BIT IS 0
B68E 1C0301     BSET  PCOFF,X,SDAMASK ; HIGH NEEDS TO BE SENT OUT

B691 2003       BRA   CONTOUT
B693          LOWOUT
B693 1D030      BCLR  PCOFF,X,SDAMASK ; SEND OUT A LOW
B696 1C0302     CONTOUT BSET  PCOFF,X,SCKMASK ; SET CLOCK BIT
B699 01         NOP                ; WAIT PROPER SCLK HI TIME

```

AN609

```
B69A 01          NOP          ;
B69B 1D0302     BCLR  PCOFF,X,SCKMASK ; CLR CLOCK BIT AND THEN
B69E 1D0301     BCLR  PCOFF,X,SDAMASK ; SET DATA BIT TO 0 FOR NEXT TX
B6A1 39          RTS
;*****
;*****
;
; This routine outputs 1 byte of data out the sdata pin
;
;*****

B6A2          TXBYTE
B6A2 C608          LDAB  #8          ; SET BIT COUNTER

B6A4 BDB682     TXBIT JSR  OUTBIT      ; SEND 1 BIT
B6A7 790100     ROL   TXBUFF        ; GET NEXT BIT READY TO XMIT
B6AA 5A          DECB
B6AB C100        CMPB  #00H          ; HAVE WE OUTPUT ALL 8 BITS
B6AD 26F5        BNE   TXBIT        ; NO, THEN SEND NEXT BIT

; GET ACK BIT AND TEST IF IT IS LOW. IF NOT, TURN ON ACK FAIL LED
B6AF BDB666     JSR  INBIT          ; RECEIVE ACK BIT
B6B2 B60101     LDAA  EE_IN        ;
B6B5 8501        BITA  #DIMASK      ; TEST IF INPUT IS 0 OR 1
B6B7 2703        BEQ  DONETX        ; IF ACK IS LOW, GO TO END OF TXBYTE
B6B9 1C0304     BSET  PCOFF,X,LEDMASK ; TURN ON ACK FAILED LED
B6BC 39          DONETXRTS
;*****

;*****
;
; This subroutine receives 1 byte from SEEPROM
;
;*****

B6BD          RXBYTE
B6BD C608          LDAB  #8          ; LOAD BIT COUNTER IN ACCB

B6BF 790103     RXBIT ROL   RXBUFF        ; GET READY TO RECEIVE NEXT BIT
B6C2 B60103     LDAA  RXBUFF        ; GUESS THAT INPUT BIT IS 0
B6C5 84FE        ANDA  #11111110B    ;
B6C7 B70103     STAA  RXBUFF        ; WRITE ACCA BACK TO RXBUFF

B6CA BDB666     JSR  INBIT          ; RECIEVES 1 BIT
B6CD B60101     LDAA  EE_IN        ; IS IN BIT A 1 OR A 0?
B6D0 8501        BITA  #DIMASK      ;
B6D2 2708        BEQ  CONTRX        ;π

B6D4 B60103     LDAA  RXBUFF        ; GET RXBUFF INPUT
B6D7 8A01        ORAA  #0000001B    ; SET INPUT BIT TO 1

B6D9 B70103     STAA  RXBUFF        ;

B6DC 5A          CONTRXDECB
B6DD C100        CMPB  #00H          ; HAVE WE OUTPUT ALL 8 BITS
B6DF 26DE        BNE   RXBIT        ; NO, THEN SEND NEXT BIT

B6E1 39          RTS
;*****

;*****
; This command tells the cross-assembler that code starts
; at the location of START
;*****

B600          END   START
```

Using the 24LC21 Dual Mode Serial EEPROM

*Author: Bruce Negley
Memory Products Division*

INTRODUCTION

The Microchip Technology Inc. 24LC21 is a 1K-bit (128 x 8) dual mode serial EEPROM that was developed primarily for use in computer monitors. This part was developed with inputs from several computer monitor manufacturers, in accordance with the VESA® (Video Electronics Standards Association) monitor committee. This committee has developed a serial communication protocol called Data Display Channel (DDC™) which was created to eliminate the need to change dip switches when configuring a new system or adding a new monitor or video card. The 24LC21 device is used in the monitor to store and transmit the EDID (extended display ID) table which contains all set-up parameters needed by the video card to operate with a particular monitor. With this system, the user can now plug any compatible monitor into any compatible graphics board and the graphics board will automatically know what type of monitor is being used and configure itself accordingly. This automatic configuration is the cornerstone for Microsoft®'s 'Plug and Play' capability being built into the new 'Windows 95™' release.

DEVICE OPERATION

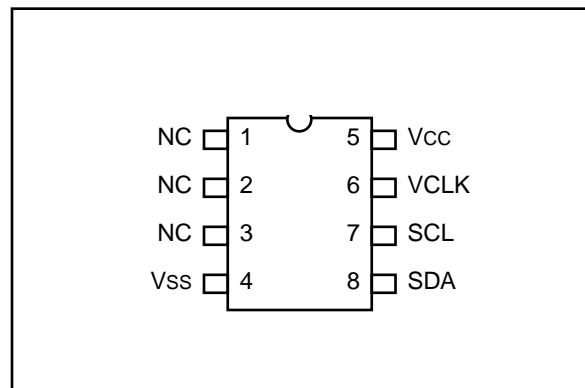
The 24LC21 can operate in two modes of operation. These two modes of operation are the transmit only mode and bi-directional mode. Upon power-up, the device will always be in the transmit-only mode. Transmit only mode is also referred to as DDC1 mode. The transmit only mode only allows the video card to read the contents of the 24LC21 in a sequential manner, one bit at a time. Writing to the device is not possible in transmit only mode.

The device will automatically transition to the bi-directional mode whenever a falling edge is seen on the SCL pin. Bi-directional mode is also referred to as DDC2 mode, and is implemented as the standard I²C™ protocol. This allows a controller to read and write specific addresses in the device like a standard I²C Serial EEPROM device. Once the device has transitioned to the bi-directional mode, there is no way to return to transmit only mode other than to reset (power-down) the device.

TRANSMIT ONLY MODE (DDC1)

The 24LC21 will always power-up in the transmit only mode. In this mode, the 24LC21 will output one bit of data at the SDA pin for every rising edge on the VCLK pin. The data will be transmitted in 8-bit words, with each word followed by a 9th null bit. This null bit will always be high. A timing diagram for transmit only mode is shown in Figure 1. As long as VCLK is present and no falling edges on SCL are received, the 24LC21 will repeatedly cycle through the entire memory array.

PACKAGE TYPE



Upon power-up, the device will not output valid data until it has been initialized. This initialization procedure (Figure 1) data will not be available until after the first 9 clocks are sent to the device. The exact memory location that the 24LC21 begins to transmit data is unknown at power-up, and the initialization procedure only initializes the device, not the starting address or bit location. In order to for a controller to determine what address is being read, a 'framing' or 'syncing' procedure must be executed by the video card.

AN610

A framing procedure involves looking for the header portion of the EDID table which is a byte of 00H followed by 6 bytes of FFH and another byte of 00H. A framing routine would continue to clock data from the 24LC21 until this unique header has been found. At this point, the current location in the EDID table has been determined and the controller has now synchronized itself with the device. Care must be

taken while using the device in the transmit only mode to prevent noise on the SCL pin, as a falling edge seen on this pin will immediately send the part into the bi-directional mode. In a DDC1-only monitor, SCL is not connected to the VGA connector, but must still be terminated to VCC through a pullup resistor.

FIGURE 1: TRANSMIT ONLY MODE

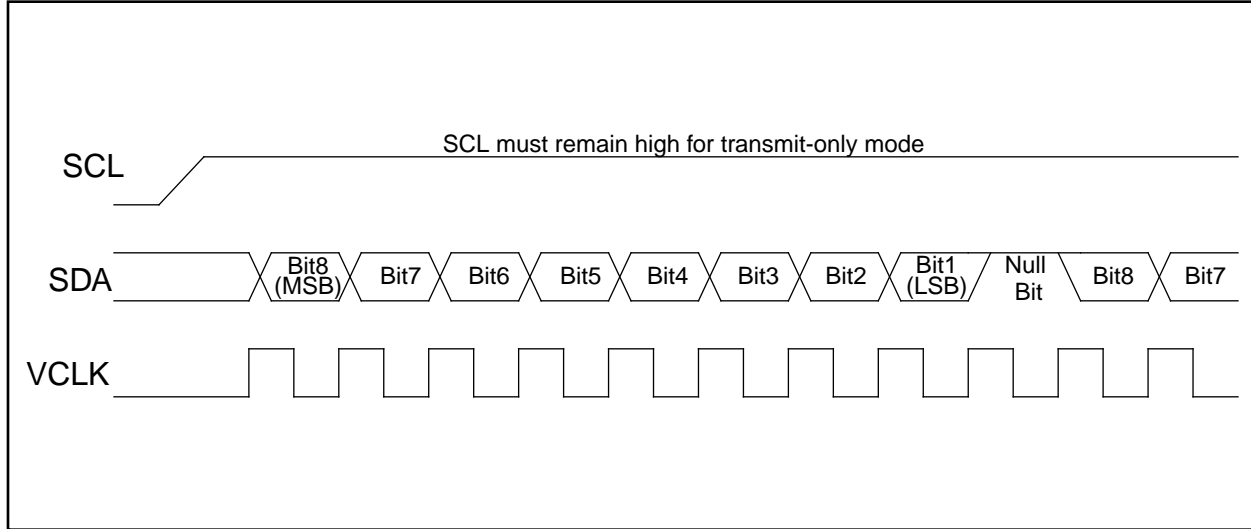
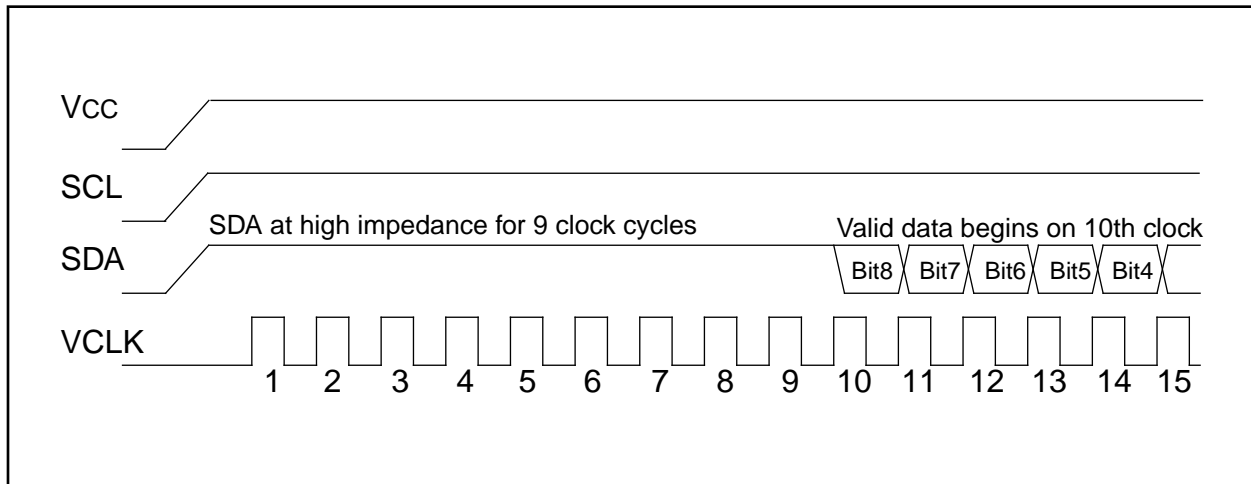


FIGURE 2: DEVICE INITIALIZATION FOR TRANSMIT-ONLY MODE



BI-DIRECTIONAL MODE (DDC2)

Bi-directional mode is essentially the standard I²C protocol and allows the controller to read and write to the device. The 24LC21 supports byte and page writes and byte and sequential reads in the bi-directional mode. This mode will be used primarily before the monitor leaves the factory to load the EDID table into the device, but it also provides a means of updating the table if necessary. It is also used for faster (up to 100 kHz) data transmission, or transmission of only specific requested data in a DDC2 system. (The I²C protocol allows the host to request data from a specific portion of the EDID table rather than waiting for the entire table.) When writing to the device, the VCLK pin must be held high while the write command is being loaded or the write will be aborted and no data will be written. Note that this is the opposite of the 24LC01B, where the WP pin must be held low for the device to be written.

EDID TABLE

The EDID table is the Extended Display ID table, specified by VESA, that will be stored in the 24LC21 and contains information about what type of display it is and the capabilities of the display. The basic EDID table consists of 128 bytes of data. A breakdown of the table is shown below in Table 1. A complete description of the table can be found in the VESA DDC Specification.

TABLE 1: EDID TABLE DESCRIPTION

Bytes	Description
8	Header
10	Vendor/Product Description
2	EDID Version/Revision
15	Basic Display Parameters/Features
19	Established/Standard Timings
72	Detailed Timing Descriptions (18 bytes each)
1	Extension Flag
1	Checksum

AN610

USING THE 24LC21 IN A SYSTEM

In order to use the 24LC21 in a monitor system, it must be programmed with a proper EDID table and then properly connected to the signals coming from the video controller card. The VESA committee has specified that the connections for DDC transmission can be part of the standard 15-pin VGA connector. A table of pinouts for this connector are shown in Table 2. Signals that pertain to the use of the 24LC21 are highlighted.

Programming of the 24LC21 can be accomplished via Microchip Technology's SEEVAL programming and evaluation system or by any final test system at the customer site which can communicate over the I²C bus.

TABLE 2: VGA CONNECTOR DESCRIPTION

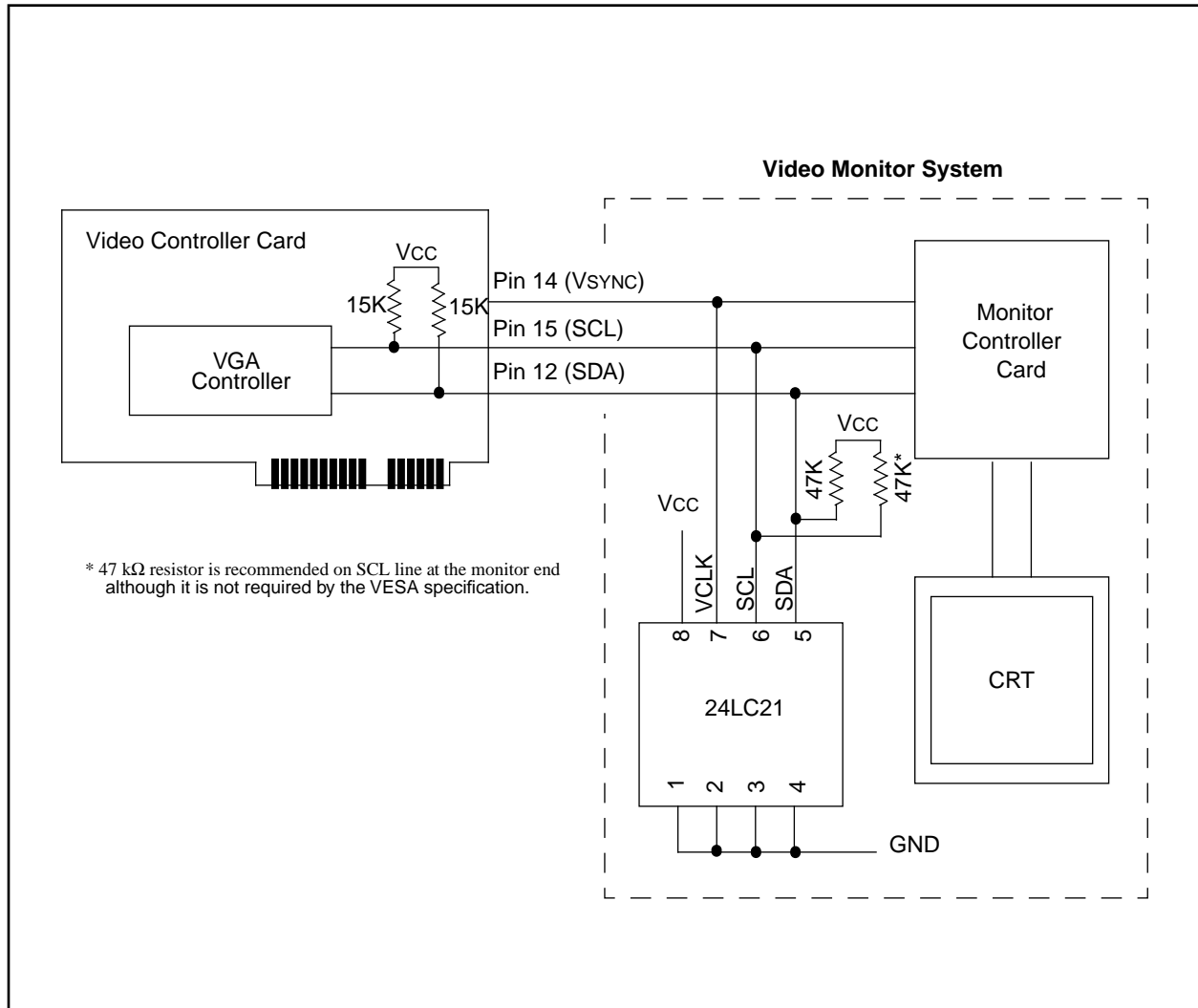
Pin	Standard VGA	DDC1 Host	DDC2 Host	DDC1.2 Display
1	Red Video	Red Video	Red Video	Red Video
2	Green Video	Green Video	Green Video	Green Video
3	Blue Video	Blue Video	Blue Video	Blue Video
4	Monitor ID Bit2	Monitor ID Bit2	Monitor ID Bit2	Return
5	Test (Ground)	Return	Return	Return
6	Red Video Return	Red Video Return	Red Video Return	Red Video Return
7	Green Video Return	Green Video Return	Green Video Return	Green Video Return
8	Blue Video Return	Blue Video Return	Blue Video Return	Blue Video Return
9	No Connection	+5V Supply (optional)	+5V Supply (optional)	+5V Supply (optional)
10	Sync Return	Sync Return	Sync Return	Sync Return
11	Monitor ID Bit0	Monitor ID Bit0	Monitor ID Bit0	Optional
12	Monitor ID Bit1	Data from Display (SDA)	Bi-directional Data (SDA)	Bi-directional Data (SDA)
13	Horizontal Sync	Horizontal Sync	Horizontal Sync	Horizontal Sync
14	Vertical Sync	Vertical Sync (VCLK)	Vertical Sync	Vertical Sync (VCLK)
15	Monitor ID Bit3	Monitor ID Bit3	Data Clock (SCL)	Data Clock (SCL)

SYSTEM CONFIGURATION

A typical system configuration is shown below. The DDC specification states that a 47 k Ω pull-up resistor is required on the SDA line at the monitor end. It also

states that a 15K pullup resistor is needed on both the SCL and SDA lines at the video controller end.

FIGURE 3: USE OF 24LC21 IN VIDEO SYSTEM



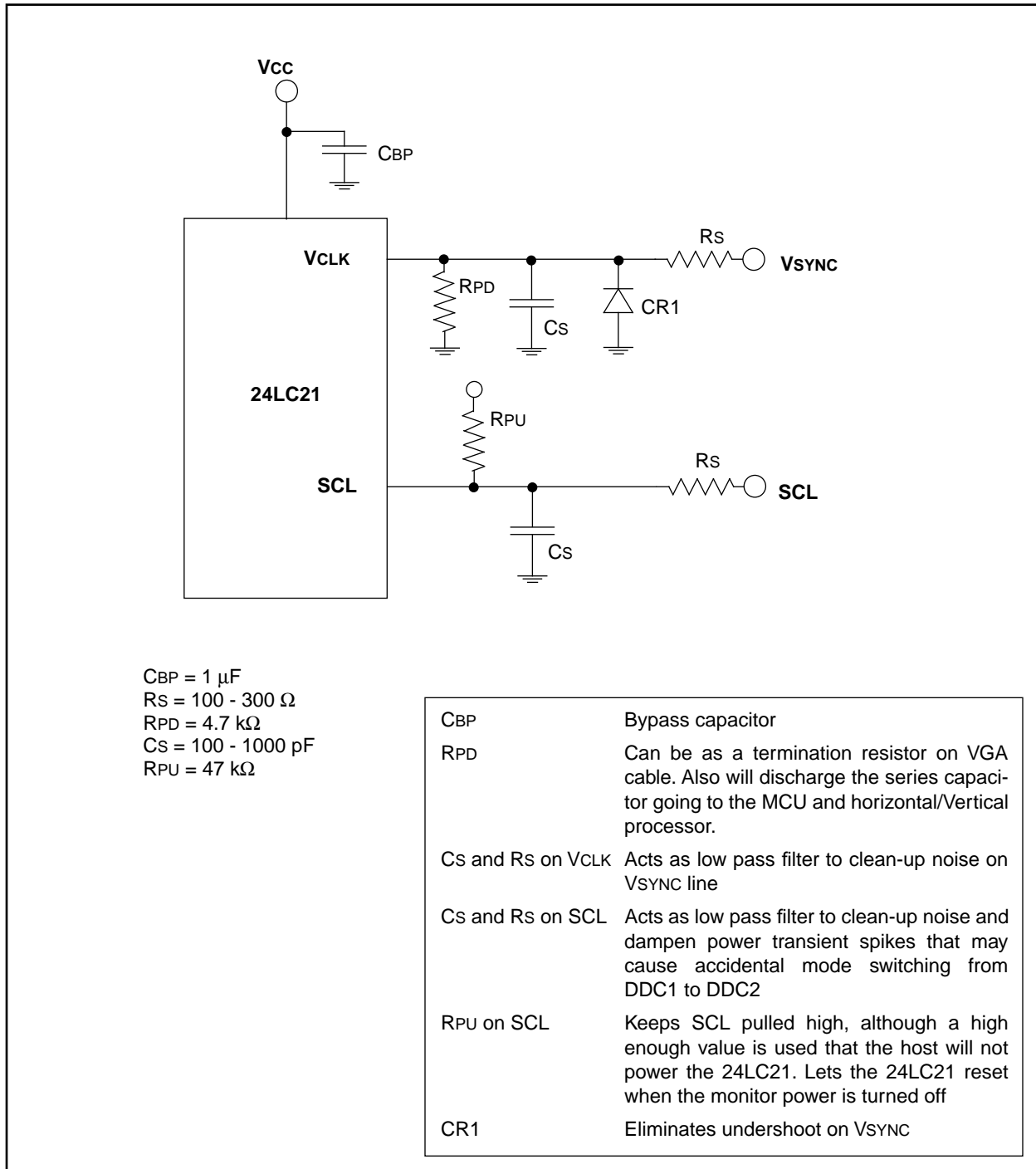
AN610

POTENTIAL PROBLEMS CAUSED BY NOISE IN A VIDEO SYSTEM

Because the typical application for the 24LC21 is in a computer monitor where electronic noise is prevalent, some precautions may need to be made in order for this device (or any other CMOS device) to work prop-

erly. The diagram below (Figure 4) shows a filter circuit that can be used to reduce the amount of noise seen by the device on the SCL and VCLK pins.

FIGURE 4: RECOMMENDED FILTER CIRCUIT FOR MONITOR APPLICATIONS



Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports

Author: Keith Pazul
Memory and ASSP Division

INTRODUCTION

Systems requiring embedded control are becoming more and more sophisticated, with microcontrollers required to control these systems increasing in complexity. Many microcontrollers today are being designed with built-in serial communication capability to be able to easily access other features that are not built in to the microcontroller itself. Devices like EEPROMs, A/Ds, D/As, LCDs etc. are all being built with a serial interface to reduce cost, size, pin count, and board area. There are many different serial interfaces on the market used to interface peripherals (I²C™, Microwire®, SPI, for example). One of the serial interfaces that is gaining in popularity is SPI (Serial Peripheral Interface). It is becoming more popular because of its communication speed, simultaneous full-duplex communication, and ease of programming.

Microchip PIC16C64/74 microcontrollers have a built-in serial port that can be configured as an SPI port. Currently, the Microchip Serial EEPROM product line does not support SPI interface Serial EEPROMs, however it is possible to use the 93 series devices on the SPI port. Any version of Microchip's 93 series devices can be communicated with via the SPI port of a PIC16C64/74. The code for this application note is written for a Microchip 93LC56/66, but talking to other 93 series devices can be accommodated with minor code changes. See the *Theory of Operation* section of this application note for more details on how this is accomplished. This code was verified by downloading to Microchip's PICMASTER™ in-circuit emulator (run at full speed with a 10 MHz crystal) and tested to make sure it writes (polling ready/busy pin to verify write cycle completion) and reads properly. A schematic (Figure 1) is included in this application note to describe exactly how the 93LC56/66 was connected to the PIC16C64/74.

The SPI interface was popularized by the Motorola 68HCXX microcontrollers. Microchip receives many requests for Motorola assembly code that uses the SPI port of the 68HC11 or 68HC05 to talk to Microchip serial EEPROMs. Because of this, Microchip has written 68HC11 assembly code to communicate with its 93 series devices via its SPI port. The program was downloaded to a 68HC11 evaluation board and tested

to make sure it writes (polling ready/busy pin to verify write cycle completion) and reads properly. A schematic (Figure 2) is included in this application note to describe exactly how the 93LC56/66 was connected to the microcontroller.

THEORY OF OPERATION

To use an SPI port to communicate with Microchip's 93 series Serial EEPROMs, the bytes to be output to the 93XXXX must be aligned such that the LSB of the address is the 8th bit (LSB) of a byte to be output. From there, the bits should fill the byte from right to left consecutively. If more than 8 bits are required, then two bytes will be required to be output. This same method will work for any 93 series device. A 93LC66 was chosen as the device to write this application note code for, so the following example will be for that particular device. The theory explained below will work for any 93 series device.

Since more than 8 bits are required to control a 93LC66, two consecutive bytes are required.

High Byte (where the start bit, op code bits, and address MSB reside)

| 0 | 0 | 0 | 0 | 0 | SB | OP1 | OP0 | A8 |

The High Byte is configured in the following format: SB is the start bit, OP1 is op code MSB, OP0 is op code LSB, and A8 is the 9th address bit that is required to address 512 bytes. The CS can be set before the byte is output because the leading 0's output to the 93xxxx prevent a start bit from being recognized by the 93xxxx until the first high bit is sent.

Low Byte (8 address LSBs)

| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |

The Low Byte contains A7-A0, which are the rest of the address bits required to access 512 bytes.

Data output from master MUST be set up on the falling edge of the clock so that it can be read from the 93XXXX on the next rising edge. Receiving data from the 93XXXX MUST also happen on the falling edge of the clock because the data is output from the 93XXXX on the rising edge of the clock. THIS REQUIRES THE CLOCK PHASE BIT OF THE SPI PORT TO BE OPPOSITE FOR RECEIVING THAN IT IS FOR TRANSMITTING. The clock phase needs to be toggled between 0 for transmitting data and 1 for receiving data. See source code for the exact spot where the clock phase bit needs to be changed.

AN613

FIGURE 1: PIC16C74 TO 93LC56/66 SCHEMATIC

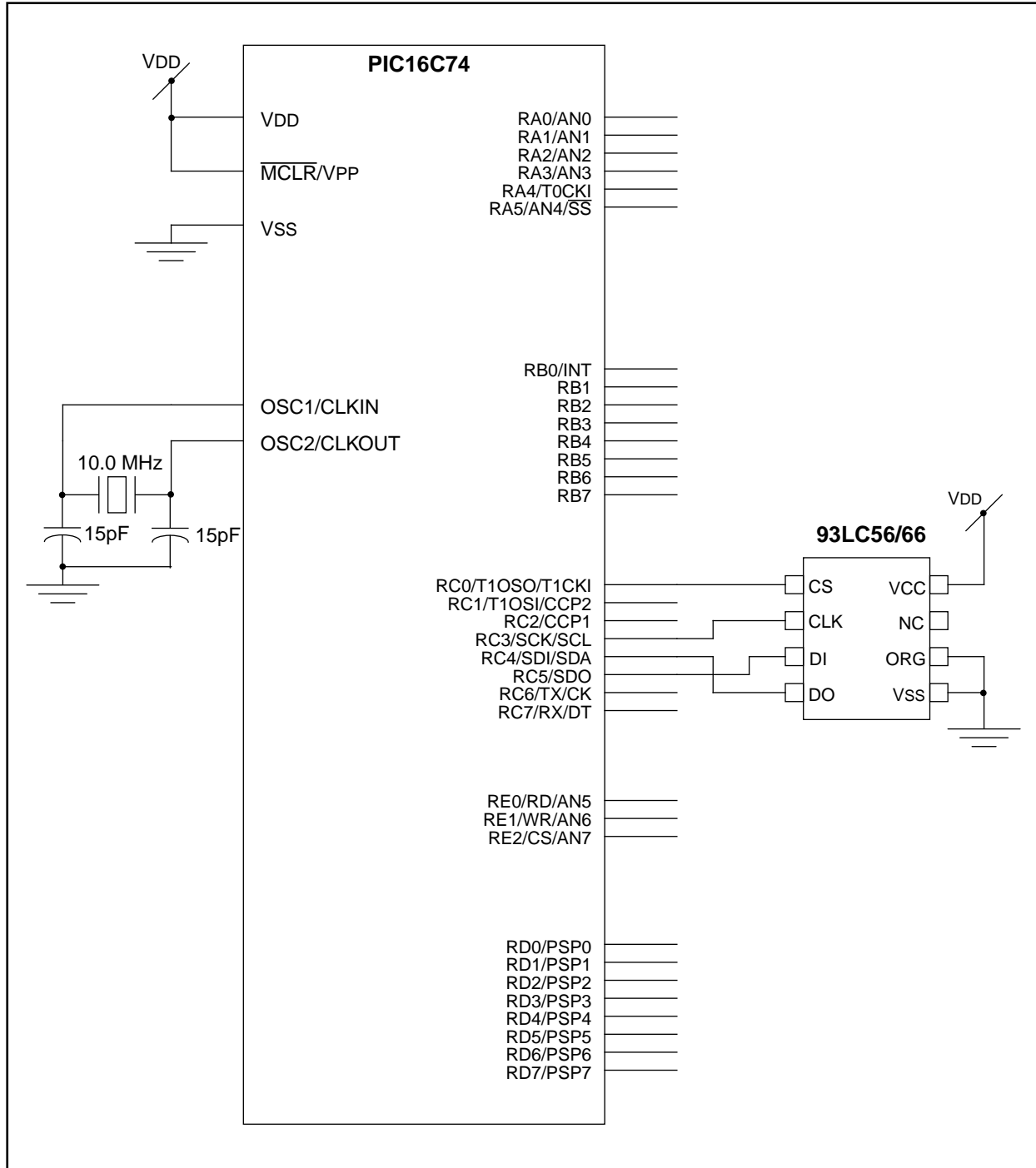
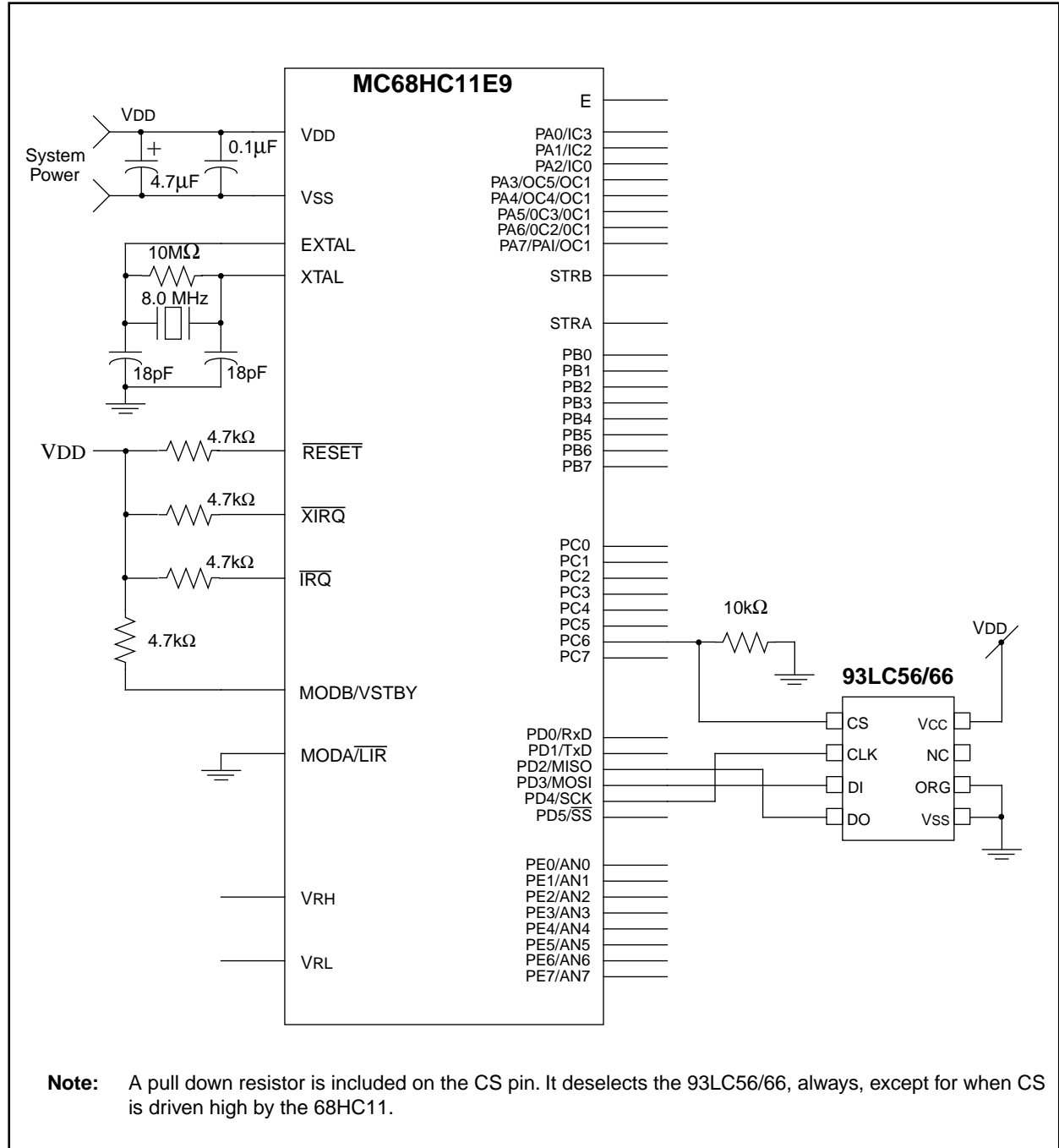


FIGURE 2: MOTOROLA 68HC11 TO MICROCHIP 93LC56/66 SCHEMATIC



AN613

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: PIC16C64/74 SOURCE CODE

```
*****
;
;
; To use the SPI port to communicate with 3-wire devices,
; the bytes to be output must be aligned such that the LSB of the
; address is the 8th bit (LSB) of a byte to be output. From there,
; the bits should fill the byte from right to left consecutively.
; This same method will work for any 93xxxx device. A 93LC66 was
; chosen as the device to write this application note code for,
; so the following example will be for that particular device.
; The theory explained below will work for any 93 series device.
;
; Since more than 8 bits are required to control a 93LC66,
; two consecutive bytes are required.
;
; High byte (where start bit, op code bits and address MSB reside)
;
; | 0 | 0 | 0 | 0 | SB | OP1 | OP0 | A8 |
;
; The High Byte is configured in the following format:
; SB is the start bit, OP1 is op code MSB, OP0 is op code LSB, and A8
; is the 9th address bit that is required to address 512 bytes. The CS
; can be set before the byte is output because the leading 0's output to
; the 93xxxx prevent a start bit from being recognized by the 93xxxx until
; the first high bit is sent.
;
; Low byte (8 address LSBs)
;
; | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
;
; The Low Byte contains A8-A0, which are address bits required
; to access 512 bytes.
;
; The chip select is set high before sending the first byte out because
; the 93LC66 will not recognize a start bit until both CS and DI are
; high on the rising edge of a clock.
;
; This code is written to use the 16C64/74 SPI port to communicate with a
; Microchip 93LC66. The only other I/O line required besides the SPI port
; pins is a chip select. The ORG pin will be grounded to set up the part
; in x8 mode.
;
; PIN DESCRIPTIONS:
;
; SCK      (serial clock)      port C, bit 3
; SDO      (serial data out)   port C, bit 5
; SDI      (serial data in)    port C, bit 4
; CS       (chip select)       port C, bit 0
;
; Transmits from the master MUST happen on the falling edge of the clock
; so they can be read by the 93xxxx on the next rising edge of the clock.
; Receiving from the 93xxx MUST also happen on the falling edge of the
; clock because on the rising edge of the clock, the 93xxxx outputs its bit
; on its DO pin. This requires the CKP bit in the SSPCON register to be set
; to 1 for transmitting data and CKP=0 for receiving data.
;
; This code was written by Keith Pazul on 10/5/94
;
*****

*****
;
; Ram Register Definitions
;
```



```

;*****
;
; received bytes from EEPROM memory locations 10h to 13h
; will be stored in RAM registers 20h to 23h.
rxdata    equ    24h
txdata    equ    25h
addr      equ    26h
loops     equ    27h
loops2    equ    28h
hibyte    equ    29h
lobyte    equ    2Ah
datbyt    equ    2Bh
;*****
;
;      Other Definitions
;
;*****
;
;*****
;      Bit Definitions
;*****
;
cs        equ    0
sdi       equ    4
;*****
include "c:\mpasm\include\p16cxx.inc" ; register map for PIC16CXX devices
org      0x000 ; Reset Vector
goto    Start
;
;*****
;
; This is the transmit/receive routine. The received bytes
; are don't cares until reading back from the array.
;
;*****
output    movwf    SSPBUF ; place data in buffer so it
; can be output
loop1     bsf      STATUS, RP0 ; specify bank 1
          btfss   SSPSTAT, BF ; has data been received (xmit done)?
          goto    loop1 ; not done yet, keep trying
          bcf      STATUS, RP0 ; specify bank 0
          movf    SSPBUF, W ; empty receive buffer, even if we
; don't need received data
          movwf   rxdata ; put received byte into location
          retlw   0 ; return from subroutine
;*****
;*****
;
;      EWEN routine
;
;*****
EWEN
          bcf      STATUS, RP0 ; need to set bank 0
          bsf      PORTC, cs ; set chip select line high
          movlw   b'00001001' ; start bit is bit 3, 00 is
; op code for EWEN
          call    output ;
          movlw   b'10000000' ; 1 req for EWEN, 0000000 are don't cares
; which is the 8 lsb address bits
          call    output;
          bcf      PORTC, cs ; bring chip select low to begin
; EEPROMs internal write cycle
          retlw   0 ; return from subroutine
; remember CS must be low for at least 250 nsec
;*****

```

AN613

```
;*****  
;  
; This routine outputs the two bytes required to send  
; the start bit, op code bits, and address bits  
;  
;*****  
WRITE  
    bcf     STATUS, RP0      ; need to set bank 0  
    bsf     PORTC, cs       ; set chip select line high  
    movf    hibyte, 0       ; put hibyte in w reg  
    call    output          ;  
    movf    FSR, 0          ; put addr pointed to by FSR into  
                                ; w reg  
    call    output          ;  
    movf    datbyt, 0       ; get ready to output data in datbyt  
    call    output          ;  
    bcf     PORTC, cs       ; bring chip select low to begin  
                                ; EEPROMs internal write cycle  
    incf    FSR, 1          ; point to next location to  
                                ; write to  
    retlw   0  
;*****  
;*****  
;  
; This is the module that reads one byte of data  
;  
;*****  
READ  
    bcf     STATUS, RP0      ; need to set bank 0  
    bsf     PORTC, cs       ; set chip select line high  
    bsf     SSPCON, CKP     ; make sure CKP is 1 to output  
                                ; next instruction and addr  
    movf    hibyte, 0       ; move data from hibyte to w  
    call    output          ;  
    movf    lobyte, 0       ; get ready to send next byte  
                                ; which is the 8 lsb address bits  
    call    output          ;  
;*****  
; This is where CKP bit is reset for receiving data.  
;*****  
    bcf     SSPCON, CKP     ; change clock polarity to 0  
    movlw   0x00           ; The byte xmitted here is a  
                                ; don't care  
    call    output          ;  
    bcf     PORTC, cs       ; bring chip select low to  
                                ; terminate read command  
    clrf    INDF            ; clr location pointed to by FSR  
    movf    rxdata, 0       ; move received data to w reg  
    movwf   INDF            ; put received data in location  
                                ; pointed to by FSR  
    incf    FSR, 1          ; point to next location to  
                                ; write to  
    incf    lobyte, 1       ; next addr to read from  
    retlw   0 ;  
;*****  
;*****  
;*****  
;*****  
Start  
    bcf     STATUS, RP0      ; need to set bank 0  
    clrf    PORTC           ; initialize port c  
    bsf     STATUS, RP0     ; need to set bank 1  
    movlw   0x10           ; all bits are outputs except SDI  
    movwf   TRISC           ; for SPI input  
    clrf    PIE1            ; disables all peripheral ints  
    clrf    INTCON          ; disables all interrupts
```

```

        bcf      STATUS, RP0      ; need to set bank 0
        clrf    SSPCON           ; clear SSP control register
        movlw   0x31             ; SPI master, clk/16, ckp=1
        movwf   SSPCON           ; SSPEN enabled
        call    EWEN             ; output EWEN for enabling writes
;*****
;   The next thing we will do is to write 0x5A to locations
;   10h through 13h.
;*****
        movlw   b'00001010'      ; start bit is bit 3, 01 is
                                ; op code for write
        movwf   hibernate        ; load into hibernate

        movlw   0x10             ; put beginning address in FSR
        movwf   FSR              ; for later use
        movlw   b'01011010'     ; load 0x5A as data to be sent out
        movwf   databyt         ;
wrnext   call    WRITE           ; call write subroutine
;*****
;   Ready/Busy poll to decide when write is complete
;   and the 93LC66 is available for writing the next
;   byte.
;*****
        nop                    ; cs must be low for > 250 ns
        bsf     PORTC, cs        ; and then be brought high
rbusy    btfss  PORTC, sdi      ; test ready/busy status
                                ; if 1, internal write is done
                                ; part still writing, stay in
                                ; loop
        goto    rbusy           ;
        bcf     PORTC, cs        ; bring cs back low
        btfss  FSR, 2           ; have we written all 4 locations?
        goto    wrnext         ; no, then write next byte
;*****
;   Now, lets read back 10h through 13h (non-sequentially) and
;   store it in ram locations 20h through 23h in the 16C74.
;   With the Picmaster, I can read those memory locations
;   and see that it was read in properly. This is how
;   I can quickly verify that the read function is working
;   properly.
;*****
        movlw   0x20             ; where in RAM to begin storing
        movwf   FSR              ; data read back from EEPROM
        movlw   0x10             ; addr of where to begin reading
        movwf   lobyte          ; EEPROM from
        movlw   b'00001100'     ; start bit is bit 3, 10 is
        movwf   hibernate        ; op code for read
rdnext   call    READ           ;
        btfss  FSR, 2           ; have we 20h thru 23?
        goto    rdnext         ; no, then read next location
;*****
;   While program is in limbo routine, it is possible
;   to halt the processor with Picmaster and look at
;   the data contained 16C74 RAM locations 20h through 23h.
;*****
limbo    nop
        goto    limbo
;
;
        end

```

AN613

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX B: MOTOROLA 68HC11 SOURCE CODE

```
*****
; 3-Wire byte write and byte read using SPI port (220 bytes)
;
; This program (hcllsp66.*) writes 4 bytes of $5A to a Microchip 93LC66
; using the SPI port of the Motorola 68HC11 microcontroller. Ready/busy
; polling is used to determine when the current byte is done writing and
; the next byte is ready to be written to the 93LC66. After the 4 bytes
; are written, they are read back and stored in RAM locations $100-$103.
; The evaluation board can be stopped after receiving the 4 bytes to
; view RAM locations and verify that what was written to the 93LC66 is
; what is read back.
;
; This code was written by Keith Pazul on 3/7/95.
;
; This is the way the bytes will be aligned to be sent to the 93LC56/66:
;
; First byte (where start bit, op codes reside)
;
; | 0 | 0 | 0 | 0 | SB | OP1 | OP0 | A8 |
;
; where SB is start bit, OP1 is op code msb, OP0 is op code lsb,
; and A8 is address msb.
;
; Next byte (address)
;
; | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
;
; where A8-A0 are address bits required to address 4k bits of memory.
;
; This code will work for any 93 series device. The only difference
; is that the number of address bits is adjusted and the start bit
; and op code bits are adjusted to follow directly after the address
; MSB.
;
; THE 93LC56 or 66 IS ASSUMED TO BE IN x8 MODE. IT REQUIRES
; THE USER TO TIE THE ORG PIN TO Vss.
;
; This program was written using a 68CH11EVBU evaluation board.
; This board has a monitor program in firmware of the 68HC11 which
; allows single stepping, register viewing, modifying, etc. Since this
; is the case, the program code will be loaded into the on-chip EEPROM.
; This EEPROM begins at $B600. The control registers are left to thier
; default location of $1000 and the RAM is left to its default location.
; RAM locations $48-$ff are used by the monitor program and are not
; available for program use. Therefore, the stack pointer is set a $47
; and will be able to use all of the RAM to $00, and the RAM variables
; begin at location $100 and go up from there. I cannot program the
; reset vector since it is ROM space (at $FFFE), so the way I run this
; program is to use the monitor program that comes with the evaluation
; board to set the program counter to the starting address of my user
; program ($B600) and begin from there. For users who do have access
; to the reset vector, the label of the beginning program (in my case, it
; is called START) should be loaded at location $FFFE. This program was
; not assembled using a Motorola assembler, but was assembled using
; Universal Cross-Assemblers Cross-32 Meta-Assembler. It has the
; ability to assemble just about any microcontroller code. There are
; certain commands that are unique to the cross-assembler. These
; commands will be commented differently than other comments to
; be recognizable. They will look like this:
;
;*****
```

```

; Special cross-assembler command(s)
;+++++

; I do not know the exact assembler commands required to accomplish
; assembly using a Motorola assembler.
;
; The crystal that comes with the evaluation board is 8 Mhz.
;
; The SPI port in on port D. The bits are defined as follows:
;
;         MISO      PORT D, PIN 2 (pin 22 on package)
;         MOSI      PORT D, PIN 3 (pin 23 on package)
;         SCK       PORT D, PIN 4 (pin 24 on package)
;         SS\      PORT D, PIN 5 (pin 25 on package)
;         CS        PORT C, PIN 6
;
; Note that only the CS pin resides on port C.
;*****

;+++++
CPU      "C:\WIN32\68HC11.TBL" ; LOAD TABLE
HOF      "MOT8"; Hex output is Motorola S-records
;+++++
;*****
; 68HC11 control register locations
;*****
REGBS    EQU      1000H          ; BEGINNING OF REGISTERS
RAMBS    EQU      100H          ; BEGINNING OF RAM VARIABLES
DDRC     EQU      REGBS+07H     ; DATA DIRECTION REG FOR PORT C
PORTC    EQU      REGBS+03H     ; PORT C DATA REGISTER
PCOFF    EQU      03H          ; OFFSET FROM CONTROL REG BEG.
PORTD    EQU      1008H        ; PORT D DATA REGISTER
DDRD     EQU      1009H        ; PORT D DATA DIRECTION REGISTER
SPCR     EQU      1028H        ; SPI CONTROL REGISTER
SPSR     EQU      1029H        ; SPI STATUS REGISTER
SPDR     EQU      102AH        ; SPE DATA REGISTER
;*****
;*****
; User defined constants
;*****
CSMASK   EQU      0100000B     ; BIT MASK FOR CHIP SELECT
SDIMASK  EQU      00000100B    ; BIT MASK FOR MISO PIN
;*****

LDS      #0047H                ; STACK POINTER BEGINS AT $47

ORG      100H                  ; RAM variables begin at 100h

;+++++
; DFS is a Universal Cross-Assembler directive that stands for define
; storage. 1 is for byte, 2 is for word, 4 is for long word. These are
; the user defined RAM variables.

RXARRAY  DFS      1 * {4}      ; 100H-103H
RXDATA   DFS      1            ; 104H
DATBYT   DFS      1            ; 105H
HIBYTE   DFS      1            ; 106H
LOBYTE   DFS      1            ; 107H
RBTEST   DFS      1            ; 108H
ADDOFF   DFS      1            ; 109H
;+++++
;*****
; Program code cannot be placed in ROM for eval board because

```

AN613

```
; eval board firmware is loaded in ROM. Program code will be
; loaded in EEPROM (which is 512 bytes and begins at B600h).
;

                ORG     0B600H
;*****
;
; This is the main portion of the code. It is where the reset
; vector should set program counter to.
;
;*****

START
    LDX     #REGBS           ; LOADS BEG OF REGISTERS INTO X

    BCLR    PCOFF,X,CSMASK   ; MASK SURE CS IS CLEARED

    LDAA    #10110000B       ; SETS UP BITS 0-3 AND 6 AS INPUTS,
    STAA    DDRC             ; BITS 4,5, AND 7 AS OUTPUTS

    CLR     PORTC            ; CLEAR ALL PORT C BITS

    LDAA    #11111011B       ; ALL BITS ARE OUTPUTS EXCEPT MISO
    STAA    DDRD             ;

    LDAA    #01010010B       ; SPIE=0,SPE=1,DWOM=0,MSTR=1,
    STAA    SPCR             ; CPOL=0,CPHA=0, CLK/16

    LDAA    #SDIMASK         ; STORE READY/BUSY MASK IN
    STAA    RBTEST           ; LOCATION RBTEST

    LDAA    #10H             ;
    STAA    LOBYTE           ; LOAD 8 LSB'S OF ADDRESS

    CLR     ADDR0FF          ; SET ADDRESS OFFSET TO 0 FOR
                            ; READ COMMANDS

    JSR     EWEN             ; SEND EWEN COMMAND

;*****
; Now lets write 5Ah out to address 10h
;*****

    LDAA    #00001010B       ; STRT BIT IS BIT 3, 01 IS
    STAA    HIBYTE           ; OP CODE FOR WRITE, LSB IS
                            ; ADDRESS

    LDAA    #01011010B       ; LOAD 0x5A IN DATA LOCATION
    STAA    DATBYT          ;

WRNEXT    JSR     WRITE       ; CALL WRITE ROUTINE

;*****
; ready/busy poll input pin to see when internal write
; cycle is complete.
;*****

    BSET    PCOFF,X,CSMASK   ; RAISE CS FOR READY/BUSY POLLING

RBUSY    LDAA    PORTD        ; INPUT PORT D
    ANDA    RBTEST           ; MASK OFF ALL BITS EXCEPT MISO
    BEQ     RBUSY           ; IF MISO IS LO, PART STILL BUSY,
                            ; ELSE WRITE IS COMPLETE

    BCLR    PCOFF,X,CSMASK   ; POLLING COMPLETE, END CHIP SELECT
```

```

        LDAA      LOBYTE          ; GET ADDRESS OFFSET
        BITA      #00000100B      ; ARE ALL BYTES WRITTEN YET?
        BEQ       WRNEXT          ; ALL BYTES HAVE NOT BEEN WRITTEN,
                                   ; WRITE NEXT BYTE

;*****
; Now lets read back the location of rxdata to see if we
; read back what we wrote to that location.
;*****

        LDAA      #00001100B      ; STRT BIT IS BIT 3, 10 IS
        STAA      HIBYTE          ; OP CODE FOR READ, LSB IS
                                   ; ADDRESS
        LDAA      #10H            ; RESET BEGINNING ADDRESS
        STAA      LOBYTE          ; FOR READ OPERATIONS

        LDY       #RAMBS          ; RX BUFF IS AT BEG OF RAM

RDNEXT   JSR      READ            ; CALL READ SUBROUTINE

        LDAA      LOBYTE          ; LOAD ACC WITH MASK THAT
                                   ; CHECKS TO SEE IF 4 BYTES
                                   ; HAVE BEEN WRITTEN
        BITA      #00000100B      ; CHECK TO SEE IF 4 BYTES HAVE
        BEQ       RDNEXT          ; BEEN WRITTEN. IF NOT, READ
                                   ; NEXT BYTE.

        LDAA      #10110000B      ; DISABLES CS TO PROHIBIT
        STAA      DDRC            ; POSSIBILITY OF INADVERTANT
                                   ; WRITES

LIMBO    NOP
        JMP       LIMBO

;*****
;
; This subroutine outputs the data stored in TXBUFF
;
;*****

OUTPUT
LOOP1    STAA      SPDR            ; MOVE DATA FROM ACC A TO SPDR
        LDAB      SPSR            ; SEE IF XFER COMPLETE FLAG SET
        BPL       LOOP1          ; IF NOT, LOOP UNTIL SET

        LDAA      SPDR            ; MOVES RECEIVED BYTE FROM DATA REG
        STAA      RXDATA          ; TO RECEIVE DATA LOCATION

        RTS

;*****

;*****
;
; EWEN routine
;
;*****
EWEN
        LDAA      #11110000B      ; SET CS FROM IN TO OUT. ALL
        STAA      DDRC            ; OTHER BITS ARE THE SAME

        BSET     PCOFF,X,CSMASK   ; DROP CS TO BEGIN COMMAND

        LDAA      #00001001B      ; STRT BIT IS BIT 3, 00 IS
                                   ; OP CODE FOR EWEN
        JSR      OUTPUT          ;

```

AN613

```
LDAA      #1000000B      ; 1 IS REQ FOR EWEN.  REST
                        ; ARE DON'T CARE BITS FOR
                        ; LOWER 8 ADDR BITS
JSR       OUTPUT        ;
BCLR      PCOFF,X,CSMASK ; SET CS TO COMPLETE EWEN

RTS
;*****
;*****
;
; WRITE routine
;*****
WRITE
BSET      PCOFF,X,CSMASK ; BEGIN WRITE COMMAND

LDAA      HIBYTE        ; GET DATA FROM HIBYTE TO OUTPUT
JSR       OUTPUT        ;

LDAA      LOBYTE        ; 8 LSBs OF ADDRESS POINTED TO
JSR       OUTPUT        ; BY ADDROFF OFFSET TO Y REG

LDAA      DATBYT        ; OUTPUT DATA BYTE
JSR       OUTPUT        ;

BCLR      PCOFF,X,CSMASK ; DROP CS TO BEGIN COMMAND

INC       LOBYTE        ; INC ADDRESS FOR NEXT LOCATION
                        ; TO WRITE TO

RTS
;*****
;*****
;
; READ routine
;*****
READ
LDAA      SPCR          ; READ IN SPI CONTROL REG
ANDA     #11111011B    ; SET CPHA TO 0 FOR CONTROL
STAA     SPCR          ; PORTION OF READ SEQUENCE

BSET      PCOFF,X,CSMASK ; DROP CS TO BEGIN COMMAND

LDAA      HIBYTE        ; OUTPUT HI BYTE FOR READ CMD
JSR       OUTPUT        ;

LDAA      LOBYTE        ; 8 LSBs OF ADDRESS
JSR       OUTPUT        ;

;*****
;*****
;*****
; To read a byte, a byte must be transmitted. Here we xmit a byte
; of don't care bits to receive a byte. WE MUST CHANGE THE CPHA
; FROM 0 TO 1 TO CHANGE BEFORE DATA IS RECEIVED. IF NOT, THEN THE
; MICRO WILL TRY TO RECEIVE AT THE SAME TIME THE 93XX66 WILL BE
; TRANSMITTING. THE SYMPTOM OF THIS NOT SET UP PROPERLY IS THAT
; THE DATA READ WILL BE SHIFTED RIGHT ONE BIT WITH MSB BEING 0.
;*****
;*****
;*****

LDAA      SPCR          ; READ IN SPI CONTROL REG
```



```
ORAA      #00000100B      ; SET CPHA TO 1
STAA      SPCR

JSR       OUTPUT          ; XMIT A BYTE OF DON'T CARE BITS
                          ; TO RECEIVE A BYTE

BCLR      PCOFF,X,CSMASK  ; CLR CS TO END COMMAND

LDAA      RXDATA          ; GET DATA RECIEVED IN RXDATA
STAA      0, Y            ; AND STORE IN LOCATION POINTED
                          ; TO BY Y INDEX REG + 0 OFFSET

INY
INC       LOBYTE          ; INC ADDR OF NEXT BYTE TO READ
                          ; FROM

RTS
;*****

;+++++
; This command tells the cross-assembler that code starts
; at the location of START

      END      START
;+++++
```

AN613

NOTES:

Interfacing the 8051 with 2-wire Serial EEPROMs

Author: Mike Rosenfield
Memory Products Division

INTERFACING MICROCHIP SERIAL EEPROMS TO THE INTEL 8051 FAMILY OF MICROCONTROLLERS

Many designers today are implementing embedded systems that require low cost non-volatile memory. Microchip has addressed this need with a full line of serial EEPROMs, in a variety of memory configurations, using the industry-standard 2- or 3-wire communication protocols. The theory and application of these protocols are addressed in detail in Microchip's application note AN536.

Microchip recognizes that its broad customer base uses a variety of micro-controllers; many firmware related questions have been asked concerning interfacing the 8051 family and its derivatives.

The purpose of this app note is to provide assembly language examples of 8051 code for the various serial EEPROMs available from Microchip. These routines are intended to provide the basic operating kernels for storing data to or retrieving data from a serial EEPROM.

All of the routines in this app note are available, as source code, for downloading from Microchip's BBS. Information on the BBS is available elsewhere in the Embedded Control Handbook. The file to download is AN61437.zip.

This app note covers all of Microchip's 2-wire serial devices. Note that some devices have features not supported in others, and therefore, some sections of the code presented here may not be applicable to a particular part. We have attempted to label those special sections to minimize confusion.

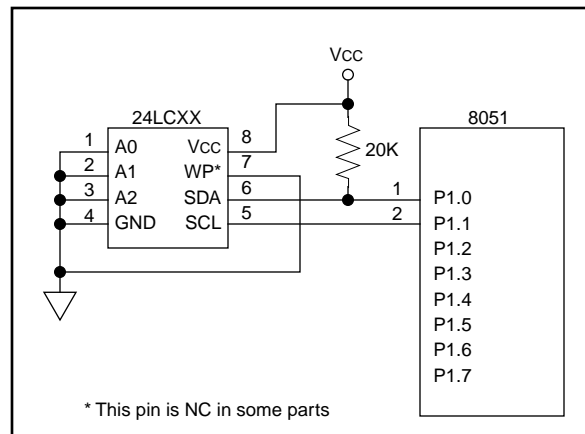
The code includes a simple loop-type shell to enable it to be executed (with an emulator) without the user having to write any other routines. The various address and data pointers must be set to the desired values by hand, before each execution cycle.

TIMING DATA

Clock and data timing is accomplished by software. There are two sets of timing specifications: 100 kHz and 400 kHz. Assuming a 12 MHz 8051 clock, extra NOP's have been added to slow timing down to 100 kHz. See Note 1 in the listing. If a 16 MHz clock is used, additional NOP's are required for 100 kHz operation. See Note 2 in the listing. For 400 kHz operation, the NOP's labelled Note 1 or Note 2 are not needed. If not needed, NOP's may be left out.

Below is the connection diagram used for this app note. Do not forget the pull-up resistor!

WIRING DIAGRAM



Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: SOURCE CODE

05/31/95

```
MCS-51 MACRO ASSEMBLER      2WIRE
DOS 6.0 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN 2WIRE.OBJ
ASSEMBLER INVOKED BY: C:\ICE5100\ASM51\ASM51.EXE 2WIRE.TXT
```

```
LOC  OBJ      LINE      SOURCE
1      $ PAGELENGTH(46) PAGEWIDTH(132) DEBUG NOPAGING XREF
2      ;REGISTER ASSIGNMENTS:
3      ;
4      ;R1      DATA OR DATA POINTER
5      ;R2      LOOP COUNTER REGISTER
6      ;R3      ADDRESS, HI BYTE
7      ;R4      ADDRESS, LOW BYTE
8      ;R5
9      ;R6      BYTE COUNT FOR PAGE OPERATIONS
10
11     ;PIN ASSIGNMENTS:
12     ;Port 1 bit 0 is data
13     ;Port 1 bit 1 is clock
14     ;
15     ;       These routines assume chip address = 0
16     ;
17     ;       The oscillator frequency assumed for this app note is 12 MHz.
18     ;
19     ;       These routines use software timing loops. They may have to be
20     ;       adjusted if a different oscillator frequency is used.
21     ;
22     ;NOTE 1 These NOP'S added for timing delays only on 'C' parts, OR 'LC' parts
23     ;       where Vcc is less than 4.5 V. and the oscillator frequency is 12 MHz.
24     ;       This allows a bit rate of 100kHz.
25     ;NOTE 2 Use these NOP's with a 16 MHz oscillator and 100kHz bit rate.
26     ;       For 400kHz bit rate, the NOP's in Note 1 and Note 2 are not required.
27     ;
28     ; The EEPROM will be busy after a write cycle is initiated (by a stop condition)
29     ; for between 1ms to 10 ms per page (or per byte if a byte write). This app note
30     ; assumes the user will program appropriate wait times after a write, or check
31     ; for Busy status. A subroutine is provided to check the Busy Status.
32     ;
```

```

33 ;RAM DEFINITIONS
34 ;
35 ORG 30H
36 BYTSTR: DS 20H ;STORAGE FOR READ DATA
37 ;
38 ;CONSTANTS -- REDEFINE AS NECESSARY
39 ;
40 WTCMD EQU 10100000B ;WRITE DATA COMMAND Note 3
41 RDCMD EQU 10100001B ;READ DATA COMMAND Note 3
42 RDEND EQU 01000000B ;READ HIGH-ENDURANCE BLOCK NUMBER COMMAND
43 ADDRH EQU 0
44 ADDR L EQU 0
45 DTA EQU 55H
46 BYTCNT EQU 8
47 ;
48 ;Note3 Some chip or byte address bits are embedded in the control byte. Refer to
49 ; the data sheet for exact configuration, which varies from part to part.
50 ;
51 ;*****
52 ; This section contains test loop routines. They form a simple operating shell to
53 ; allow the 2-wire interface code to be tested in a stand-alone mode. Using an
54 ; emulator, change "NONE" to one of the four listed
55 ; routines to test that function. The address and data constants can also be set as
56 ; desired.
57 ; If using a 24XX32 or 24XX65, change the called routines by adding 'L' to the end
58 ; of the name. This is required because these parts use TWO address bytes. The 'L'
59 ; routines send out the extra address byte.
60 ;*****
61 ORG 0
62 JMP START
63
64 START: MOV P1,#0FFH ;INIT PORT 1
65 CALL NONE ;TEST LOOP INSERT PROPER ADDRESS HERE
66 JMP START
67
68 NONE: RET
69 ;*
70 ;* WRITE ONE BYTE TO EEPROM
71 ;* The Address Pointer is the address in the EEPROM. The byte to be sent to the
72 ;* EEPROM is stored in the constant 'DTA'
73 ;*
74 TESTWR: MOV R3,#ADDRH ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY
75 MOV R4,#ADDRL ;LOAD ADDRESS POINTER FOR ALL DEVICES
76 MOV R1,#DTA ;LOAD DATA BYTE
77 CALL BYTEW ;CALL BYTE WRITE ROUTINE
78 RET
79
0030
0030
00A0
00A1
0040
0000
0000
0055
0008
0000
0000 020003
0003 7590FF
0006 12000B
0009 80F8
000B 22
000C 7B00
000E 7C00
0010 7955
0012 120037
0015 22

```

```

80 ;
81 ; * WRITE A BLOCK OF DATA TO EEPROM
82 ; * The address pointer is the address in EEPROM where data will start. The byte
83 ; * pointer is the starting address of RAM containing the block of data to be sent.
84 ; * The byte count indicates how many bytes to send to the EEPROM.
85 ; * The number of bytes that can be sent before a STOP command is issued is
86 ; * dependant on EEPROM type. Refer to the data book for specific values.
87 ;
88 BLKWR:  MOV  R3,#ADDRH      ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY
89         MOV  R4,#ADDRL      ;LOAD ADDRESS POINTER FOR ALL DEVICES
90         MOV  R1,#BYTSTR     ;LOAD BYTE POINTER
91         MOV  R6,#BYTCNT     ;LOAD BYTE COUNT
92         CALL PAGEW         ;CALL PAGE WRITE ROUTINE
93         RET
94
95 ;
96 ; * READ ONE BYTE FROM EEPROM
97 ; * The address pointer is the address of the desired byte in EEPROM.
98 ; * The byte will be returned in R1.
99 ;
100 TESTRD: MOV  R3,#ADDRH      ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY
101         MOV  R4,#ADDRL      ;LOAD ADDRESS POINTER FOR ALL DEVICES
102         CALL BYTERD         ;CALL BYTE READ ROUTINE.
103         MOV  R1,A          ;SAVE THE BYTE
104         RET
105
106 ;
107 ; * READ A BLOCK FROM EEPROM
108 ; * The address pointer is the starting address of the desired data block in EEPROM.
109 ; * The data pointer is the starting address in RAM where data will be stored.
110 ; * The byte count indicates how many bytes should be read.
111 ; * The entire EEPROM may be read with one READ command this way.
112 ;
113 BLOKRD: MOV  R3,#ADDRH      ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY
114         MOV  R4,#ADDRL      ;LOAD ADDRESS POINTER FOR ALL DEVICES
115         MOV  R1,#BYTSTR     ;LOAD DATA POINTER
116         MOV  R6,#BYTCNT     ;LOAD BYTE COUNT
117         CALL BLKRD         ;CALL BLOCK READ ROUTINE
118         RET
119
120 ; *END OF TEST LOOP
121 ; *****
122
123 ; *****
124 ; This routine writes a byte of data to EEPROM
125 ; The EEPROM address is assumed to be in R4. See NOTE 3.
126

```

```

127 ; The DATA to be written is assumed to be in R1
128 ; *****
129 BYTEM: MOV A,#WTCMD ;LOAD WRITE COMMAND *****
130 CALL OUTS ;SEND IT
131 MOV A,R4 ;GET LOW BYTE ADDRESS
132 CALL OUT ;SEND IT
133 MOV A,R1 ;GET DATA
134 CALL OUT ;SEND IT
135 CALL STOP ;SEND STOP CONDITION
136 RET
137
138 ; *****
139 ; THIS ROUTINE WRITES A PAGE OF DATA TO EEPROM *****
140 ; The EEPROM start address is assumed to be in R4. See NOTE 3.
141 ; The DATA pointer is in R1
142 ; The BYTE count is in R6
143 ; The number of bytes that can be transferred depends on the EEPROM used.
144 ; *****
145 PAGEW: MOV A,#WTCMD ;LOAD WRITE COMMAND *****
146 CALL OUTS ;SEND IT
147 MOV A,R4 ;GET LOW BYTE ADDRESS
148 CALL OUT ;SEND IT
149 MOV A,@R1 ;GET DATA
150 CALL OUT ;SEND IT
151 INC R1 ;INCREMENT DATA POINTER
152 DJNZ R6,BTLP ;LOOP TILL DONE
153 CALL STOP ;SEND STOP CONDITION
154 RET
155
156 ; *****
157 ; THIS ROUTINE READS A BLOCK OF DATA FROM EEPROM AT A SPECIFIED ADDRESS *****
158 ; EEPROM address in R4. See NOTE 3.
159 ; Stores data at RAM location pointed to by R1
160 ; Byte count specified in R6. May be 1 to 256 bytes
161 ; *****
162 BLKRD: MOV A,#WTCMD ;LOAD WRITE COMMAND TO SEND ADDRESS *****
163 CALL OUTS ;SEND IT
164 MOV A,R4 ;GET LOW BYTE ADDRESS
165 CALL OUT ;SEND IT
166 MOV A,#RDCMD ;LOAD READ COMMAND
167 CALL OUTS ;SEND IT
168 BRDLP: CALL IN ;READ DATA
169 MOV @R1,A ;STORE DATA
170 INC R1 ;INCREMENT DATA POINTER
171 DJNZ R6,AKLP ;DECREMENT LOOP COUNTER
172 CALL STOP ;IF DONE, ISSUE STOP CONDITION
173
0037 74A0
0039 120120
003C EC
003D 120158
0040 E9
0041 120158
0044 120197
0047 22

0048 74A0
004A 120120
004D EC
004E 120158
0051 E7
0052 120158
0055 09
0056 DEF9
0058 120197
005B 22

005C 74A0
005E 120120
0061 EC
0062 120158
0065 74A1
0067 120120
006A 12017E
006D F7
006E 09
006F DE04
0071 120197

```

```

0074 22      RET                ;DONE, EXIT ROUTINE
174
175
0075 C290    CLR      P1.0        ;NOT DONE, ISSUE ACK
176
0077 D291    SETB     P1.1        ;NOTE 1
177
0079 00      NOP
178
007A 00      NOP
179
007B 00      NOP
180
007C 00      NOP
181
007D 00      NOP
182
007E C291    CLR      P1.1        ;CONTINUE WITH READS
183
0080 80E8    JMP      BRDLP
184
185
*****
186
187 ; THIS ROUTINE READS A BYTE OF DATA FROM THE EEPROM
188 ; The EEPROM address is in R4. See NOTE 3.
189 ; Returns the data byte in R1
190 ; *****
191 BYTERD: MOV   A,#WTCMD      ;LOAD WRITE COMMAND TO SEND ADDRESS
192          CALL OUTS         ;SEND IT
193          MOV   A,R4        ;GET LOW BYTE ADDRESS
194          CALL OUT          ;SEND IT
195          CALL CREAD       ;GET DATA BYTE
196          RET
197
*****
198
199 ; THIS ROUTINE READS A BYTE OF DATA FROM EEPROM
200 ; From EEPROM current address pointer.
201 ; Returns the data byte in R1
202 ; *****
203 CREAD:  MOV   A,#RDCMD     ;LOAD READ COMMAND
204          CALL OUTS         ;SEND IT
205          CALL IN           ;READ DATA
206          MOV   R1,A        ;STORE DATA
207          CALL STOP        ;SEND STOP CONDITION
208          RET
209
*****
210
211 ; The next four routines are used with the 24XX32 or 24XX65 only. These parts
212 ; require two address bytes, and these routines send the second byte out.
213 ; Other than this, these routines are the same as the previous four.
214 ; *****
215 ; THIS ROUTINE READS A BLOCK OF DATA FROM EEPROM AT A SPECIFIED ADDRESS
216 ; This routine is for the 24LC32 or 24LC64
217 ; EEPROM address in R3:R4
218 ; Stores data at RAM location pointed to by R1
219 ; Byte count specified in R6. May be 1 to 256 bytes
220 ; *****

```



```

009C 74A0      BLKRD1: MOV     A,#WTCMD      ;LOAD WRITE COMMAND TO SEND ADDRESS
009E 120120    CALL    OUTS                ;SEND IT
00A1 EB       MOV     A,R3                ;GET HI BYTE ADDRESS
00A2 120158    CALL    OUT                 ;SEND IT
00A5 EC       MOV     A,R4                ;GET LOW BYTE ADDRESS
00A6 120158    CALL    OUT                 ;SEND IT
00A9 74A1     MOV     A,#RDCMD           ;LOAD READ COMMAND
00AB 120120    CALL    OUTS                ;SEND IT
00AE 80EA     JMP     BRDLP              ;CONTINUE WITH DATA READ
230
231 ;*****
232 ; This routine writes a byte of data to EEPROM
233 ; This routine is for the 24LC32 or 24LC64
234 ; The EEPROM address is assumed to be in R3:R4
235 ; The DATA to be written is assumed to be in R1
236 ;*****
00B0 74A0      BYTEWL: MOV     A,#WTCMD      ;LOAD WRITE COMMAND
00B2 120120    CALL    OUTS                ;SEND IT
00B5 EB       MOV     A,R3                ;GET HI BYTE ADDRESS
00B6 120158    CALL    OUT                 ;SEND IT
00B9 EC       MOV     A,R4                ;GET LOW BYTE ADDRESS
00BA 120158    CALL    OUT                 ;SEND IT
00BD E9       MOV     A,R1                ;GET DATA
00BE 120158    CALL    OUT                 ;SEND IT
00C1 120197    CALL    STOP                ;SEND STOP CONDITION
00C4 22       RET
246
247
248 ;*****
249 ; THIS ROUTINE WRITES A PAGE OF DATA TO EEPROM
250 ; This routine is for the 24LC32 or 24LC64
251 ; The EEPROM start address is assumed to be in R3:R4
252 ; The DATA pointer is in R1
253 ; The BYTE count is in R6
254 ; The number of bytes that can be transferred depends on the EEPROM in use.
255 ;*****
00C5 74A0      PAGEWL: MOV     A,#WTCMD      ;LOAD WRITE COMMAND
00C7 120120    CALL    OUTS                ;SEND IT
00CA EB       MOV     A,R3                ;GET HI BYTE ADDRESS
00CB 120158    CALL    OUT                 ;SEND IT
00CE EC       MOV     A,R4                ;GET LOW BYTE ADDRESS
00CF 120158    CALL    OUT                 ;SEND IT
00D2 E7       MOV     A,@R1              ;GET DATA
00D3 120158    CALL    OUT                 ;SEND IT
00D6 09       INC     R1                  ;INCREMENT DATA POINTER
00D7 DEF9     DJNZ   R6,BTLP            ;LOOP TILL DONE
00D9 120197    CALL    STOP                ;SEND STOP CONDITION
00DC 22       RET
267

```

```

268 ; *****
269 ; THIS ROUTINE READS A BYTE OF DATA FROM THE EEPROM *****
270 ; This routine is for the 24LC32 or 24LC64
271 ; The EEPROM address is in R3:R4
272 ; Returns the data byte in R1
273 ; *****
274 ; *****
275 BYTERDL: MOV A,#WTCMD ;LOAD WRITE COMMAND TO SEND ADDRESS
276 CALL OUTS ;SEND IT
277 MOV A,R3 ;GET HI BYTE ADDRESS
278 CALL OUT ;SEND IT
279 MOV A,R4 ;GET LOW BYTE ADDRESS
280 CALL OUT ;SEND IT
281 CALL CREAD ;GET DATA BYTE
282 RET
283
284 ;
285 ; SUBROUTINES
286 ;
287 ; *****
288 ; This routine tests for WRITE DONE condition *****
289 ; by testing for an ACK.
290 ; This routine can be run as soon as a STOP condition
291 ; has been generated after the last data byte has been
292 ; sent to the EEPROM. No ACK will be returned until
293 ; the EEPROM is done with the write operation.
294 ; *****
295 ACKTST: MOV A,#WTCMD ;LOAD WRITE COMMAND TO SEND ADDRESS
296 MOV R2,#8 ;LOOP COUNT -- EQUAL TO BIT COUNT
297 CLR P1.0 ;START CONDITION -- DATA = 0
298 NOP ;NOTE 1
299 NOP ;NOTE 2
300 NOP
301 NOP
302 NOP
303 CLR P1.1 ;CLOCK = 0
304 AKTLP: RLC A ;SHIFT BIT
305 JNC AKTLS
306 SETB P1.0 ;DATA = 1
307 JMP AKT1L
308 AKTLS: CLR P1.0 ;CONTINUE
309 AKTLL: SETB P1.1 ;DATA = 0
310 NOP ;CLOCK HI
311 NOP ;NOTE 1
312 NOP
313 NOP
314 NOP ;NOTE 2
315 NOP
00DD 74A0
00DF 120120
00E2 EB
00E3 120158
00E6 EC
00E7 120158
00EA 118F
00EC 22
00ED 74A0
00EF 7A08
00F1 C290
00F3 00
00F4 00
00F5 00
00F6 00
00F7 00
00F8 C291
00FA 33
00FB 5005
00FD D290
00FF 020104
0102 C290
0104 D291
0106 00
0107 00
0108 00
0109 00
010A 00

```

```

010B C291      315      CLR      P1.1      ;CLOCK LOW
010D DAEB      316      DJNZ     R2,AKTLP    ;DECREMENT COUNTER
010F D290      317      SETB     P1.0      ;TURN PIN INTO INPUT
0111 00        318      NOP          ;NOTE 1
0112 00        319      NOP          ;NOTE 2
0113 D291      320      SETB     P1.1      ;CLOCK ACK
0115 00        321      NOP          ;NOTE 1
0116 00        322      NOP          ;NOTE 2
0117 00        323      NOP          ;NOTE 2
0118 00        324      NOP          ;NOTE 2
0119 00        325      NOP          ;NOTE 2
011A 309002    326      JNB      P1.0,EXIT    ;EXIT IF NO ACK (WRITE NOT DONE)
011D C291      327      CLR      P1.1      ;SET DONE FLAG
011F 22        328      ;
329      EXIT:  RET
330
331
332
333      ;*****
334      ; THIS ROUTINE SENDS OUT CONTENTS OF THE ACCUMULATOR
335      ; to the EEPROM and includes START condition. Refer to the data sheets
336      ;for discussion of START and STOP conditions.
337      ;*****
338
0120 7A08      339      MOV      R2,#8      ;LOOP COUNT -- EQUAL TO BIT COUNT
0122 D290      340      SETB     P1.0      ;INSURE DATA IS HI
0124 D291      341      SETB     P1.1      ;INSURE CLOCK IS HI
0126 00        342      NOP          ;NOTE 1
0127 00        343      NOP          ;NOTE 2
0128 00        344      NOP          ;NOTE 2
0129 00        345      NOP          ;NOTE 2
012A 00        346      NOP          ;NOTE 2
012B C290      347      CLR      P1.0      ;START CONDITION -- DATA = 0
012D 00        348      NOP          ;NOTE 1
012E 00        349      NOP          ;NOTE 2
012F 00        350      NOP          ;NOTE 2
0130 00        351      NOP          ;NOTE 2
0131 00        352      NOP          ;NOTE 2
0132 C291      353      CLR      P1.1      ;CLOCK = 0
0134 33        354      RLC      A          ;SHIFT BIT
0135 5005      355      JNC      BITLS      ;DATA = 1
0137 D290      356      SETB     P1.0      ;CONTINUE
0139 02013E    357      JMP      OTSL1      ;DATA = 0
013C C290      358      CLR      P1.0      ;CLOCK HI
013E D291      359      SETB     P1.1      ;NOTE 1
0140 00        360      NOP          ;NOTE 1
0141 00        361      NOP

```

```

0142 00      NOP
0143 00      NOP
0144 00      NOP
0145 C291    CLR P1.1
0147 DAEB    DJNZ R2,OTSLP
0149 D290    SETB P1.0
014B 00      NOP
014C 00      NOP
014D 00      NOP
014E D291    SETB P1.1
0150 00      NOP
0151 00      NOP
0152 00      NOP
0153 00      NOP
0154 00      NOP
0155 C291    CLR P1.1
0157 22      RET
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
0158 7A08    OUT: MOV R2,#8
015A 33      OTLP: RLC A
015B 5005    JNC BITL
015D D290    SETB P1.0
015F 020164  JMP OTL1
0162 C290    CLR P1.0
0164 D291    SETB P1.1
0166 00      NOP
0167 00      NOP
0168 00      NOP
0169 00      NOP
016A 00      NOP
016B C291    CLR P1.1
016D DAEB    DJNZ R2,OTLP
016F D290    SETB P1.0
0171 00      NOP
0172 00      NOP
0173 00      NOP
0174 D291    SETB P1.1
0176 00      NOP
0177 00      NOP
0178 00      NOP
0179 00      NOP
017A 00      NOP

*****
; THIS ROUTINE SENDS OUT CONTENTS OF ACCUMULATOR TO EEPROM
; without sending a START condition.
*****
0158 7A08    OUT: MOV R2,#8
015A 33      OTLP: RLC A
015B 5005    JNC BITL
015D D290    SETB P1.0
015F 020164  JMP OTL1
0162 C290    CLR P1.0
0164 D291    SETB P1.1
0166 00      NOP
0167 00      NOP
0168 00      NOP
0169 00      NOP
016A 00      NOP
016B C291    CLR P1.1
016D DAEB    DJNZ R2,OTLP
016F D290    SETB P1.0
0171 00      NOP
0172 00      NOP
0173 00      NOP
0174 D291    SETB P1.1
0176 00      NOP
0177 00      NOP
0178 00      NOP
0179 00      NOP
017A 00      NOP

*****
; LOOP COUNT -- EQUAL TO BIT COUNT
; SHIFT BIT
; DATA = 1
; CONTINUE
; DATA = 0
; CLOCK HI
; NOTE 1
; NOTE 2
; NOTE 2
; CLOCK LOW
; DECREMENT COUNTER
; TURN PIN INTO INPUT
; NOTE 1
; NOTE 2
; CLOCK ACK
; NOTE 1
; NOTE 2
; NOTE 2

```

```

017B C291          CLR    P1.1
017D 22          RET

409
410
411
412 *****
413 ; THIS ROUTINE READS IN A BYTE FROM THE EEPROM
414 ; and stores it in the accumulator
415 *****
416 *****
417 IN:          MOV    R2,#8          ;LOOP COUNT
418          SETB   P1.0          ;SET DATA BIT HIGH FOR INPUT
419          INLP:  CLR    P1.1          ;CLOCK LOW
420          NOP
421          NOP
422          NOP
423          NOP
424          NOP
425          NOP
426          SETB   P1.1          ;CLOCK HIGH
427          CLR    C              ;CLEAR CARRY
428          JNB   P1.0,INL1       ;JUMP IF DATA =0
429          CPL    C              ;SET CARRY IF DATA =1
430          INL1: RLC    A          ;ROTATE DATA INTO ACCUMULATOR
431          DJNZ  R2,INLP         ;DECREMENT COUNTER
432          CLR    P1.1          ;CLOCK LOW
433          RET
434
435
436          STOP:  CLR    P1.0          ;STOP CONDITION SET DATA LOW
437          NOP
438          NOP
439          NOP
440          NOP
441          NOP
442          SETB   P1.1          ;SET CLOCK HI
443          NOP
444          NOP
445          NOP
446          NOP
447          NOP
448          SETB   P1.0          ;SET DATA HI
449          RET
450
451 *****
452 ;These routines contain special commands only for the 24LC65 SMART SERIAL EEPROM
453 ; *
454 ; SET SECURE BLOCK
455 ; ASSUMES START BLOCK 0 & BLOCK LENGTH OF 1

```

```

456 ; The numbers are implicit in the commands. Refer to the data sheet for details.
457 ; *
458 SETSEC: MOV R3,#80H ;LOAD COMMAND AND STARTING BLOCK NUMBER
459 MOV R4,#0
460 MOV R1,#81H ;SET COMMAND FOR NUMBER OF BLOCKS TO SECURE
461 CALL BYTEW ;EXECUTE
462 RET
463
464 ; *
465 ; READ SECURE BLOCK NUMBER(S)
466 ; RETURNS BLOCK NUMBER IN R1 AND BLOCK COUNT IN R2
467 ; (UPPER NIBBLES WILL BE 1'S)
468 ; *
469
470 RDSEC: MOV A,#WTCMD ;LOAD WRITE COMMAND TO SEND ADDRESS
471 CALL OUTS ;SEND IT
472 MOV A,#80H ;LOAD COMMAND
473 CALL OUT ;SEND IT
474 MOV A,#0 ;LOAD COMMAND
475 CALL OUT ;SEND IT
476 MOV A,#0C0H ;LOAD COMMAND
477 CALL OUT ;SEND IT
478 CALL IN ;READ STARTING BLOCK NUMBER
479 MOV R1,A ;STORE IT
480 NOP ;NOTE 1
481 NOP
482 NOP
483 NOP
484 NOP ;NOTE 2
485 CLR P1.0 ;ISSUE ACK
486 SETB P1.1
487 NOP ;NOTE 1
488 NOP
489 NOP
490 NOP ;NOTE 2
491 NOP
492 CLR P1.1
493 CALL IN ;READ NUMBER OF BLOCKS
494 MOV R2,A ;STORE IT
495 CALL STOP ;SEND STOP CONDITION
496 RET
497
498 ; *
499 ; SET HIGH-ENDURANCE BLOCK NUMBER
500 ; ASSUMES BLOCK 0
501 ; *
502

```

```

01DA 7B80          SETHI:  MOV   R3,#80H      ;LOAD COMMAND AND BLOCK NUMBER
01DC 7C00          MOV   R4,#0
01DE 7900          MOV   R1,#0      ;SET DATA = 0
01E0 1137          CALL  BYTEW     ;EXECUTE
01E2 22           RET
507
508
509
510          ; *
511          ; READ HIGH-ENDURANCE BLOCK NUMBER
512          ; RETURNS BLOCK NUMBER IN R1 (UPPER NIBBLE WILL BE 1'S)
513          ; *
01E3 7B80          READHI: MOV   R3,#80H      ;LOAD COMMAND
01E5 7C00          MOV   R4,#0
01E7 1201EB        CALL  HIEND     ;EXECUTE
01EA 22           RET
515
516
517
518          HIEND:  MOV   A,#WTCMD  ;LOAD WRITE COMMAND TO SEND ADDRESS
519          CALL  OUTS     ;SEND IT
520          MOV   A,R3      ;GET HI BYTE ADDRESS
521          CALL  OUT      ;SEND IT
522          MOV   A,R4      ;GET LOW BYTE ADDRESS
523          CALL  OUT      ;SEND IT
524          MOV   A,#RDEND  ;LOAD READ COMMAND
525          CALL  OUT      ;SEND IT
526          CALL  IN       ;READ DATA
527          MOV   R1,A      ;STORE DATA
528          CALL  STOP     ;SEND STOP CONDITION
529          RET
530
531          ;END of 24LC65 Routines
532          ; *****
533          END

```

ASSEMBLY COMPLETE, NO ERRORS FOUND

AN614

NOTES:

SECTION 5

QUALITY & RELIABILITY APPLICATION NOTES

AN595	Improving the Susceptibility of an Application to ESD	5-1
AN598	Plastic Packaging and the Effects of Surface Mount Soldering Techniques	5-19
AN603	Continuous Improvement	5-27

NOTES:

Improving the Susceptibility of an Application to ESD

Author: David Wilkie
Reliability Engineering

INDUCED LATCH-UP

All semiconductor devices are sensitive to electrostatic discharge (ESD) damage to varying degrees. This is true whether they are soldered to a PC board in an application, or whether they are unattached in the shipping or application assembly process. Good handling techniques such as groundstraps, static free work stations and ionizers can reduce the risk of static build up during assembly. Often more attention is paid to reducing ESD during assembly than is paid to reducing ESD risk during the lifetime of the application.

When a device is installed in an application, it is still susceptible to damage due to ESD. This can take on a different form when the application is powered up and running. If power is supplied to a CMOS device such as a memory product or a microcontroller when an ESD event occurs, the device can be triggered into a "latch-up" condition. This is a high current mode where internal circuitry can be disturbed into making a short circuit (or a circuit with very low resistance) between power (Vcc) and ground (Vss) on a device. This

condition is self-sustaining; it does not require subsequent ESD events to continue the latch-up condition.

This short circuit will tend to reduce the voltage level on the application (particularly if the application is battery powered) and will do a great deal of damage to the device which has latched-up. The only way to halt this condition is to remove power from the device.

Microchip uses careful design practices to reduce the susceptibility of all products (microcontrollers or memories) to ESD events. However, the protection level varies from pin to pin, reflecting the different functions of each pin. Certain types of pins (notably supply pins) are much more susceptible to latch-up caused by ESD pulses than other pins. This is due to the different design and layout considerations that reduce the effectiveness of ESD protection.

There is a great deal that the system designer can do to improve (by up to an order of magnitude) the level of protection of a device from latch-up inducing ESD events. This tutorial is intended as a guide for helping designers choose protection. This type of protection is application dependent, so consideration should be made of the type of environment that the application, or the device, will be in.

FIGURE 1: CIRCUIT DIAGRAM FOR ESD-INDUCED LATCH-UP

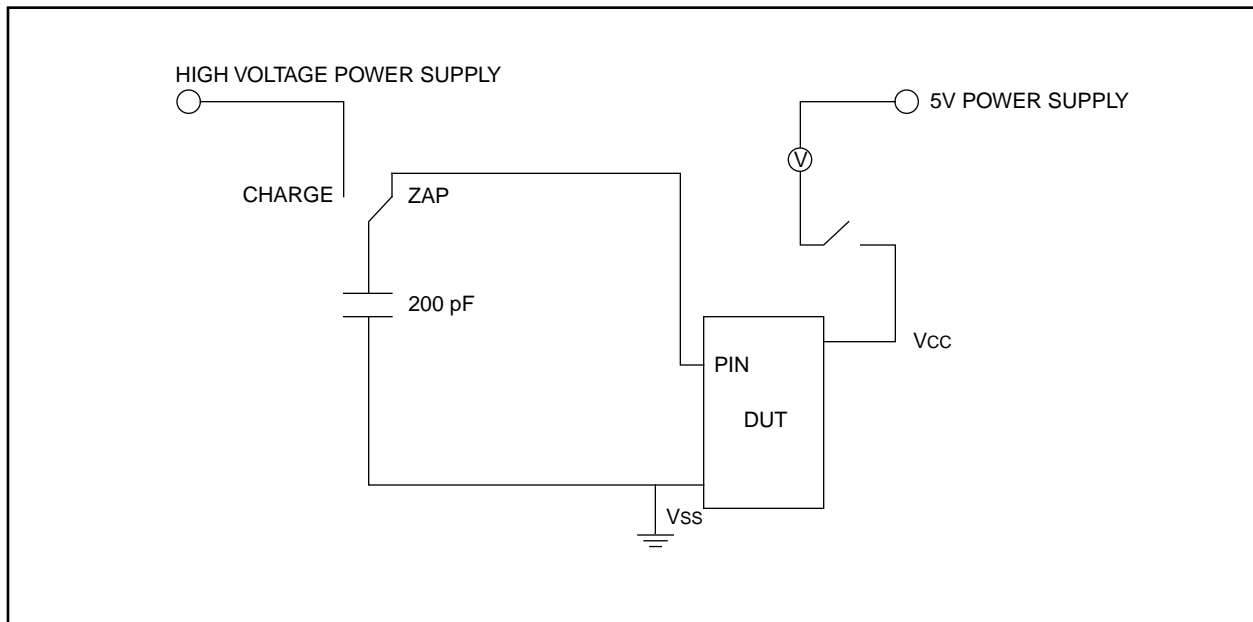
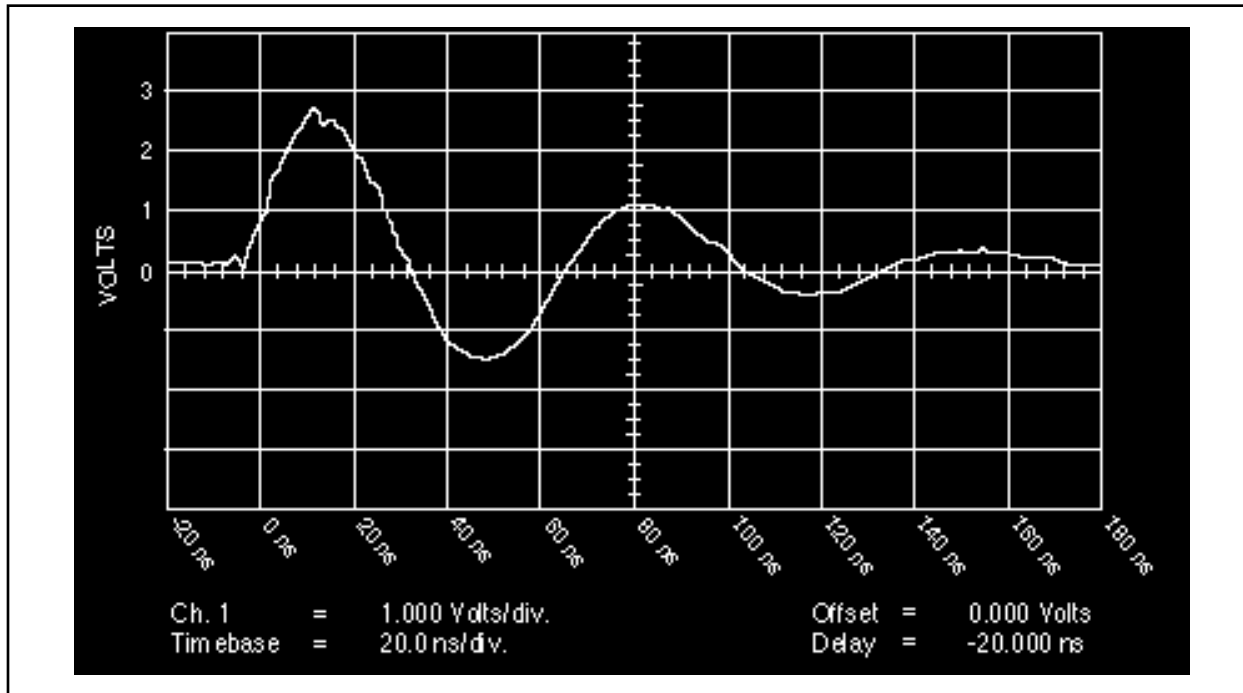


FIGURE 2: WAVEFORM USED TO SIMULATE ESD EVENT



ELECTROSTATIC DISCHARGE

Electrostatic discharges can come from a variety of sources. The traditional ESD pulse is caused by a body at very high potential coming into contact, or near contact, with a grounded object. This could be a human body, a piece of electrical equipment, or even a piece of furniture.

In a dry environment, where static dissipation is low, a human body can develop tens of thousands of volts of potential. Almost everyone has experienced the shock of walking across a new carpet and touching a door handle. An audible snap can be heard when the potential difference between the person and the door handle is around 5,000V.

Any piece of equipment which has metal components moving against other metal components can develop a charge. An automated device handler will generally have a metal tray where devices move around, with pins in occasional contact with the tray. Static build up can reach hundreds of volts and can damage the devices in the same way as an ungrounded human handler can.

Incorrectly placed ionizers, meant to improve static dissipation, can build large potentials on office or laboratory furniture. Any person touching such a piece of furniture might feel a shock. Any devices being placed on a table with a large potential can suffer damage.

All these situations can be avoided by careful handling procedures. Groundstraps for manual handling of devices is essential, as are grounded tables and work surfaces with anti-static surfaces such as metal or specially designed plastic mats. Equipment can be carefully grounded wherever the potential exists for static build up.

ELECTROSTATIC DISCHARGE IN A POWERED APPLICATION

Once the device has been mounted to a board or installed in an application, most types of ESD events will no longer occur. For example, unless the PC board is out in the open, it is very unlikely to be touched by the user, and so a direct ESD pulse will not be a concern.

However, there are several sources of indirect ESD pulses, or noise pulses which are very similar in nature and magnitude to ESD pulses. ESD pulses can induce currents in nearby wiring. So, for example, if the user creates an ESD event on the casing of an application, the magnetic field of the pulse can induce currents inside the casing, in the wiring of an application. This electromagnetic interference can occasionally be seen in other ways, such as a noisy TV picture when a vacuum cleaner is being used, or a "click" heard on the radio when a light switch is turned on. This type of event is often called Radio Frequency Interference (RFI) or Electromagnetic Interference (EMI).

Often the application itself can induce noise spikes during operation. If one component in that application is a high speed, high current transistor or other type of

switch, the sudden change in current can induce a noise spike which could be seen by that component, or other components in the system. This switching noise is endemic in systems with metal wires, and can not be removed completely. Large magnitude pulses of switching noise can induce latchup on sensitive CMOS devices.

The effects of switching noise or EMI can be catastrophic to an application with sensitive components. Even components which are not particularly sensitive or already have some built-in protection (such as Microchip Technology Inc.'s products) can still be latched-up by noise pulses if they are of a large enough magnitude.

PROTECTING AGAINST ESD PULSES

Basic protection can be provided by a simple decoupling capacitor, placed as close as possible to the power and ground pins of components. Each component should have its own capacitor; simply decoupling the whole application at the supply points will not be sufficient if a component in the application is producing noise.

If the value of the capacitor is chosen to match the device, then non-supply pins can also benefit from a decoupling capacitor between power and ground. A single capacitor can not filter out all the frequencies associated with a noise spike, but it can still offer very effective and low cost protection.

We used a simple test circuit to accurately model an ESD-induced noise spike. The test circuit is shown in Figure 1. A 200 pF capacitor was charged to various voltages and then switched to discharge directly into the device being tested. The waveform is a high frequency (≈ 14 MHz) and short period (≈ 100 ns) decaying oscillation. The resulting waveform, measured through a Tektronix CT-1 current probe with a 10X attenuator, is shown in Figure 2. The testing was conducted at 25°C.

The period of the oscillation is governed by the relationship:

$$\omega = \sqrt{\frac{1}{LC} - \frac{R^2}{4L^2}}$$

where L, C, and R are the inductance, capacitance, and resistance of the oscillating circuit, and ω is the frequency of the oscillation. Since R and L are very small, and C is 200pF, the oscillation period is very fast.

Two devices were tested. A typical serial EEPROM, the 24LC04B, was tested on all function and supply pins. The results, shown in graphical form (Figure 3 through Figure 7) show that, even with a capacitor placed between power (Vcc) and ground (Vss) other pins, such as SDA can have increased protection. From the figures it is clear that the best all-round protection can be obtained from a 10,000 pF capacitor.

Particular applications may be different. For example, the designer may know that there is a better chance for a noise spike on the SDA signal, and so may want to use a 1,000 pF capacitor to improve the protection level of SDA.

A PIC16C54 microcontroller in XT mode was tested on all functional and supply pins. The results are shown in Figure 8 through Figure 17. From these results, it is clear that the particular application environment will be much more important in determining which capacitor value to use. Some pins, such as \overline{MCLR} , do not respond at all to a decoupling capacitor. Others, such as the RA ports, respond strongly to a capacitor. Note that for the RA and RB graphs, the pin with the lowest latch-up threshold was used. All other pins are higher.

It is important for the designer to be aware of the potential problems associated with noise in a powered application. A combination of using sound design techniques to reduce causes of noise and an awareness of the protection levels of the components being used can make the problems of ESD manageable.

AN595

FIGURE 3: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON V_{SS}. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.

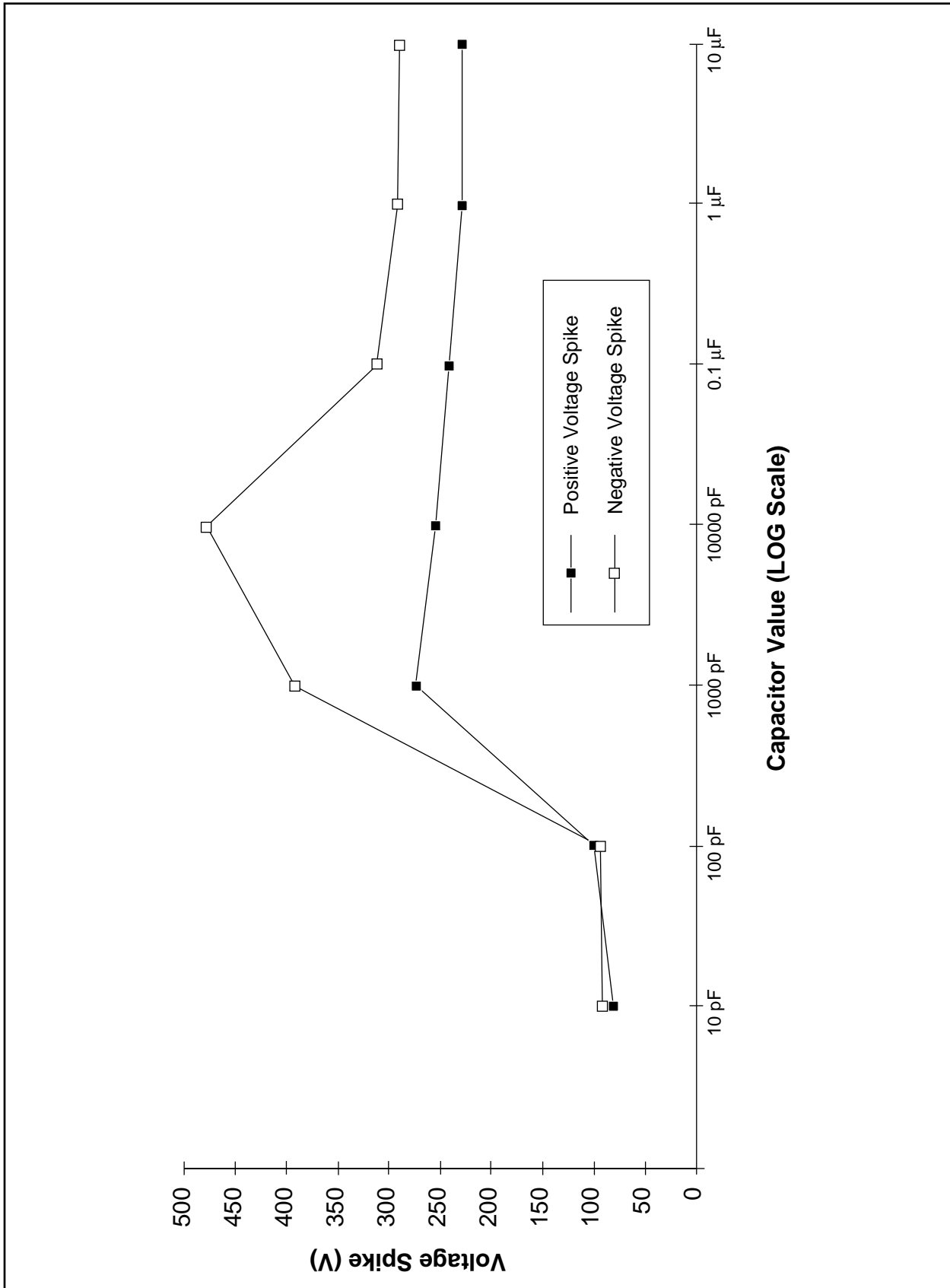
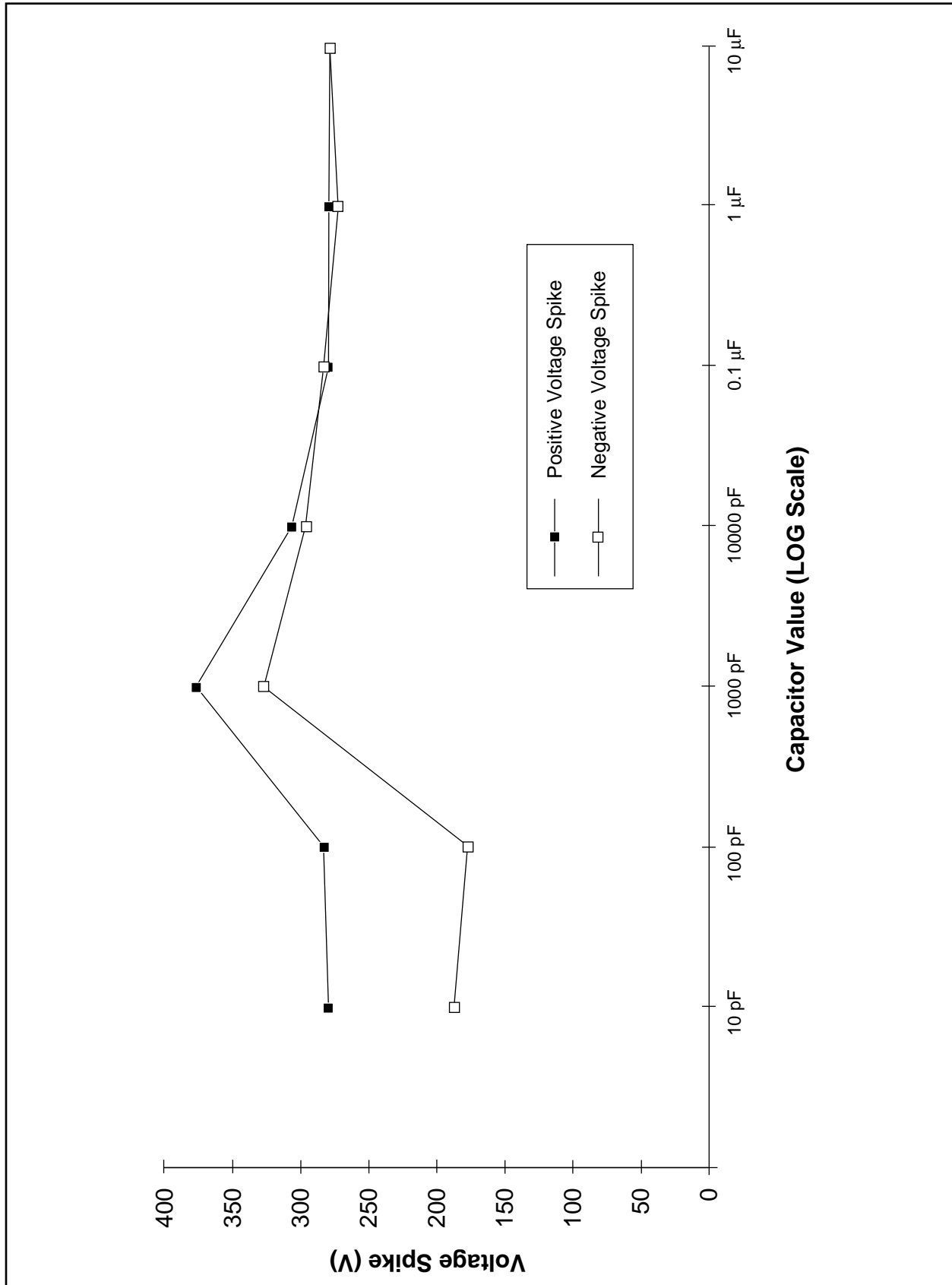


FIGURE 4: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON SDA. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.



AN595

FIGURE 5: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON SCL. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.

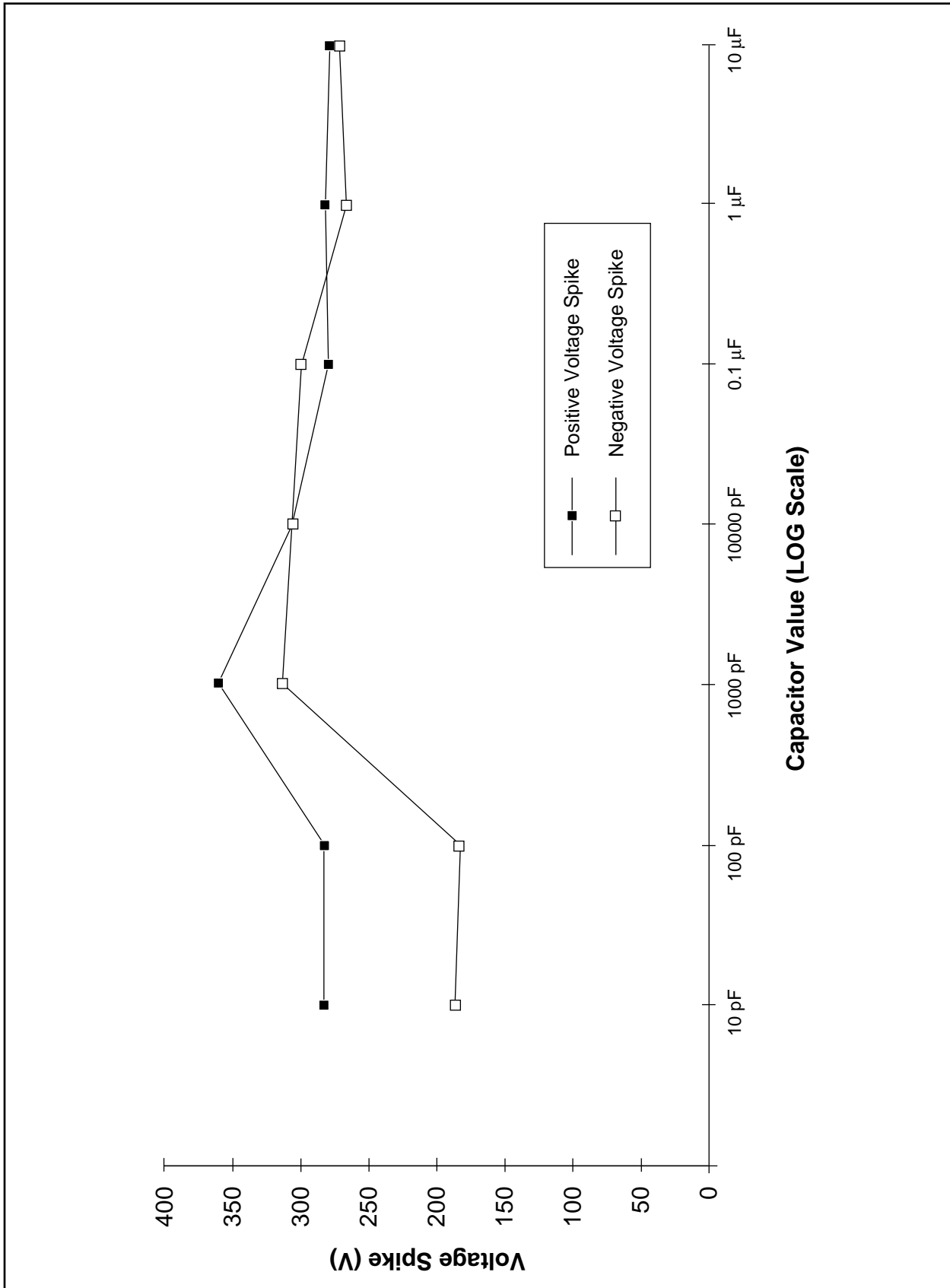
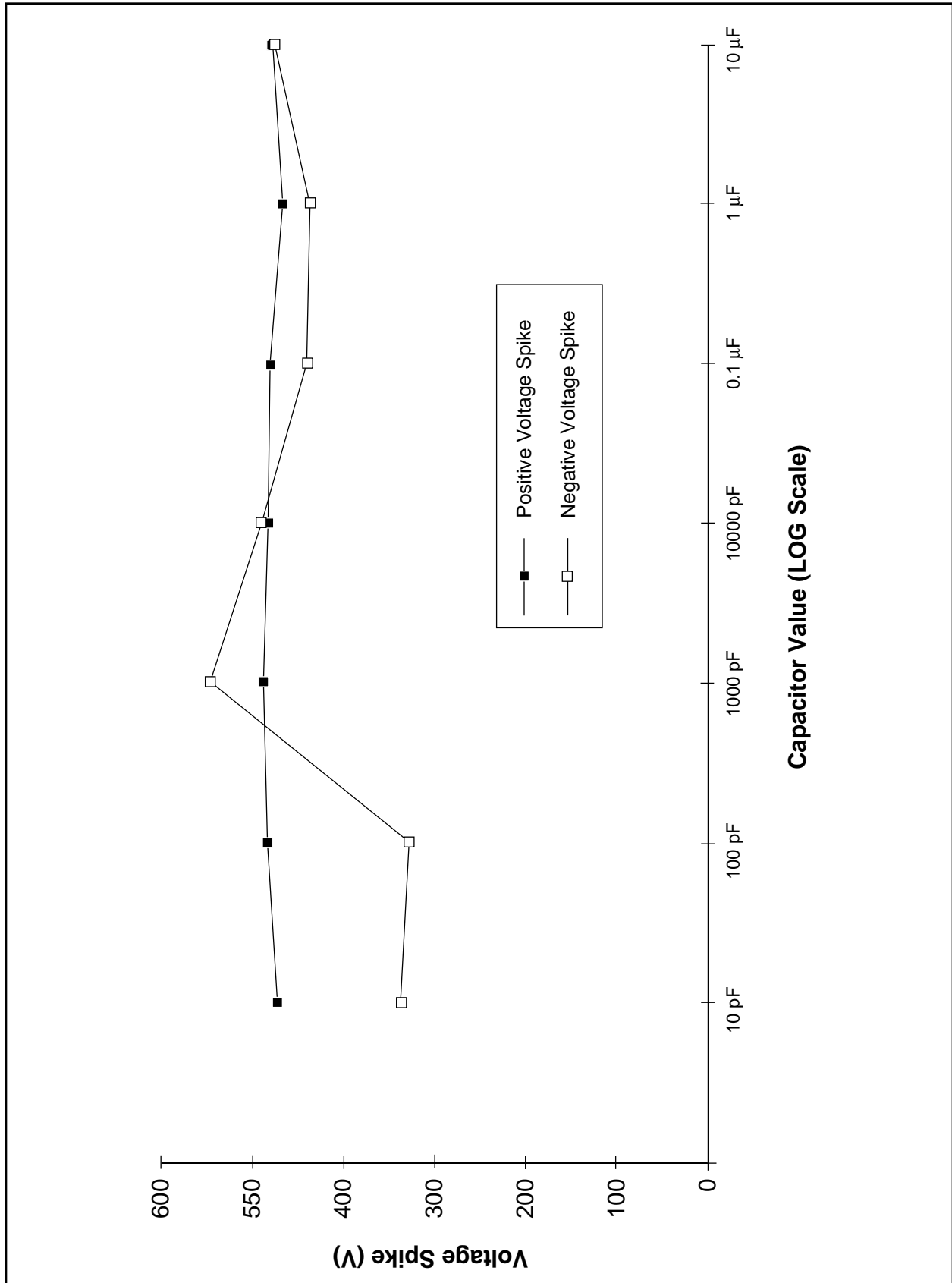


FIGURE 6: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON WP. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.



AN595

FIGURE 7: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON V_{CC}. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.

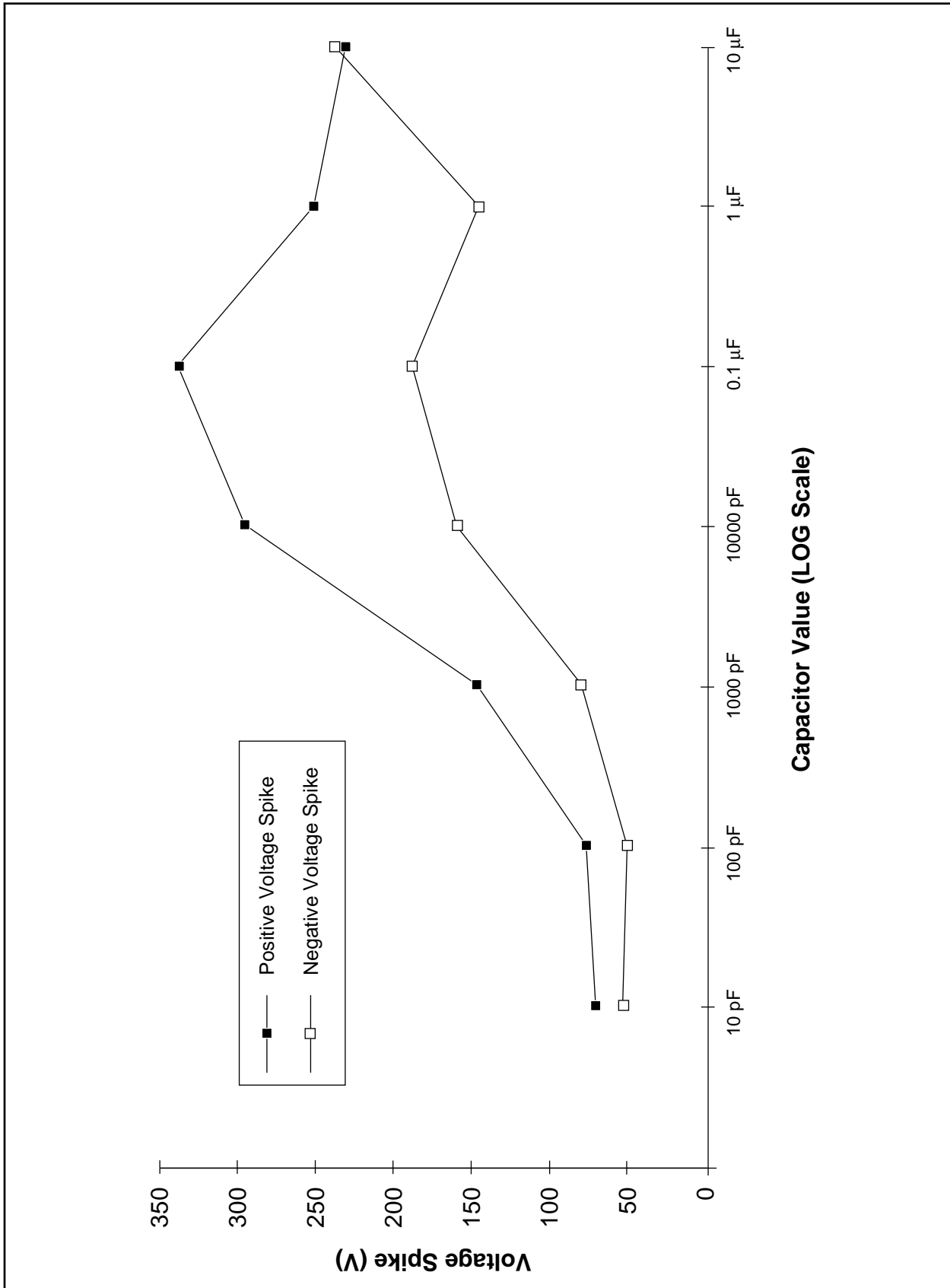
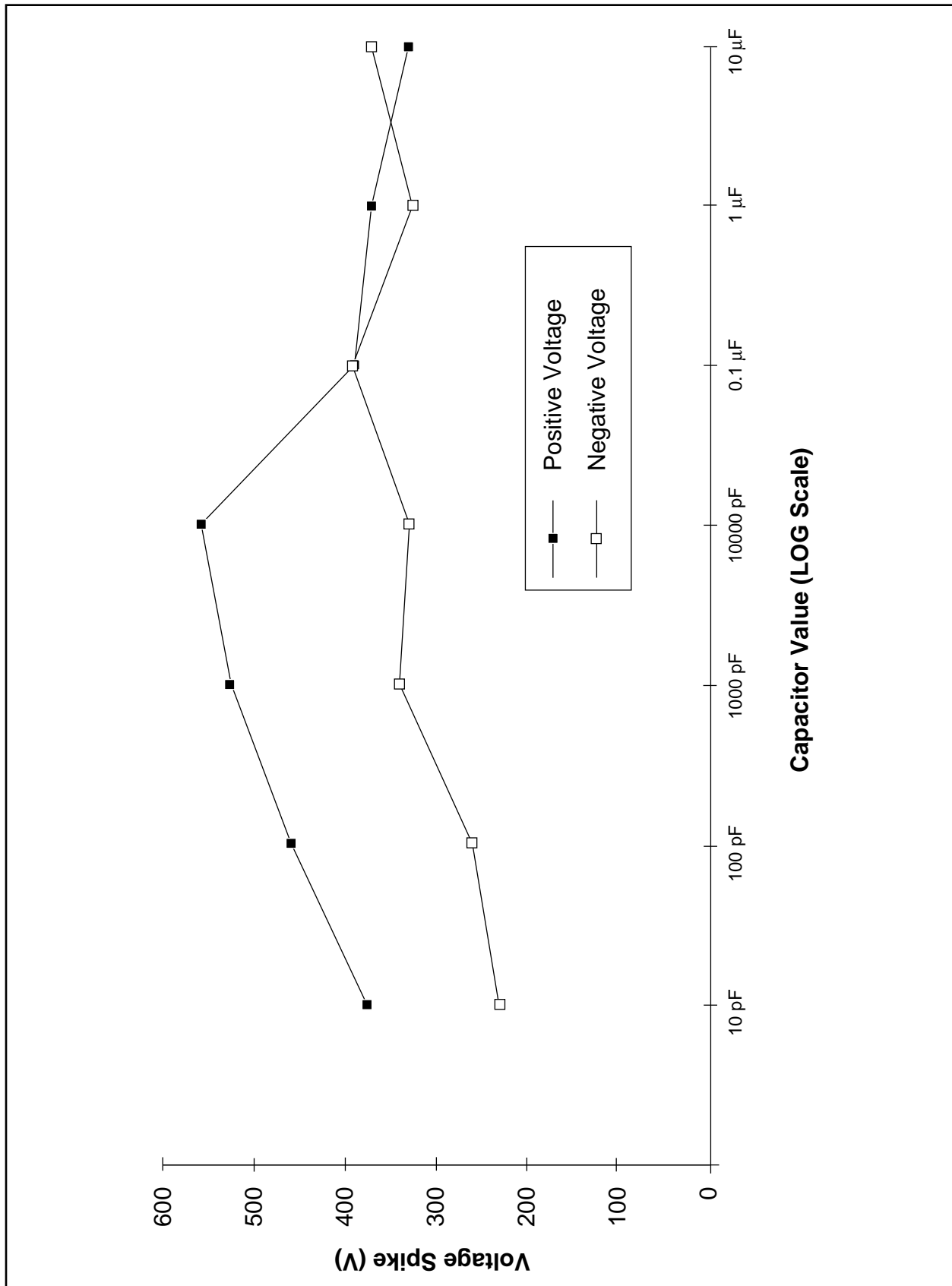


FIGURE 8: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RTCC. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.



AN595

FIGURE 9: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON \overline{MCLR} . PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC} .

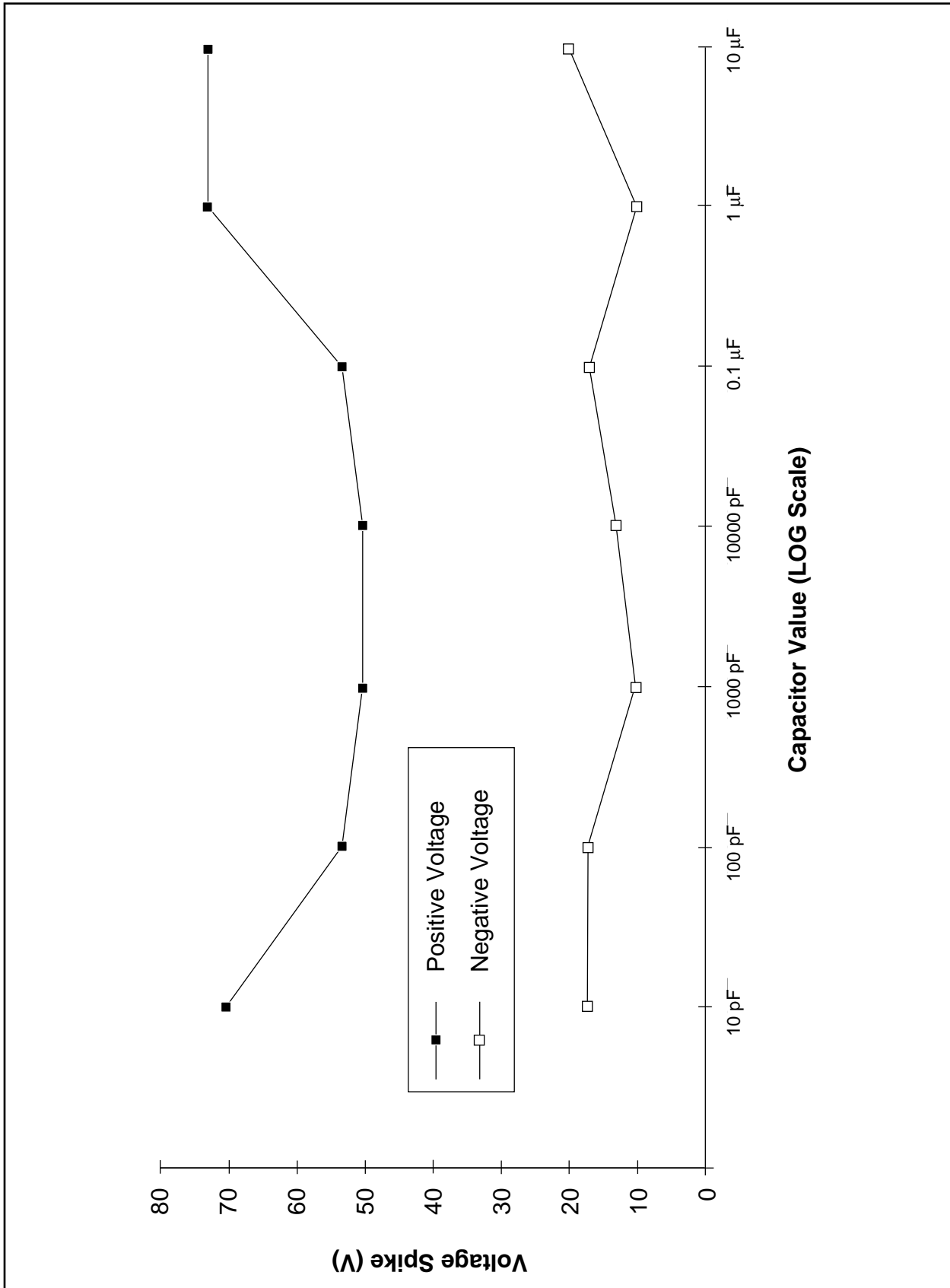
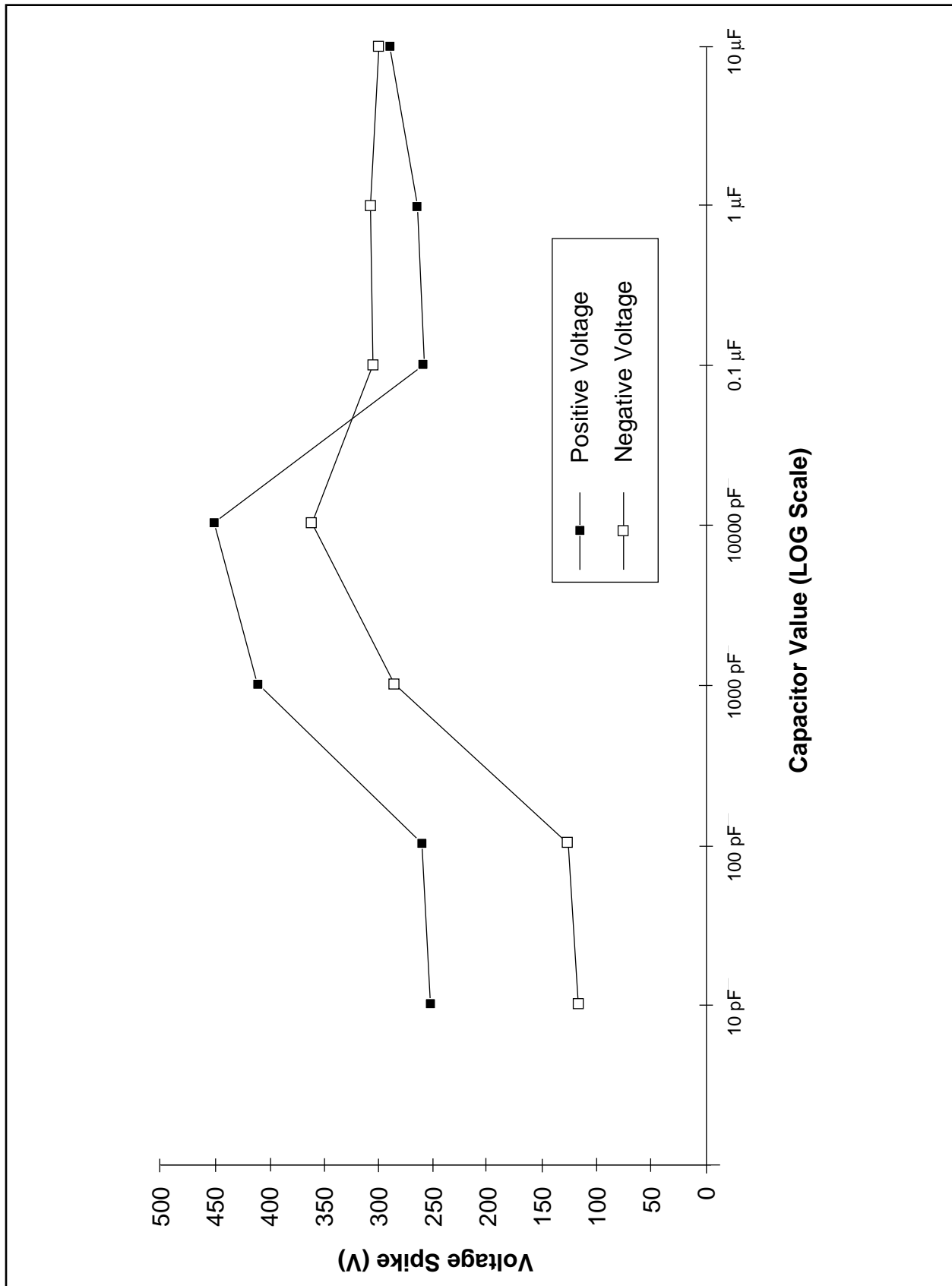


FIGURE 10: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON V_{SS}. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.



AN595

FIGURE 11: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RB PORTS. PROTECTION CAPACITOR BETWEEN VSS AND VCC.

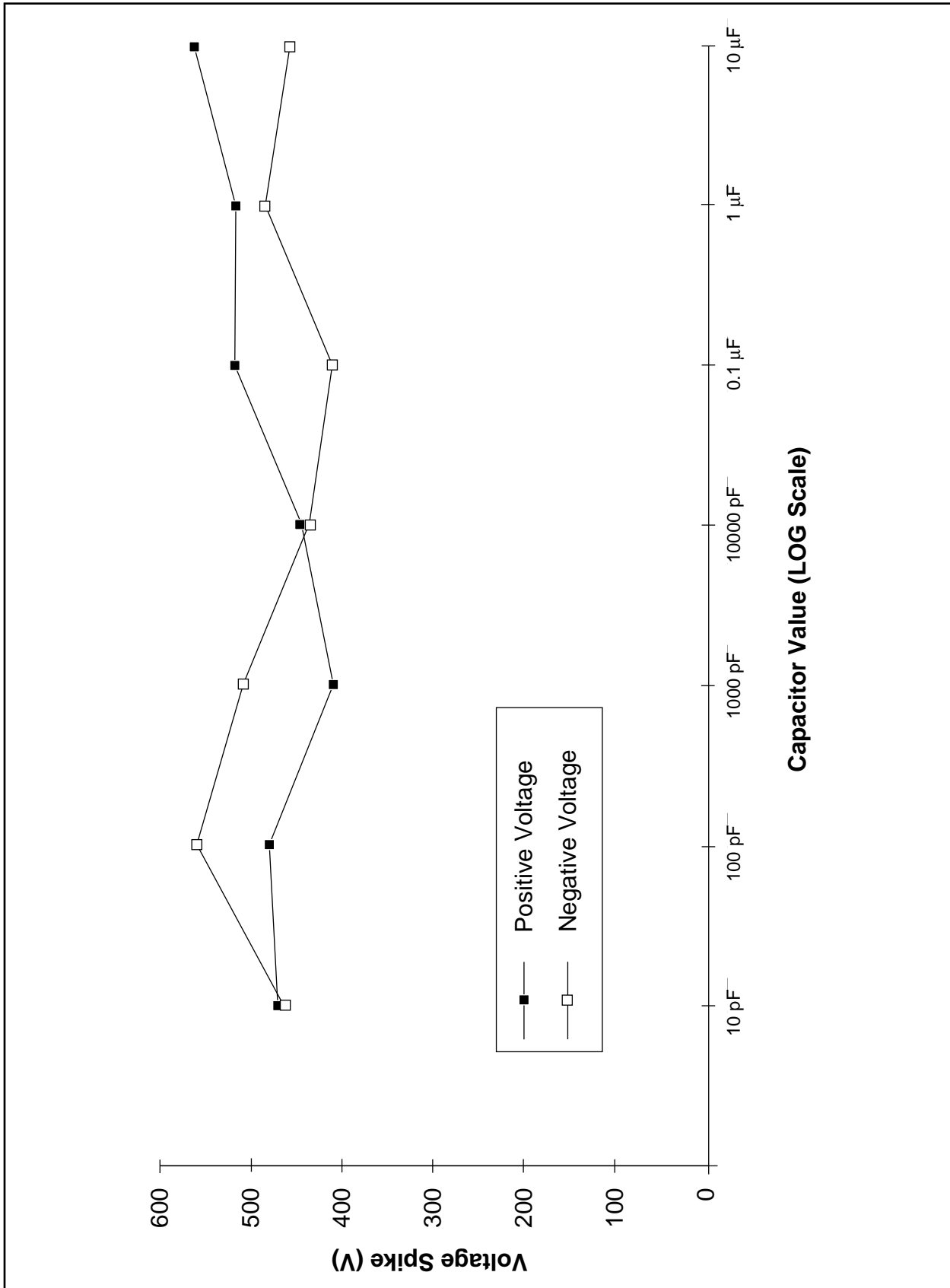
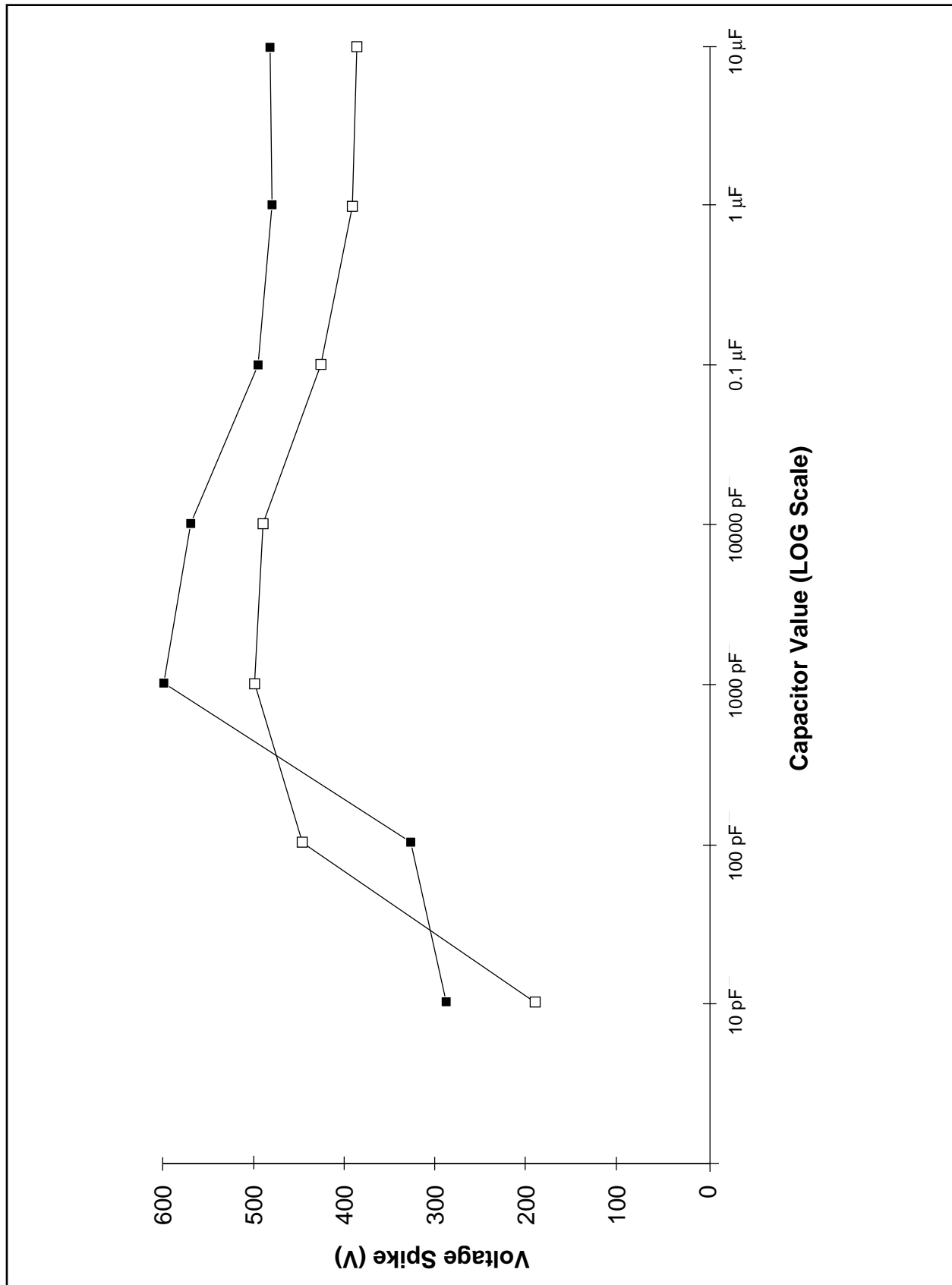


FIGURE 12: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RA PORTS. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.



AN595

FIGURE 13: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC1. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.

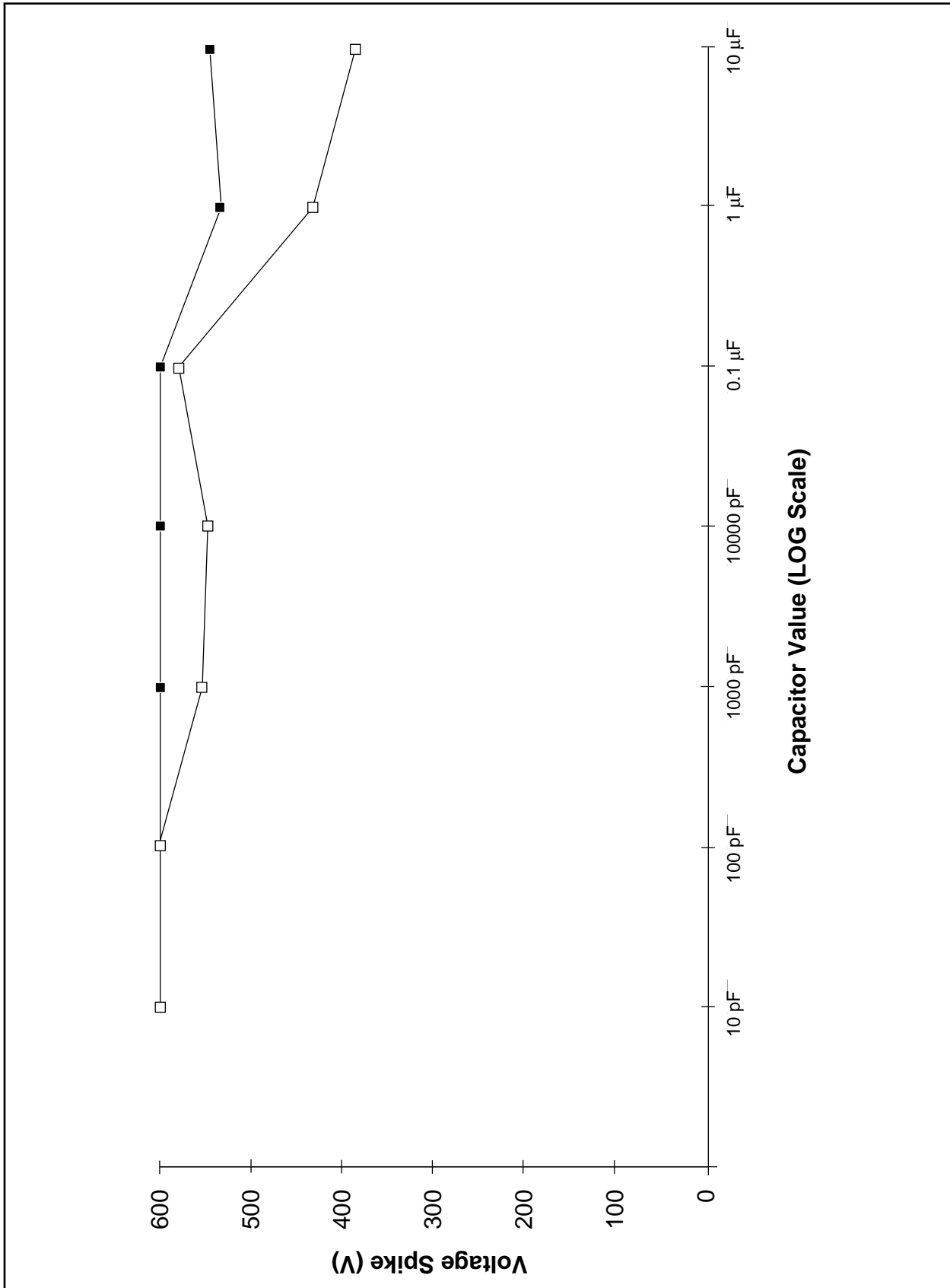
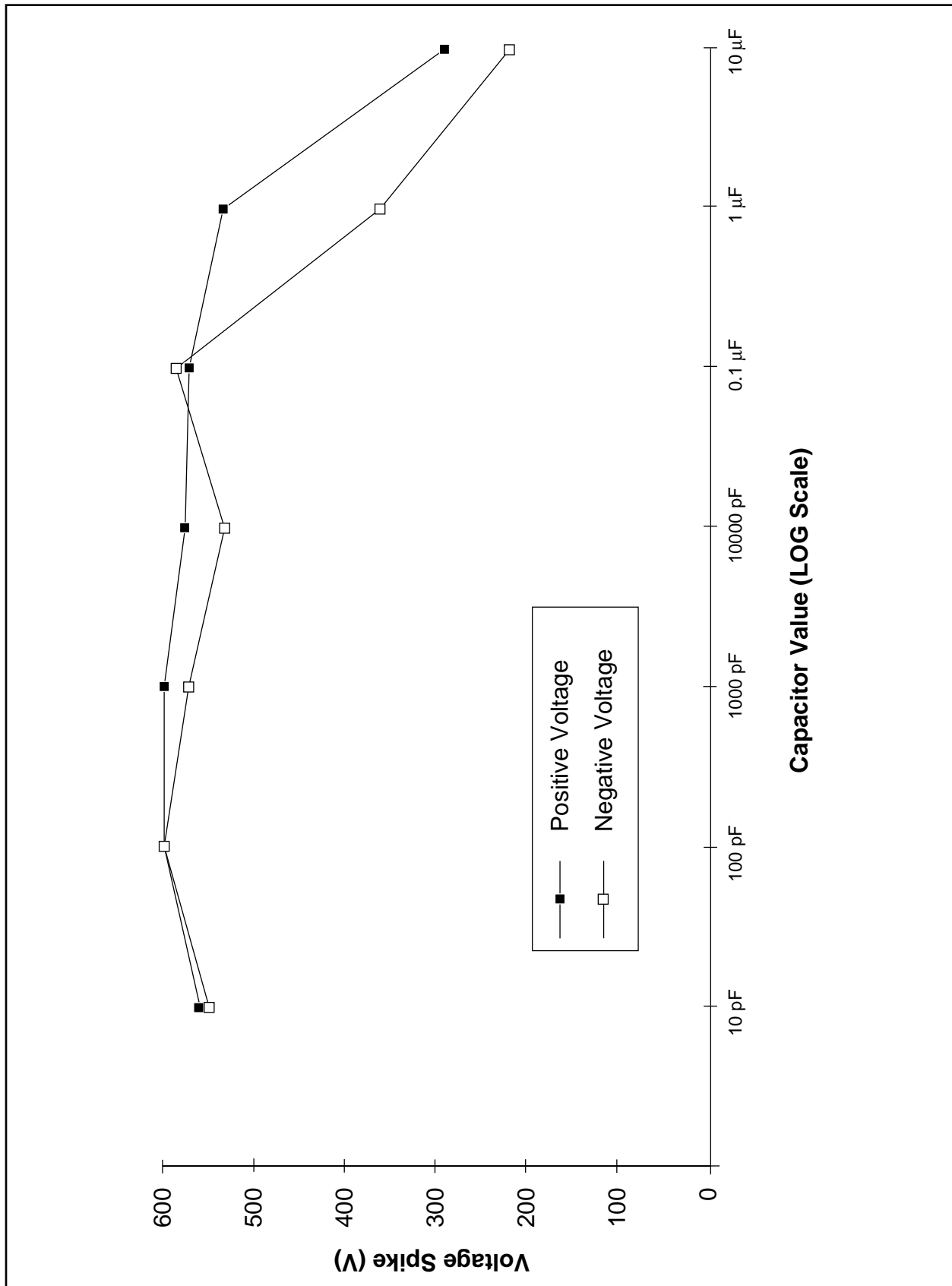


FIGURE 14: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC2. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.



AN595

FIGURE 15: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC1. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.

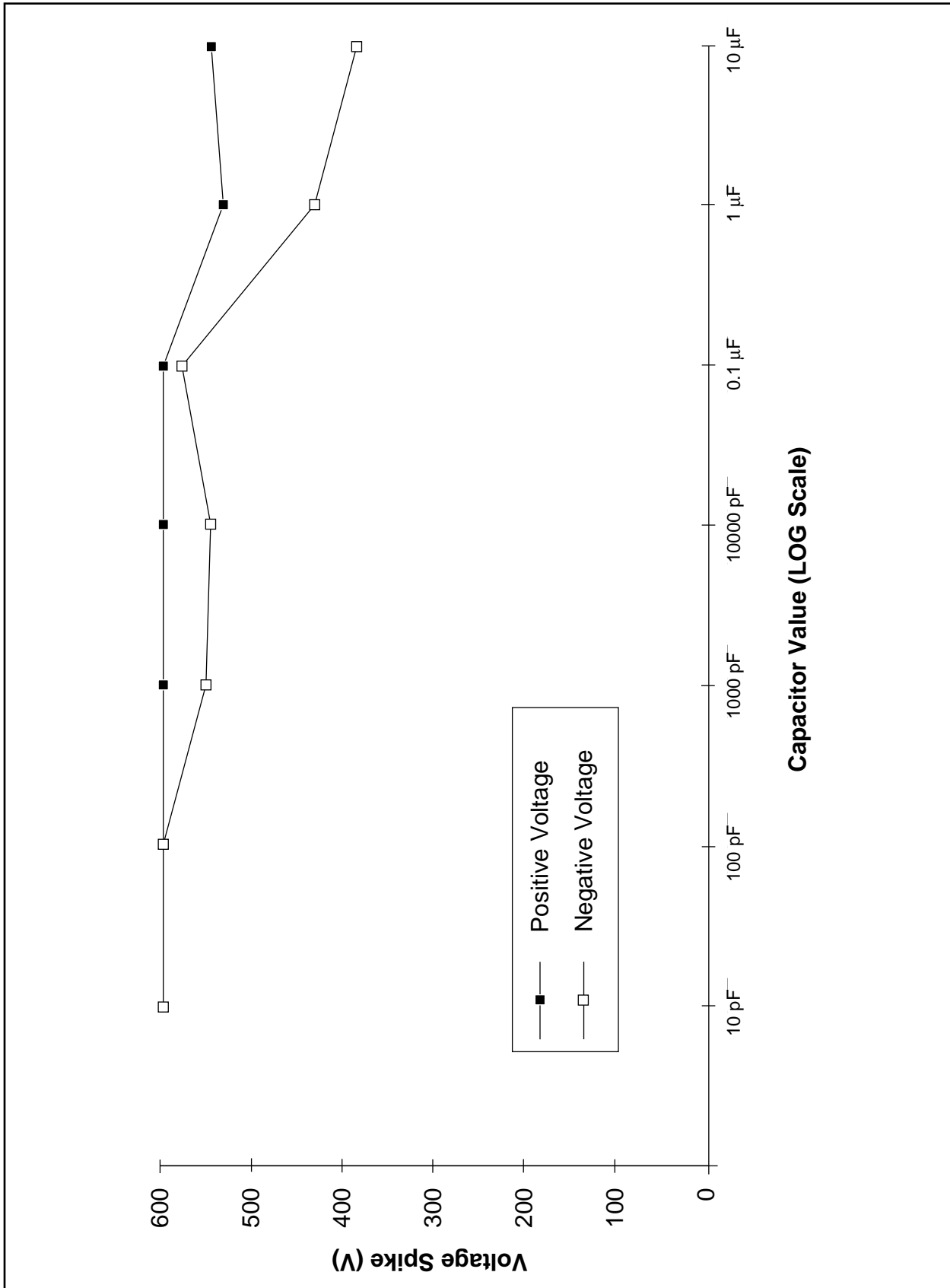
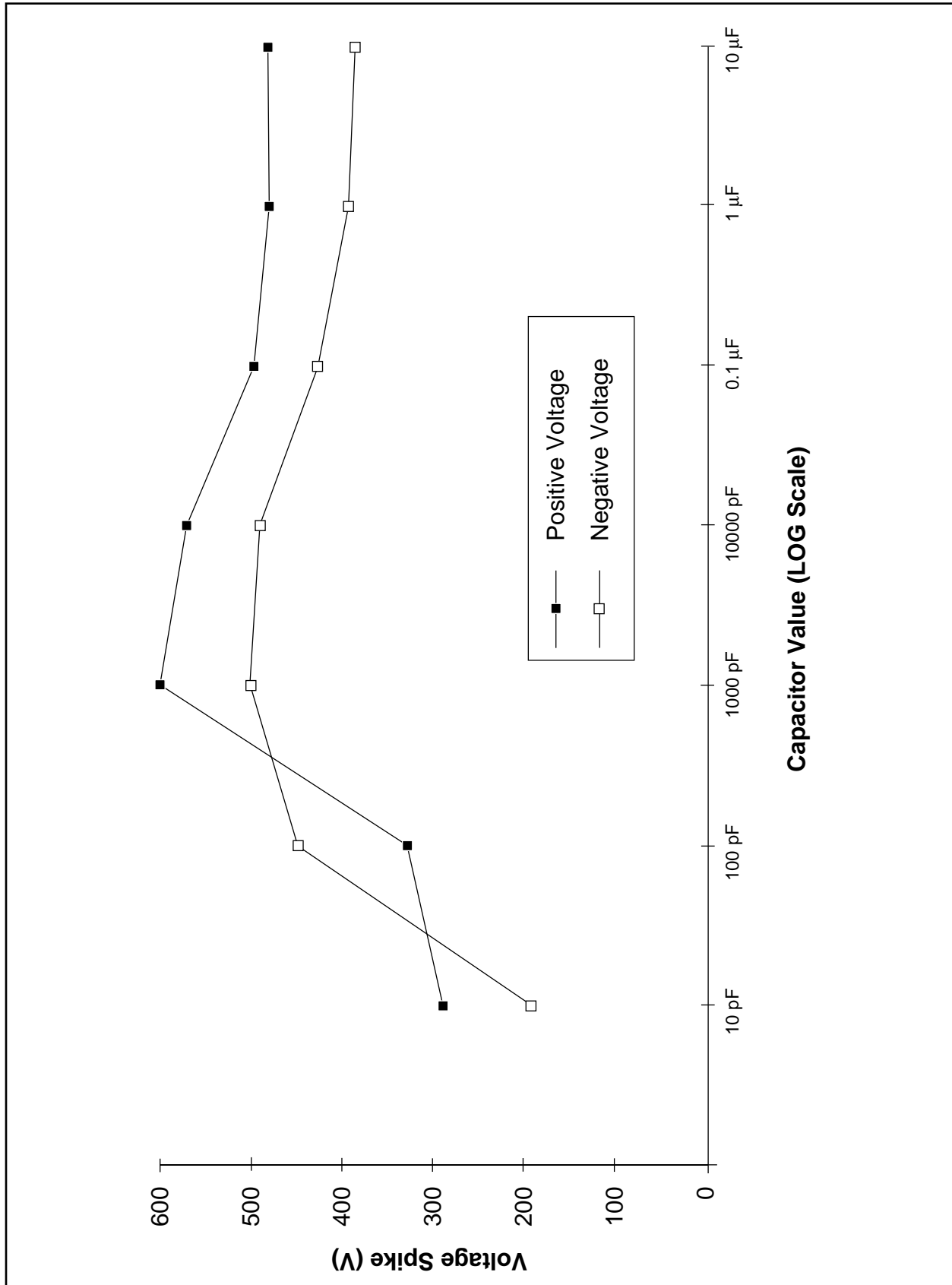
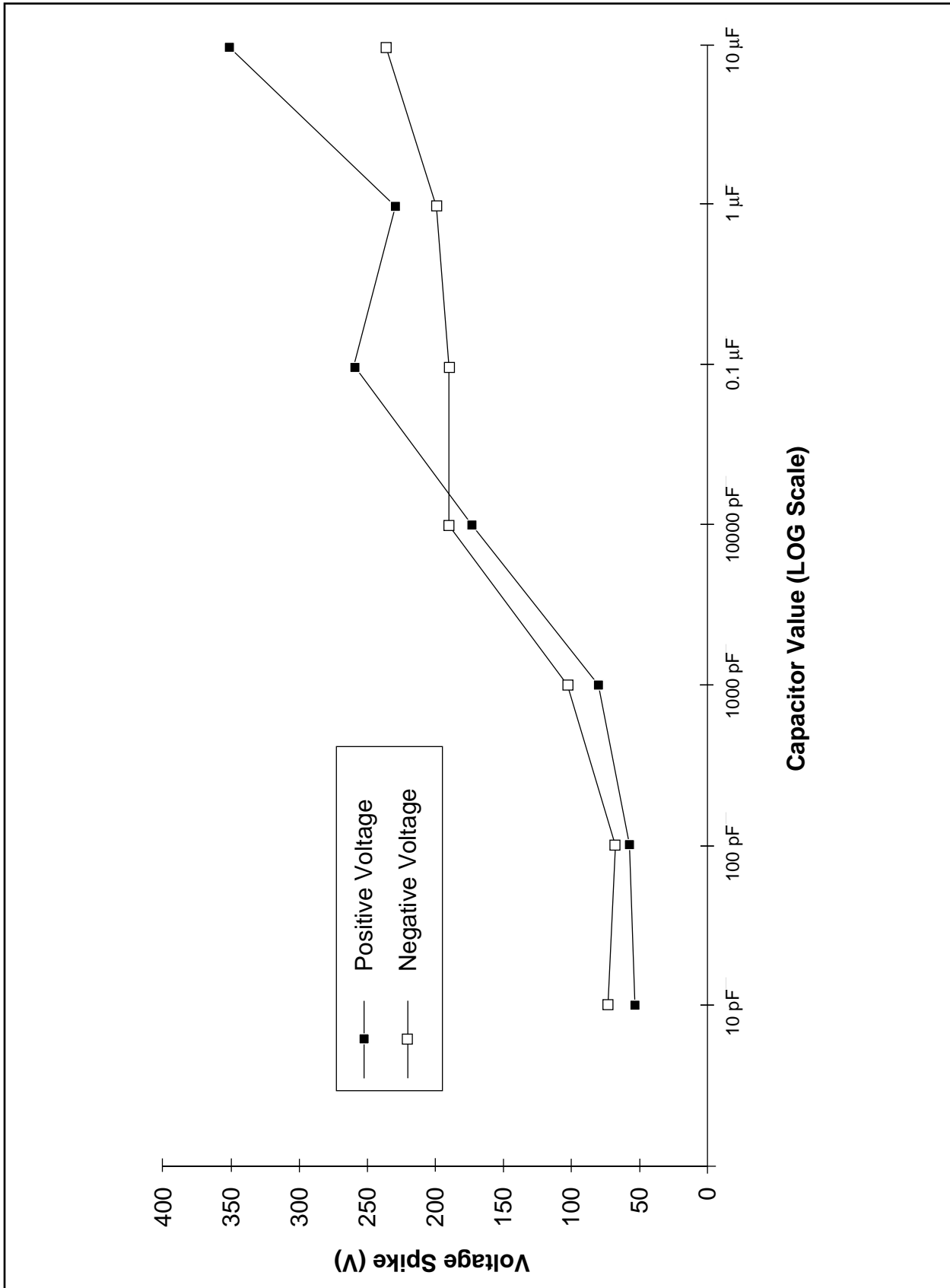


FIGURE 16: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RA PORTS. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.



AN595

FIGURE 17: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON V_{CC}. PROTECTION CAPACITOR BETWEEN V_{SS} AND V_{CC}.



Plastic Packaging and the Effects of Surface Mount Soldering Techniques

<i>Author:</i> <i>John Barber</i> <i>Surface Mount Technology Team</i>

PURPOSE

This application note is intended to inform and assist the customers of Microchip Technology Inc. with Surface Mount Devices (SMD's). The process of packaging a semiconductor in plastic brings to pass a somewhat unlikely marriage of different materials. In order to minimize potential adverse effects of surface mount solder techniques, it is worthwhile to understand the interaction of the package materials during the time they are subjected to thermal stress. Understanding both the limits of thermal stressing that SMD's can withstand and how those stresses interact to produce failures are crucial to successfully maintaining reliability in the finished product. A recommended Infrared (IR) solder profile is provided as a reference later.

The electronics industry has moved to smaller and thinner surface mount packaging in the progress toward miniaturization of circuits. This trend has necessitated the use of lower profile and smaller footprint packages. There has been an increase in reliability problems corresponding with the shrinking size of plastic SMD's. These problems manifest themselves in such ways as moisture sensitivity, cracked packages, open bond wires and intermittent continuity failures. Problems of this type are not present in the devices prior to assembly onto printed circuit boards but are the result of thermally induced stressing to the part during assembly or any rework such as desoldering.

WHAT CAN HAPPEN DURING THE SOLDER PROCESS

In surface mount soldering both the body and leads are intentionally heated. This direct heating of the reduced sized device package is at the heart of the problems experienced with Surface Mount Technology (SMT). Older techniques were concerned with the heating of the leads only. Some degree of heating was present in the body due to the thermal conductivity of the leadframe but this did not produce the same level of stress that devices are now subject to with SMT.

The heat from soldering causes a buildup of additional stresses within the device that were not present from the manufacturing process. Board level solder processes, such as IR reflow and Vapor phase reflow, are well-known areas where temperatures can reach levels sufficient to cause failure of the package integrity. A plastic semiconductor package forms a rigid system where the various components are locked together. Differences in the physical expansion rate of materials will result in internal package stresses because the constituent parts cannot move. When a package is heated, the stresses in the device are applied to the die in such a way that the maximum areas of stress¹ are at the corners. Forces can build to the point where the areas of adhesion between different components of the package give way causing device failure. Failure modes associated with excess stress include delamination of surfaces, fracturing of bond wires, die cracking, cratering of bond pads and package cracking.

Moisture sensitivity of plastic packages has been a concern for semiconductor manufacturers. Moisture can and will permeate any molding compound. The rate of permeation will vary with package compound thickness and type. Relative humidity will also play a role in the time required to saturate a device.

Moisture content will affect the ability of a device to withstand the stresses of surface mount soldering. If sufficient moisture is present inside of a device during soldering, high temperatures can cause steam which has the potential to crack the package. This type of damage is commonly called "pop-corning"².

Moisture can lead to corrosion of exposed aluminum metallization inside the device. Fortunately, a film of water is required³ for corrosion to take place. Water vapor alone is not sufficient to produce the onset of corrosion. It is difficult to collect a film of water inside a package where there is no defect.

Chemical compatibility is important to control corrosion of aluminum (especially in the presence of moisture). Most molding compounds and die attach epoxies contain free ions that can lead to corrosion under conditions where moisture is available to support the chemical reaction(s). Careful selection and handling of materials minimize the number of chemical impurities in the device that could lead to corrosion.

MATERIAL INTERACTIONS

To understand the significance of what is happening to a device during solder reflow, it is necessary to understand something about a few specific material properties and how those properties interact. Physical properties of interest⁴ are listed for reference in Table 2. These properties will be defined as needed to explain various concepts.

There are five major components used in a plastic package. Basic package components are: (1) epoxy molding compound, (2) leadframe made of copper or Alloy 42, (3) die attach epoxy, (4) silicon die and (5) gold bond wires. Molding compounds have several significant contributing factors that define their performance. These must be considered by the manufacturer when a selection is made. These items are Temperature of Glass Transition (T_g), Coefficient of Thermal Expansion (CTE), moisture absorption characteristics, flexural modulus and strength, and thermal conductivity. In reality, molding compound suppliers provide the test bed for development of compounds, only a few of the very large semiconductor manufacturers have published data⁵ suggesting independent experimentation into this area. Leadframes used by Microchip are copper with a silver plated area for die attach and wire bonding. Die attach material is typically a silver filled epoxy. The silver is added for thermal and electrical conduction. Plastic devices have gold bond wires.

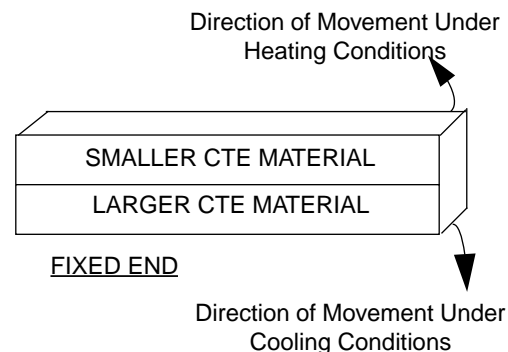
Thermally induced transients generated during surface mount soldering can have a significant impact on the reliability of plastic encapsulated devices in the field. In most cases, thermal transients below 125°C are not of sufficient magnitude to cause damage to the part. As a device experiences a significant thermal transient, such as would result from IR reflow soldering, the differing materials expand and contract at unequal rates. The CTE describes the behavior of a material as it expands or contracts when subjected to a temperature change. Materials with similar expansion coefficients will have similar thermal behavior if the phase boundaries are not approached. The CTE characterizes performance over a given temperature range. Table 2 shows that there is a change in the coefficient (ΔCTE) of roughly 12.9 ($16 - 3.1 = 12.9$) between a copper leadframe and silicon, and a ΔCTE of 4.4 ($7.5 - 3.1 = 4.4$) between silicon and lowest stress molding compounds available. The rate of expansion and contraction is not a simple linear relationship with temperature change. The rate can vary dramatically with the phase of the material. For example, molding compounds will greatly increase the rate of expansion when temperatures above the T_g is reached.

In general terms, T_g is the temperature where the material changes from a solid to something more like a plastic or vice versa. More precisely, the temperature of glass transition would be the temperature at which atoms, in chains of 30 to 40 atom groups, start to move.

Generally, T_g is in the area of 170°C for molding compounds. Several factors affect T_g such as the basic formulation of the resin from the supplier, cure time and the temperature used in the manufacturing process. A common misconception when trying to relate to these concepts is to assume that the entire package is at the same temperature at any given moment in time. This is not the case.

To aid in understanding the effects of CTE mismatching, take, for example, a bi-metallic strip. They can be found in every automobile and are used in turn signal flasher units for controlling the flashing action of the lights used as turn indicators. Inside the flasher unit the contact arm is formed by use of a bi-metallic strip to make and break electrical contact.

The metals used in the construction of a bi-metallic strip have different CTE's and are specifically chosen to produce a desired effect. These two materials are bonded so that they move together. In the automobile example, current flowing through the bi-metallic strip causes local heating of the strip. Due to the unequal CTE's, the strip will bend away from the contact as it is heated which causes the circuit to open. When current stops flowing in the strip, heating also stops. The bending action of the strip is produced by unequal expansion on one side of the strip. As the strip cools, it will move back thus making contact again allowing current induced heating which starts the cycle over again. In similar fashion as bi-metallic strip, the die in a plastic package will be stressed when subjected to a thermal transient.



Adhesion of materials is a matter of importance since it is in areas of delamination that moisture can collect. The collecting of moisture can lead to corrosion problems.

Differing rates of expansion and contraction can make joining two materials a distinct challenge. Especially if the materials are significantly different in other characteristics that affect surface adhesion. Some materials that are not suitable for use in the packaging system require special adhesion promoting modifications of the surfaces. The topic of adhesion is beyond the scope of this work, but is an important factor and should be given careful attention⁶.

The property known as “Fracture Toughness” is the ability of the material to resist the propagation of a fracture once the defect has been initiated. Silicon has a very low fracture toughness, therefore, a fracture will readily propagate in silicon. An example of a material at the other end of the spectrum would be Gold, one of the ductile metals, which has a high fracture toughness. This property is responsible for the tendency for glass or tungsten carbide to shatter rather than simply chip or crack. In semiconductor devices, fracture toughness should be considered when cratering and die cracking are a problem.

The closest point to a zero stress state in a plastic package is at the temperature used to cure the molding compound ($\cong 175^{\circ}\text{C}$). Present plastic molding compounds are thermosetting polymers. Thermosetting means that the compound sets up and becomes hard as a result of being heated. A cooling cycle after the set up period does not undo the process. The molding compound is held at an elevated temperature for the time period required to harden or cure. During the subsequent cooling, after cure, stresses are trapped in the package because of the differing CTE's. This trapping of stress results in a net compressive force, at room temperature, on the die surface in a plastic molded device. As external stresses such as thermal stress from soldering are presented, if stresses are of sufficient magnitude, the material strength will be exceeded resulting in package damage. Manufacturing processes tend to leave stresses trapped inside the devices. Thickness of the die attach material is regulated to control stressing levels due to its presence. A thin die attach will result in higher tensile stress on the die surface after die attach. The interaction of the die attach material and the molding compound result in the compressive stress on the die surface⁷ after packaging. Low stress molding compounds are used on SMD's to minimize the thermal stresses generated during soldering operations.

FAILURE MODES AND MECHANISMS

Let us now review the types of damage that may be seen as a result of SMT soldering. Package damage may be manifested in several ways and may not always result in immediate device failure. Following is a list of failure types and their morphology:

1. Delamination of the molding compound along the leadframe interface and or die surface can take place. This delamination, or separation, can provide a path for moisture and contaminant ingress and pooling along the interfaces where the materials are no longer adhering to each other. This condition may lead to corrosion related problems⁸.
2. Cracking of the mold compound can produce immediate failure if the crack crosses a bond wire or it can allow similar moisture effects⁹ as in delamination. It can also produce intermittent contact problems.
3. Cracking of the die is generally seen as a functional failure but can be temperature sensitive if the crack is in a more benign area of the die.
4. Cratering of wire bonds is characterized as a phenomenon where portions of the silicon below the ball bond are fractured. The ball bonds pull up “plugs” or “chunks” of silicon, thereby, leaving crater shaped damaged spots in the bulk silicon below the bond pad. Cratering is a possible result of lateral stress on ball bonds. A cratering failure will typically but not always show up during electrical testing. However, it can lay dormant until another temperature excursion comes along which causes the Al metal conductor (bond pad) to open. This is a result of the soft nature of the aluminum used for interconnects in semiconductor devices. The silicon below a bond pad can be damaged without breaking contact in the aluminum. Intermittent or thermally sensitive continuity failures may be produced.
5. Moisture inside the device may collect in the die attach, along material interfaces (primarily leadframe to mold compound and leadframe to die surface) or in the molding compound. Rapid heating causes pressure build up as the moisture expands. This results in delamination and cracking of the package along with the failure modes associated with those phenomena.
6. Corrosion is heavily related to moisture effects and is intensified by delamination and cracks. The typical failure mode for corrosion is loss of continuity since the area most often affected is in the bond pads where there is no passivation layer to give additional protection to the metal.

HOW TO CORRECT THIS PROBLEM: A CASE STUDY

After the device leaves the factory there are two things that need to be done to maintain reliability.

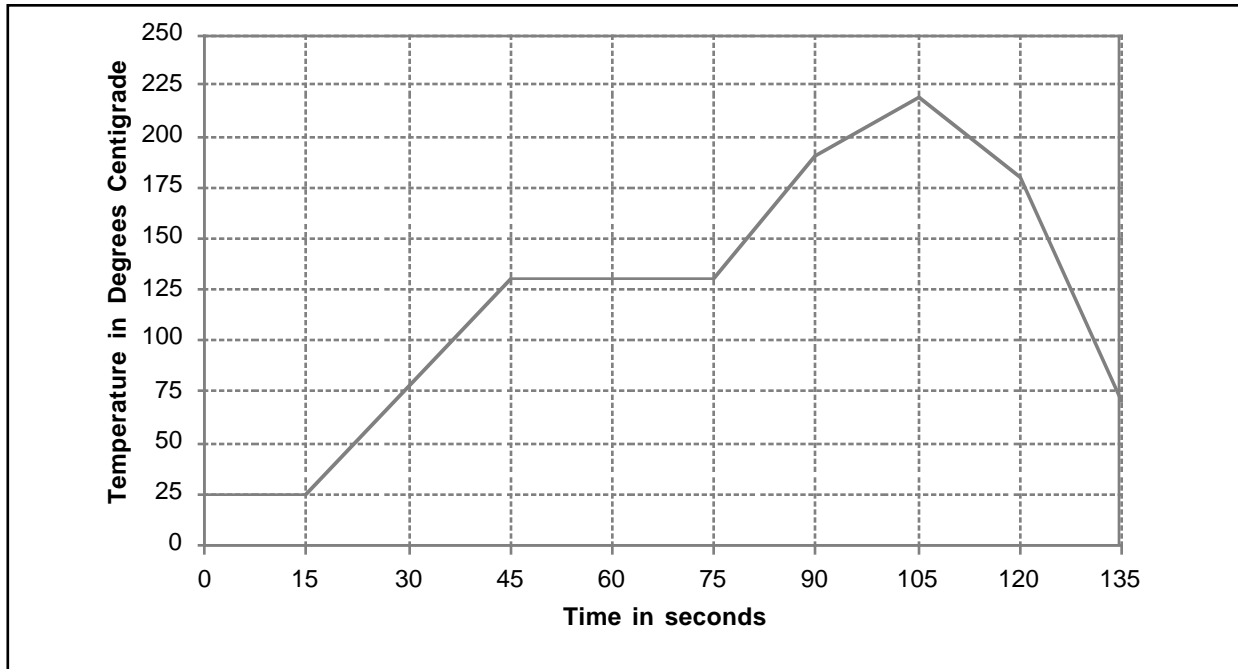
1. The exposure to thermal stressing needs to be minimized. This is not always an easy task in light of the varied components that may go onto a single board. Some will require different conditions to solder due to their bulk size.

Microchip carried out a partial factorial experiment on 32-lead PLCC devices with the intent of verifying an industry standard profile and providing a specification for our various customers. The devices were subjected to IR solder reflow profiles with ramp rates ranging from 1.5°C/second to 4°C/second and maximum temperatures from 220°C to 310°C (Table 1).

TABLE 1: 32-LEAD PLCC PACKAGE PARTIAL FACTORIAL EXPERIMENT

Variables used in the analysis	Conditions							
M = Maximum Temperature (C)	220	235	250	265	280	295	310	7
R = Ramp Rate (C/second)	1.5	2	3	4				4
P = Number of Passes	1	2						2 56 Total Conditions

EXAMPLE 1: IR REFLOW PROFILE



1. (Continued)

No drypack state was employed and the parts were allowed to sit on the shelf in excess of 30 days prior to temperature exposure. The RH of the room was not recorded but is estimated to be approximately 60%. This condition was employed to insure that worst case conditions were evaluated.

Our experiments, to determine optimum soldering temperatures, indicated that a maximum temperature of 220°C should be observed during solder processing. The temperature ramp rate was found to be a less significant factor but should be kept in the range of 2 to 5°C per second. Very slow ramp rates, i.e. those of less than 2°C per second, show a slight decrease in performance. Ramp rates of 3°C showed the best performance and should be used whenever possible. Results from the experiment independently confirm the exposure recommendations set forth in IPC-SM-786¹⁰. A sample profile for IR reflow is illustrated by the graph in Example 1.

Prior to exposure to the IR temperature profiles all parts were electrically good and were examined by Scanning Acoustic Microscopy (SAM) to verify a

defect free state and establish a baseline. After exposure to the IR profiles, all parts were again tested electrically and examined via SAM. Delamination can be easily detected by use of SAM. No package cracks were observed after temperature exposure. Only limited electrical failures occurred and all were at temperatures above 220°C.

In the experiments a thermocouple was interfaced to the exterior of a device and passed through an IR reflow oven. Both the maximum temperature and ramp rate were recorded in this manner. General trends for temperature sensitivity in relationship to maximum temperature and ramp rate are presented in the graphs shown in Example 2 and Example 3.

Pass/fail criteria for the experiments was primarily the extent of delamination, observed by Scanning Acoustic Microscopy, after exposure to two passes through an IR reflow oven. Top side die delamination and delamination in areas of the leadframe where the bonds are made are classed as unacceptable. Some delamination along non critical areas are classed as acceptable (i.e., no delamination extending to the exterior of the

package or covering more than minor portions of the leadframe). Specific information on pass/fail criteria can be found in IPC - SM - 786.

Maximum temperature should not exceed 220°C.

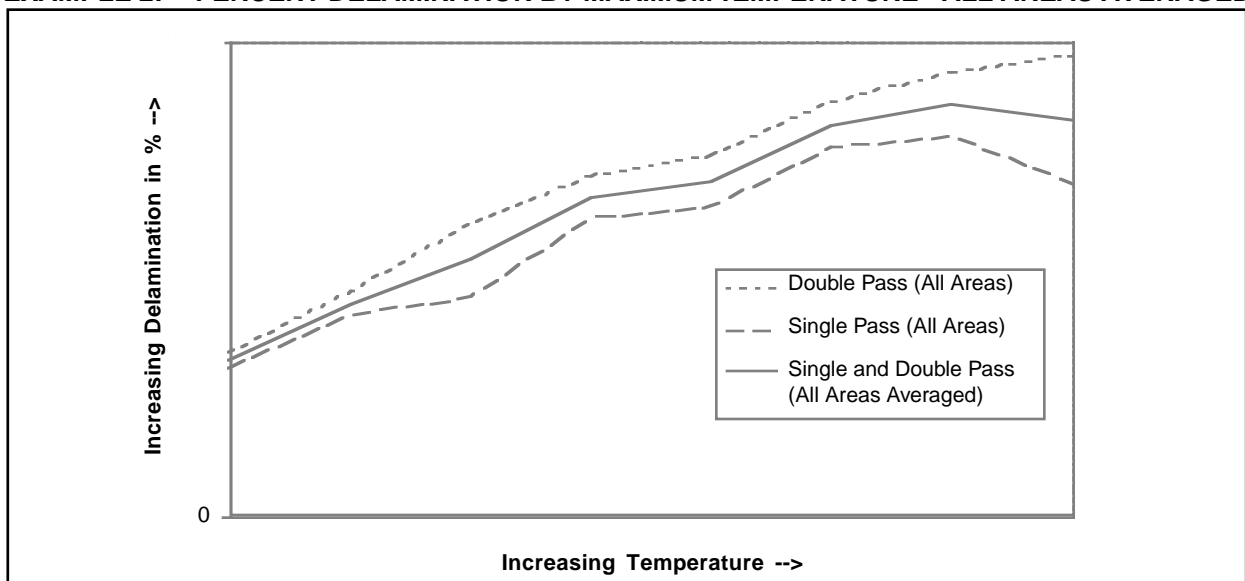
Temperature ramp rate should be between 2 to 5°C per second.

Use of the above temperature profile will increase the reliability of the device following board level assembly.

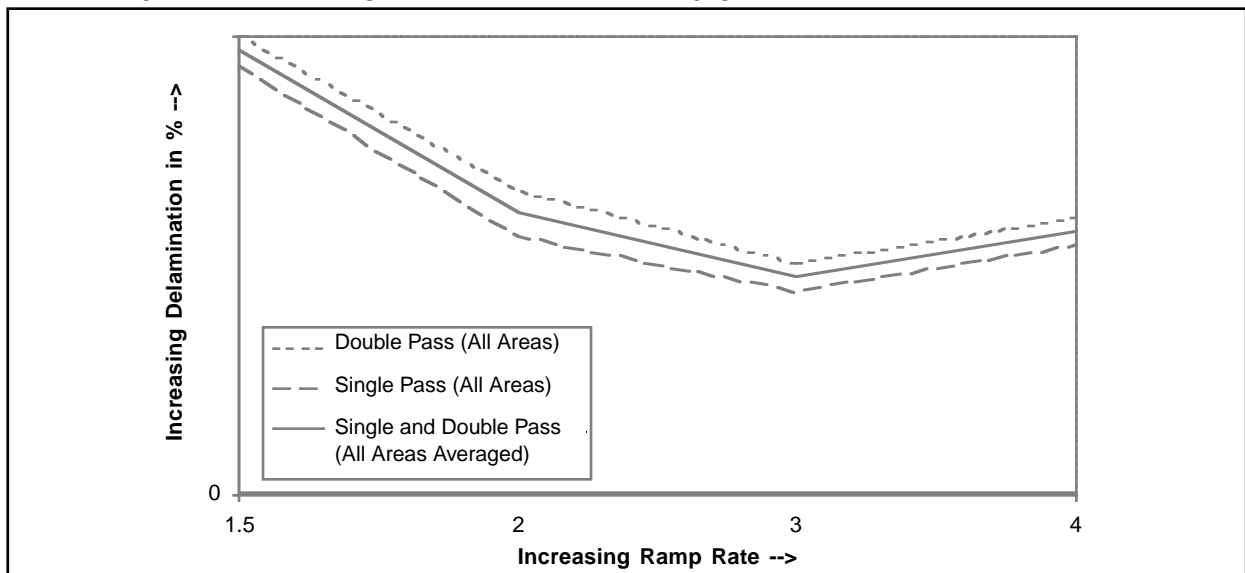
- The second item that must be observed is the dry packing of SM devices. Dry packing is a standard product packaging procedure for surface mount devices. This practice

guarantees to the customer that devices are shipped in a dry state. All devices are sealed inside plastic bags with a moisture indicator to monitor moisture when they are shipped. Customers should be aware of this practice and maintain the dry packed state until the time of use. Shelf life¹¹ for opened parts is a function of package style and ambient conditions. A small increase in failure rate will be experienced if the drypack condition is violated.

EXAMPLE 2: PERCENT DELAMINATION BY MAXIMUM TEMPERATURE - ALL AREAS AVERAGED



EXAMPLE 3: DELAMINATION BY RAMP RATE AT 220°C



AN598

CONCLUSION

Significant thermal stressing can cause a diverse list of package or device failures. To minimize any damage caused by high temperature exposure, moisture content needs to be controlled in electronic devices.

Experiments conducted by Microchip validate the recommendation of IPC - SM - 786 to limit maximum temperature exposure during surface mount soldering

to 220°C. A temperature ramp rate of 2°C/second to 5°C/second will also serve to safeguard the reliability of the device.

Microchip is committed to the principles of continuous improvement. Product reliability and customer satisfaction are a primary focus. For this reason, new packaging technology enhancements are continually evaluated to improve the performance of Microchip devices.

TABLE 2: PHYSICAL PROPERTIES OF COMMON MATERIALS USED IN PLASTIC PACKAGING OF SEMICONDUCTOR DEVICES

Material	Coefficient of Thermal Expansion in 10 ⁶ /°C (CTE)	Temp. of Glass Transition in °C (Tg)	Modulus of Elasticity in 10 ⁶ psi (10 ³ MPa.) (E)	Tensile Strength in 10 ³ psi
Silicon (Si)	2.3-4.2		19-29	16-25
Alloy 42 Leadframe	4.0- 4.7		21	75
Alumina (Al ₂ O ₃)	6.5-8.8		37	25-50
Gold Eutectic Die Attach	14.2		12.5	16
Copper Leadframe	16-18		17.3	60
Pb -- Sn Die Attach	29		1-2.5	1-4
Epoxy Mold Compounds (α ₁ is for Temperatures below Tg, (α ₂ is for temperatures above Tg)	7.5-28 α ₁ 20-22 ppm/°C α ₂ 60-65 ppm/°C ¹²	145-170	1.8-3	10-20
Epoxy Die Attach	20-70		0.4	1-8
Polyimide Die Attach	35-60		0.6	1-8
Unfilled Epoxies	60-70			
Unfilled Silicones	300			

ENDNOTES

1. Thomas M. Moore and Robert G. McKenna, *Characterization of Integrated Circuit Packaging Materials* (Stoneham: Butterworth - Heinemann, 1993), pp. 68.
2. A.S. Chen, et al., "A Study of the Interaction of Molding Compound and Die Attach Adhesive, with regards to Package Cracking", Proceedings of the 44th Electronic Components & Technology Conference. 1994, pp. 115-120.
Michael Tencer, "Moisture Ingress into Nonhermetic Enclosures and Packages. A Quasi-Steady State Model for Diffusion and Attenuation of Ambient Humidity Variations", Proceedings of the 44th Electronic Components & Technology Conference. 1994, pp. 196-209.
3. Thomas M. Moore and Robert G. McKenna, *Characterization of Integrated Circuit Packaging Materials* (Stoneham: Butterworth - Heinemann, 1993), pp. 13.
4. George G. Harman, *Wirebonding In Microelectronics* (International Society for Hybrid Microelectronics, 1991), pp. 160.
Justin C. Bolger, Short course notes. *Advanced Microelectronics Assembly Technology: Reliability & Yield Issues of Materials, Plastic Molding and Wire Bonding*, Univ. Of Ariz. February 8-11, 1994, pp. C2.
5. Samuel S. Kim, "Improving Plastic Package Reliability Through Enhanced Mold Compound Adhesion", Tutorial of the International Reliability Physics Symposium, 1992, pp. 2d.1-2d.17.
6. H. Ohsuga, et al., "Development of Molding Compounds Suited for Copper Leadframes", Proceedings of the 44th Electronic Components & Technology Conference. 1994, pp. 141-146.
7. Justin C. Bolger, Short course notes. *Advanced Microelectronics Assembly Technology: Reliability & Yield Issues of Materials, Plastic Molding and Wire Bonding*, Univ. Of Ariz. February 8-11, 1994, pp. C6.
8. T.M. Moore, S.J. Kelsall. "The Impact of Delamination on Stress-Induced and Contamination-Related Failure in Surface Mount IC's", *Proceedings of the International Reliability Physics Symposium*. 1992, pp. 169-176.
T.R. Conrad and R. L. Shook. "Impact of Moisture/Reflow Induced Delaminations on Integrated Circuit Thermal Performance", Proceedings of the 44th Electronic Components & Technology Conference. 1994, pp. 527-531.
9. T.M. Moore, S.J. Kelsall. *Op. Cit.*, pp. 169-176.
10. IPC - SM - 786A is obtainable from the Institute for Interconnecting and Packaging of Electronic Circuits, 7830 N. Lincoln Ave., Lincolnwood, ILL. 60646.
11. IPC - SM - 786A, Table 4-2 (See endnote above).
12. Thomas M. Moore and Robert G. McKenna, *Characterization of Integrated Circuit Packaging Materials* (Stoneham: Butterworth - Heinemann, 1993), pp. 49.

An Excellent Reference Article --

L.T. Nguyen, "Reliability of Postmolded IC Packages", Transactions of the ASME, Journal of Electronic Packaging, December 1993, Vol. 115, pp. 346-355.

AN598

NOTES:

Continuous Improvement

<i>Author: Randy Drwina Product Enhancement Engineering</i>

INTRODUCTION TO MICROCHIP'S CULTURE

The corporate culture at Microchip Technology Inc. is embodied in our *Guiding Values*. This culture has been key to our success in business because of its emphasis on customer satisfaction, quality, continuous improvement, empowerment and communication. The synergy of these values has created a very dynamic, very successful culture. They have impacted many aspects of Microchip, including the EEPROM Technology Team. Two of our *Guiding Values* in particular have steered this group.

Continuous Improvement is Essential

We utilize the concept of "Vital Few" to establish our priorities. We concentrate our resources on continuously improving the Vital Few while empowering each employee to make continuous improvements in their area of responsibility. We strive for constructive and honest self-criticism to identify improvement opportunities.

Products and Technology Are Our Foundation

We make ongoing investments and advancements in the design and development of our manufacturing process, device, circuit, system, and software technologies to provide timely, innovative, reliable, and cost effective products to support current and future market opportunities.

We recognized in the late 1980's that the industry as a whole did not have great endurance on EEPROMs. We were no exception. And this was at a 10,000 E/W cycle level, which is orders of magnitude less than where we are today. We also recognized that there were many uncertainties in the interaction of design, process and customers applications with respect to endurance performance and expectations. It was also clear that the customers were not educated enough on these interactions to be able to say clearly what they needed in a product. This was clearly an "improvement opportunity" that could allow us "to support current and future market opportunities".

THE EEPROM TECHNOLOGY TEAM

Microchip had been using the team approach to problem solving and continuous improvement activities such as new product introduction in the development groups and defect reduction in the manufacturing area. These had been in place for some time, and were beginning to work extremely well in bringing fast, well thought out and researched solutions to problems that crossed organizational boundaries.

A cross functional team was formed at Microchip that would be the technology management for an entire family of products. The team was comprised of engineers from design, process, test, product, quality, reliability, and yield enhancement, who reported in to different functional groups in the regular organizational structure. This team was charged with setting the technical direction for all EEPROMs at Microchip. This included both areas of development and sustaining, which were always in competition for the same resources.

The Microchip EEPROM Technology team went through the typical early phases of team development. The initial elation of having a great group of technical contributors wore off after a while. The competition for resources lead to some rough areas at the start, but this was overcome with the help of management support and direction. A balance developed, where the development side really understood the need to improve yields today, while the sustaining side understood the need for new products and innovations for the future. The emphasis on the Guiding Values and strong support from management for the team concept greatly helped the group through the early phases.

THE PROCESS

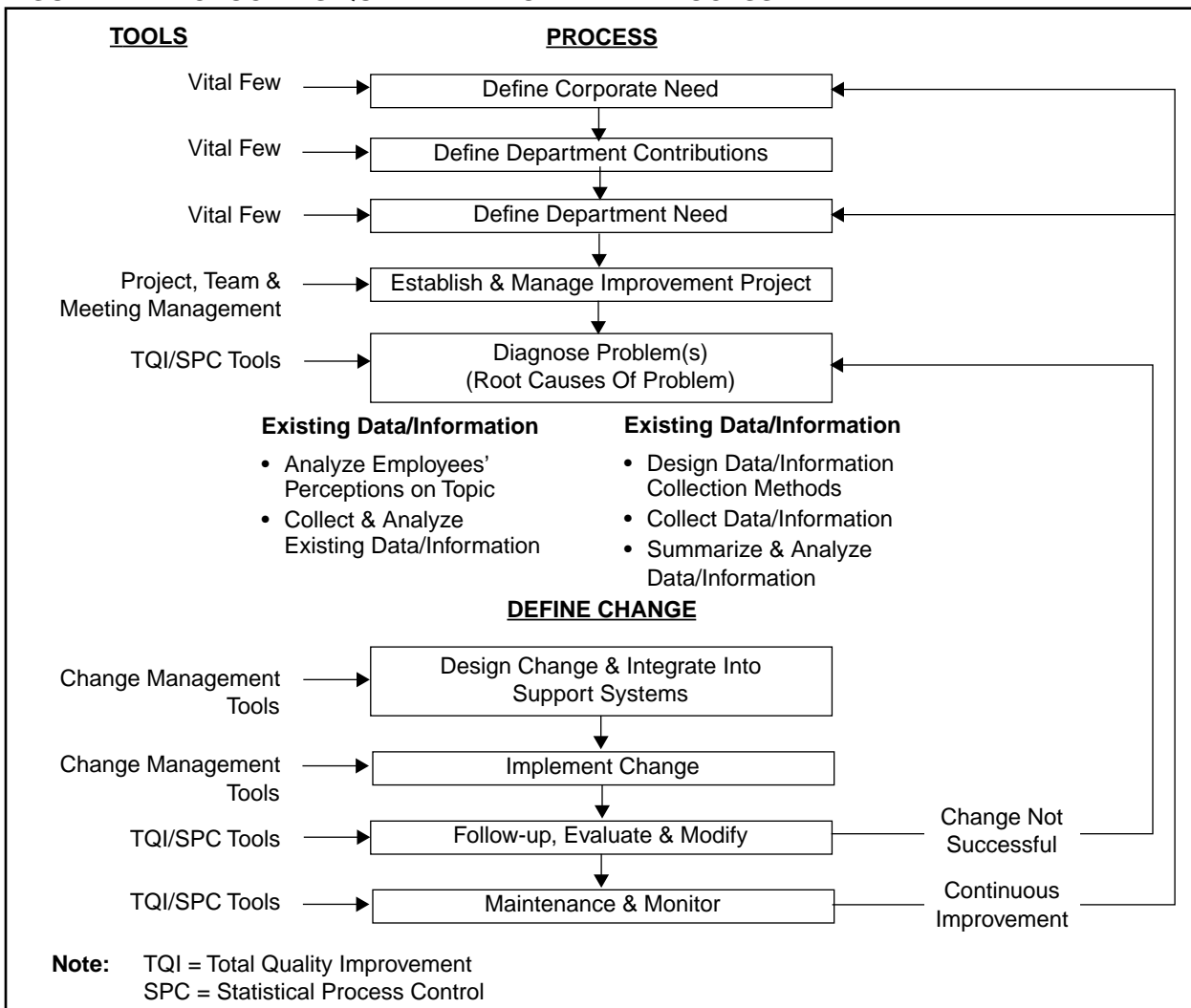
There is a team training program at Microchip that teaches high performance team leadership, and provides training on a set of tools for use. These tools are aimed at improving the teams ability to identify problems, more importantly to identify the real root causes of the problems, to use the Pareto and other techniques to establish the "Vital Few", to brainstorm ideas on solutions, to investigate solutions prior to implementing, to execute implementations smoothly and accurately, to follow-up to ensure that the fix is as expected, and to re-examine the process to see what other improvements are needed. The key elements of this process are shown in the flowchart (Figure 1).

This process lead to improvements in every area of Microchip's EEPROM technology: design, process, product, test, quality, reliability, and yield. Most of these changes are confidential and proprietary, and several have been patented. These can only be discussed here in general terms.

The actual EEPROM cell design has been the subject of repeated studies. There have been improvements made to the original that have increased process tolerance, increased yield, and of course dramatically increased endurance performance. "Repeated studies" sounds as if we did not get it right the first time, but in fact is just an acknowledgment that continuous improvement is essential. We have the best design in the industry today, but the EEPROM technology team is still working on improving it.

The process also has gone through many iterations of improvement. During some of the early studies, it was determined that new equipment with better control was needed to ensure the reproducibility of the product. This equipment was purchased, installed, and carefully monitored based on the recommendation from the team. We saw the expected improvement as a results. There have also been ongoing studies to improve control, reproducibility, and defect reduction in many other areas of the process.

FIGURE 1: MICROCHIP'S QUALITY IMPROVEMENT PROCESS



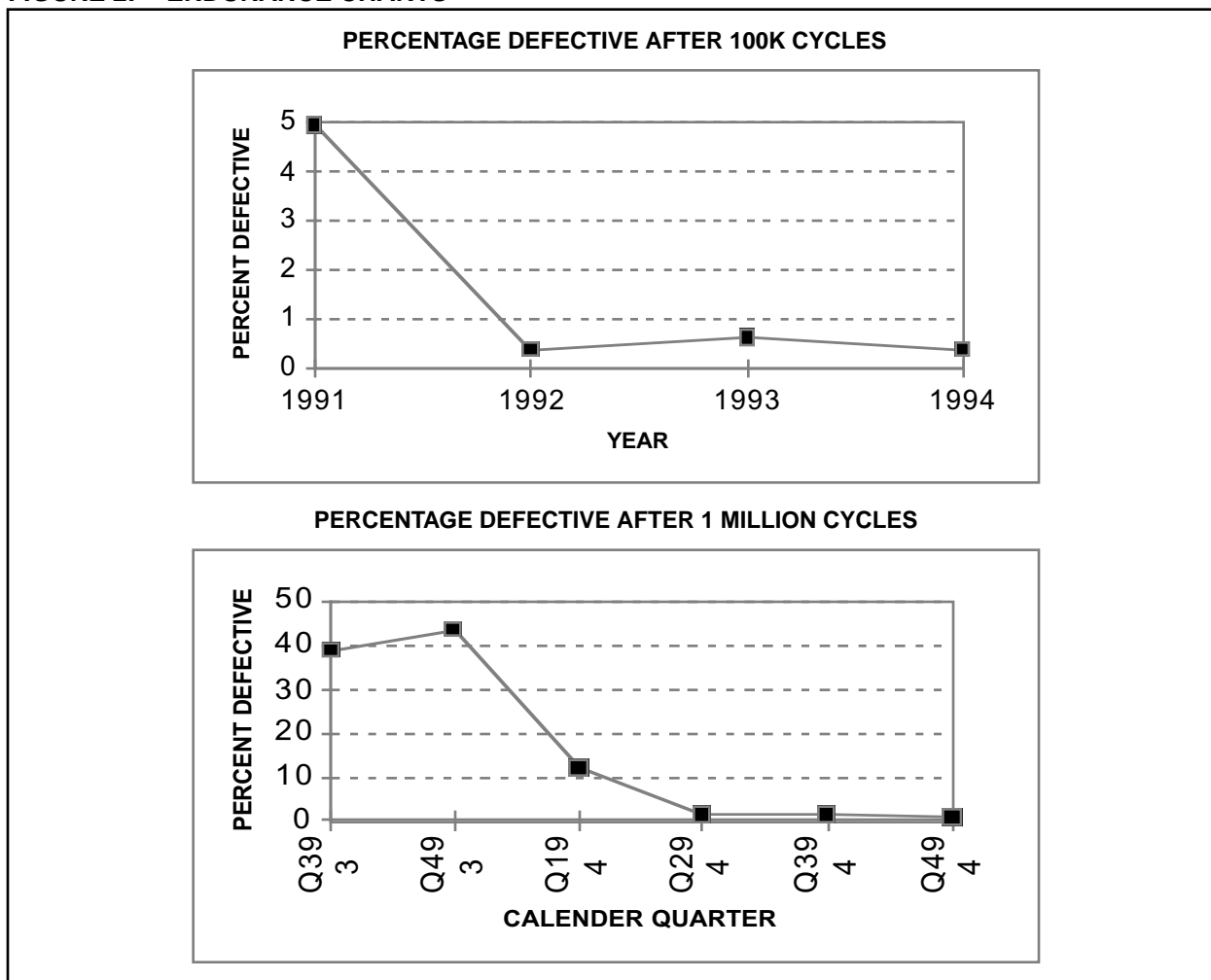
Improvements to the product itself have often shown up in the data sheet. These include reduction in current consumption, or improved timings as a result of improved design as verified by characterization on test equipment. We have greatly expanded the package variety for our Serial EEPROMs as a response to market needs. These activities have all been prioritized by the team with management input, evaluated, and implemented by the team.

The example of a new package introduction highlights the value of the team. A new package requires new test hardware, verification of the test program with the package and hardware, qualification of the package with the reliability group, assurance of acceptable yields based on the yield enhancement engineers involvement, and continued monitoring of the final products outgoing quality. Other companies often have these disciplines separated, with communication by memos back and forth. This is a cumbersome way to do business, and often details are overlooked, or the opportunity for synergy between groups to go the extra step is lost.

Testability is a vital concern for microelectronics. As the feature set grows, the ability to test them accurately can be lost. The introduction of the 64K-bit Serial EEPROM, with its security block and programmable endurance options, presented a concern for testability. By having even the initial design objective discussions in the team environment, with a test engineer, the designers were able to ensure that the new features could be tested thoroughly and cost effectively. This greatly improved our time to market, as well as the quality of the part that reached our customers.

The quality group has been a tremendous contribution to the success of the team. There were initial concerns on some members parts that this group would hold back progress by acting as "policeman" for the status quo. Quite the opposite, the quality group brought the statistical ability to measure the results of proposed changes with accuracy and confidence. So changes that really lead to improvement were actually implemented more quickly than they might have been without quality engineering input at the beginning. The team was also able to move on quickly from proposed changes that did not give measurable improvement as a result of having these measurement tools.

FIGURE 2: ENDURANCE CHARTS



AN603

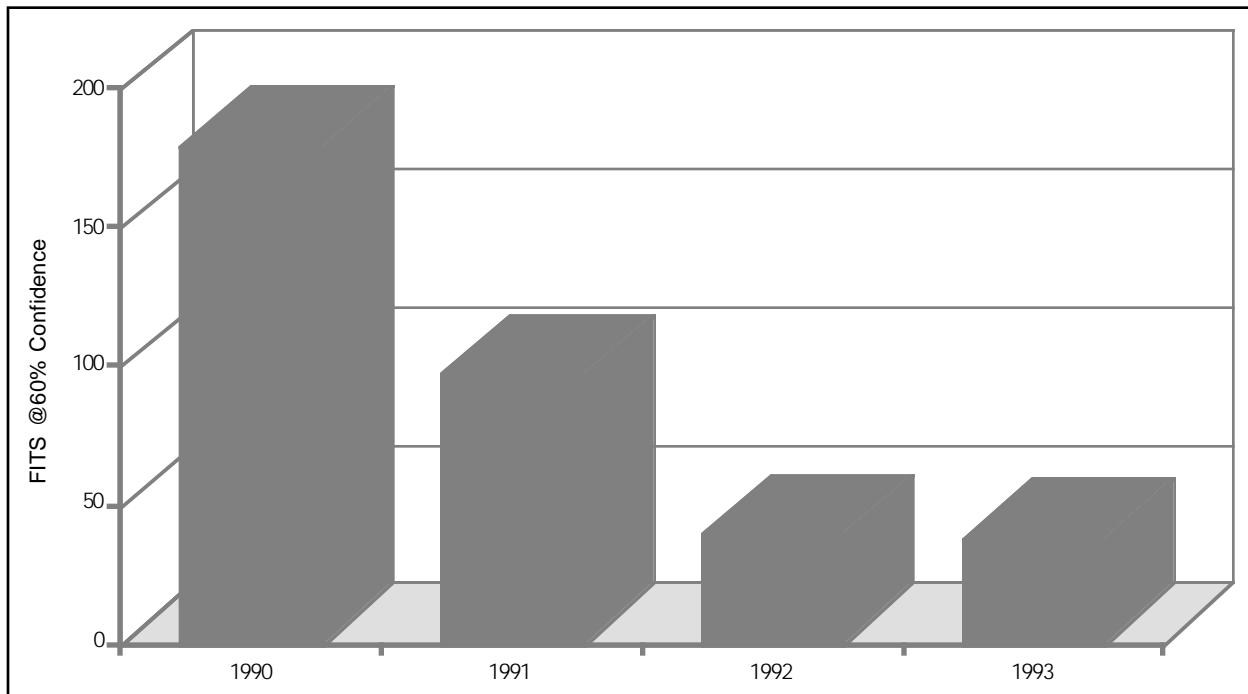
The quality engineering group is also more directly exposed to customer needs than some of the other groups. They heard first hand what some of the areas for improvement were from the field. These inputs were brought directly into the team environment, where the breadth of engineering expertise we had was able to apply Microchip's Quality Improvement Process to them.

The dramatic improvement to the endurance performance of Microchip's Serial EEPROMs was a direct result of the reliability group's involvement with the team. This group was able to establish standard methods for testing and measuring endurance on products. This was the first step in determining that this was another area for continuous improvement. Inputs from the business unit were that this was going to develop into a critical point of competition in the industry. We found out later that our standards were significantly higher than most competitors. The reliability group set up competitive analysis and benchmarking between vendors that put us on a level playing field. We found that Microchip's endurance was much better than we thought! Even so, the Microchip EEPROM Technology team drove programs for improving endurance that gave impressive results (Figure 2). And the elimination of the root causes of endurance problems gave equally impressive gains in other areas of reliability. This is shown in the reduced failure rates for both dynamic life testing and retention bake (Figure 3 and Figure 4).

The group also kept one eye on the bottom line through the years. Yields have always been a concern, since continuous improvement in that area is also vital to the business. The yield enhancement engineers, like the other groups, brought tools with which to measure and identify problems. As with the others, they were dependent on the rest of the team to help find and implement solutions. They also acted as a conduit to the front end teams that were implementing defect density reduction and process control improvements in the manufacturing process across product families. This coordination between the teams led to a whole new level of synergy that allowed other products to benefit from improvements from the EEPROM team, and the EEPROM product line to piggyback on improvements aimed primarily at other areas. These other teams were focused on plant wide areas of continuous improvement: oxide integrity, the poly-silicon modules, the metal modules, defect reduction, wafer handling techniques, etc. The result of this cross functional synergy is shown in the yield trend (Figure 5).

As we grew and learned together, each member developed increasing respect and trust for the others. This was critical to the team's development. Not all of the proposed changes were successful. In the stages of implementing, follow-up, and maintenance, some proposals did not hold up. Either they did not really fix a root cause, or the implementation caused other problems to surface. Only by having established respect and trust was the team able to raise and address these issues quickly.

FIGURE 3: EEPROM DYNAMIC LIFE FAILURE RATE



THE RESULTS

The improvements to both areas that this team has made are spectacular. At the time of formation, the industry standard was 10,000 E/W cycles of endurance, and 4K bits of memory. The Microchip lead the world when the efforts of the team lead to introducing:

- The first 10,000,000 E/W cycle guarantee
- The very first 64K bit Serial EEPROM
- The first to put a 64K bit Serial EEPROM in an 8-pin SOIC package
- The first family of Serial EEPROMs to operate at 1.8V
- The first VESA busing compatible Serial EEPROMs

These improvements are orders of magnitude increases in critical areas of market need. They establish Microchip Technology Inc. as the technical leader in EEPROM technology. This is what the EEPROM Technology team set out to do.

The team's efforts continue today to make sure Microchip remains a technical leader. We have the best products in the market today. But that will not be good enough tomorrow. Continuous improvement is essential.

FIGURE 4: EEPROM RETENTION BAKE FAILURE RATE

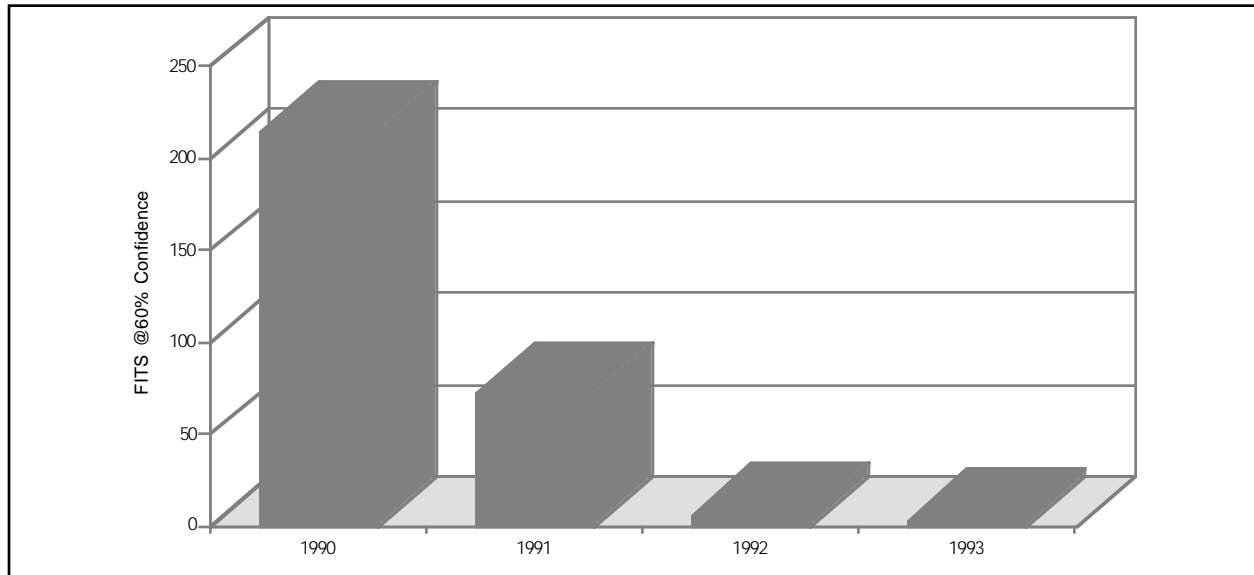
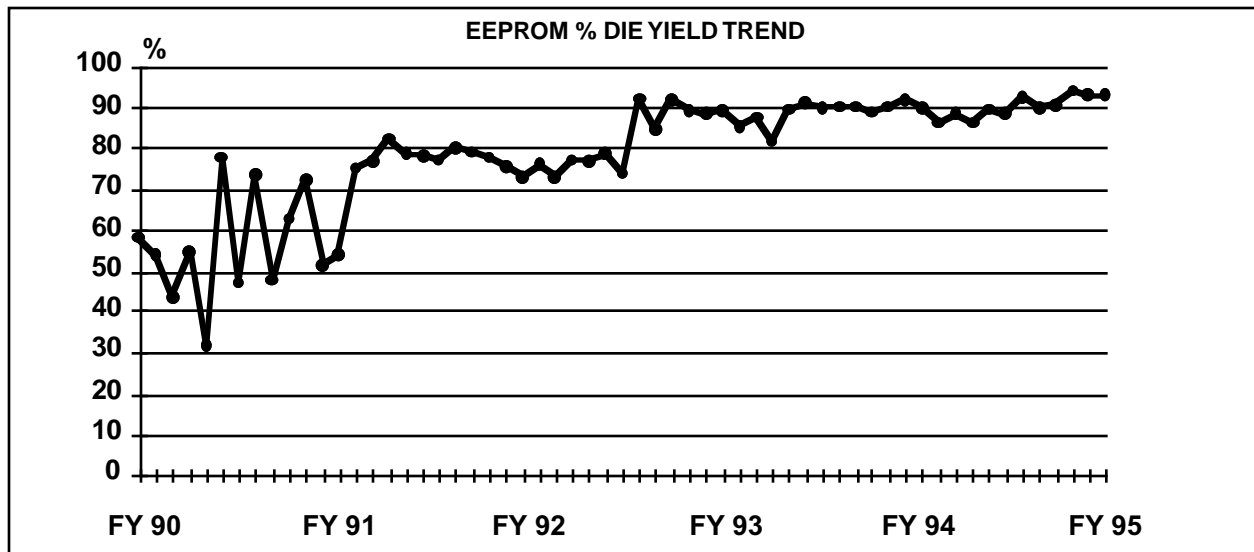


FIGURE 5: WORLD CLASS MANUFACTURING



AN603

NOTES:

SECTION 6

DEVELOPMENT TOOLS

DEVELOPMENT SYSTEMS:

System Support	Development System Selection Chart	6-1
Microchip BBS	Microchip Bulletin Board Service	6-3
Design Kit	Microchip Serial EEPROM Designer's Kit.....	6-5
PICMASTER™	Universal In-Circuit Emulator System with MPLAB IDE.....	6-7
PRO MATE™	Universal Device Programmer	6-13
PICSEE™ Tools	PICSEE Product Development Tools	6-17
PICSTART™-16B1	PIC16CXX Low-Cost Microcontroller Development System	6-25
PICSTART-16C	PIC16CXX Low-Cost Microcontroller Development System	6-27
PICDEM-1	Low-Cost PIC16/17 Demonstration Board.....	6-29
PICDEM-2	Low-Cost PIC16CXX Demonstration Board	6-31

SOFTWARE TOOLS:

<i>fuzzyTECH</i> ®-MP	Fuzzy Logic Development System for PIC16/17.....	6-33
MPASM	Universal PIC16/17 Microcontroller Assembler Software	6-35
MPSIM	PIC16/17 Microcontroller Simulator	6-37
MP-C	C Compiler.....	6-39
Total Endurance™	Microchip Serial EEPROM Endurance Model.....	6-41
TrueGauge™ Tools	TrueGauge Intelligent Battery Management Development Tool.....	6-43



SYSTEM SUPPORT

Development System Selection Chart

PRODUCT	PIC16/17 Microcontroller														ASSP			Memory						
	PIC16C54	PIC16C54A	PIC16C55	PIC16C56	PIC16C57	PIC16C58A	PIC16C60	PIC16C61	PIC16C62	PIC16C63	PIC16C64	PIC16C65	PIC16C71	PIC16C73	PIC16C74	PIC16C84	PIC17C42	PIC17C43	PIC17C44	MTA81010	MTA85xxx	MTA11200	SEEPROM Devices	
Emulator Systems																								
PICMASTER™-16B													X											
PICMASTER-16C																X								
PICMASTER-16D	X	X	X	X	X	X																		
PICMASTER-16E									X		X													
PICMASTER-16F										X		X		X	X									
PICMASTER-16G									X															
PICMASTER-16H							X	X	X															
PICMASTER-17																	X							
PICMASTER-17B																	X	X	X					
Emulator Probe Kits																								
PICPROBE-16B													X											
PICPROBE-16C																X								
PICPROBE-16D	X	X	X	X	X	X																		
PICPROBE-16E									X		X													
PICPROBE-16F										X		X		X	X									
PICPROBE-16G								X																
PICPROBE-16H							X	X	X															
PICPROBE-17A																	X							
PICPROBE-17B																	X	X	X					
Development Kits																								
fuzzyTECH®-MP	X	X	X	X	X	X					X	X		X	X	X								
PICSTART™-16B1	X	X	X	X	X	X	X	X	X				X			X				X	X			
PICSTART-16C										X	X	X	X		X	X								
PRO MATE™	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
PICSEEKIT-81A																				X				
PICSEESTART-81A																				X				
PICSEEKIT-85A																					X			
PICSEESTART-85A																					X			
Serial EEPROM Designer's Kit																								X
TrueGauge™																						X		
PIC16/17 Software Tools: MPASM/MPSIM	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
Total Endurance Software Model																								X
Demonstration Boards																								
PICDEM-1	X	X	X	X	X	X	X	X	X				X			X	X	X	X					
PICDEM-2										X	X	X	X		X	X								

SYSTEM SUPPORT

Compatible & Universal				
Product	Description	PIC16C5X	PIC16CXX	PIC17CXX
MPASM	Universal Assembler	X	X	X
MPSIM	Software Simulator	X	X	X
MP-C	C Compiler	X	X	X
fuzzyTECH	Fuzzy Logic Development Kit	X	X	X
PICSTART	Low Cost Development Kit	X	X	—
PRO MATE	Universal Programmer/Development Kit	X	X	X
PICMASTER	Universal In-Circuit Emulator	X	X	X
MPLAB	Integrated Development Environment	X	X	X



MICROCHIP BBS

Microchip Bulletin Board Service

Get current information and help on Microchip's Bulletin Board Service (BBS)! Microchip wants to provide you with the most responsive service possible. To accomplish this, the systems team monitors the BBS, posting the latest component data and software tool updates, providing technical help and embedded systems insights, and discussing how Microchip products provide project solutions. Extend your technical groups staff with microcontroller and memory experts through Microchip's BBS communication channel.

SYSTEMS INFORMATION AND UPGRADE HOT LINE

The Systems Information And Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits. The Hot Line Numbers are: 1-800-755-2345 for U.S. and most of Canada, and 1-602-786-7302 for the rest of the world.

These phone numbers are also listed on the "Important Information" sheet that is shipped with all development systems. The hot line message is updated whenever a new software version is added to the Microchip BBS, or when a new upgrade kit becomes available.

CONNECTING TO MICROCHIP

Connect worldwide to the Microchip BBS using the CompuServe® communications network. In most cases, a local call is your only expense. The Microchip BBS connection does not use CompuServe membership services, therefore, **you do not need CompuServe membership to join Microchip's BBS.**

There is **no charge** for connecting to the BBS, except for a toll charge to the CompuServe access number, where applicable. You do not need to be a CompuServe member to take advantage of this connection (you never actually log in to CompuServe).

The procedure to connect will vary slightly from country to country. Please check with your local CompuServe agent for details if you have a problem. CompuServe service allow multiple users at baud rates up to 14400 bps.

The following connect procedure applies in most locations.

1. Set your modem to 8-bit, No parity, and One stop (8N1). This is not the normal CompuServe setting which is 7E1.
2. Dial your local CompuServe access number.
3. Depress **<Enter.>** and a garbage string will appear because CompuServe is expecting a 7E1 setting.
4. Type +, depress **<Enter.>** and `Host Name:` will appear.
5. Type **MCHIPBBS**, depress **<Enter.>** and you will be connected to the Microchip BBS.

In the United States, to find CompuServe's phone number closest to you, set your modem to 7E1 and dial (800) 848-4480 for 300-2400 baud or (800) 331-7166 for 9600-14400 baud connection. After the system responds with `Host Name:`, type **NETWORK**, depress **<Enter.>** and follow CompuServe's directions.

For voice information (or calling from overseas), you may call (614) 457-1550 for your local CompuServe number.

USING THE BULLETIN BOARD

The bulletin board is a multifaceted tool. It can provide you with information on a number of different topics.

- Special Interest Groups
- Files
- Mail
- Bug Lists

Special Interest Groups

Special Interest Groups, or SIGs as they are commonly referred to, provide you with the opportunity to discuss issues and topics of interest with others that share your interest or questions. SIGs may provide you with information not available by any other method because of the broad background of the PIC16/17 user community.

There are SIGs for most Microchip systems, including:

- MPASM
- PICMASTER™
- PRO MATE™
- PICSTART™
- Utilities
- Bugs
- MPSIM
- TRUE GAUGE™
- *fuzzyTECH*®-MP
- ASSP
- MTE1122

MICROCHIP BBS

Note: The SIGs provide you with the opportunity to discuss issues and exchange ideas. Technical support and urgent questions should be referred to your local distributor, FAE or sales representative. They are your first level of support.

These groups are monitored by the Microchip staff.

Files

Microchip regularly uses the Microchip BBS to distribute technical information, application notes, source code, errata sheets, bug reports, and interim patches for Microchip systems software products. Users can contribute files for distribution on the BBS. For each SIG, a moderator monitors, scans, and approves or disapproves files submitted to the SIG. No executable files are accepted from the user community in general to limit the spread of computer viruses.

Mail

The BBS can be used to distribute mail to other users of the service. This is one way to get answers to your questions and problems from the Microchip staff, as well as keeping in touch with fellow Microchip users worldwide.

Consider mailing the moderator of your SIG, or the SYSOP, if you have ideas or questions about Microchip products, or the operation of the BBS.

Software Releases

Software products released by Microchip are referred to by version numbers. Version numbers use the form:

`xx.yy.zz <status>`

Where `xx` is the major release number, `yy` is the minor number, and `zz` is the intermediate number. The `status` field displays one of the following categories:

- Alpha
- Intermediate
- Beta
- Released

Production releases are numbered with major, and minor version numbers like:

`3.04 Released`

Alpha, Beta and Intermediate releases are numbered with the major, minor and intermediate numbers:

`3.04.01 Alpha`

Alpha Release

Alpha designated software is engineering software that has not been submitted to any quality assurance testing. In general, this grade of software is intended for software development team access only, but may be sent to selected individuals for conceptual evaluation.

Intermediate Release

Intermediate released software represents changes to a released software system and is designated as such by adding an intermediate number to the version number. Intermediate changes are represented by:

- Bug Fixes
- Special Releases
- Feature Experiments

Intermediate released software does not represent our most tested and stable software. Typically, it will not have been subject to a thorough and rigorous test suite, unlike production released versions. Therefore, users should use these versions with care, and only in cases where the features provided by an intermediate release are required.

Intermediate releases are primarily available through the BBS.

Beta Release

Preproduction software is designated as Beta. Beta software is sent to Applications Engineers and Consultants, FAEs, and select customers. The Beta Test period is limited to a few weeks. Software that passes Beta testing without having significant flaws, will be production released. Flawed software will be evaluated, repaired, and updated with a new revision number for a subsequent Beta trial.

Production Release

Production released software is software shipped with tool products. Example products are PRO MATE, PICSTART, and PICMASTER. The Major number is advanced when significant feature enhancements are made to the product. The minor version number is advanced for maintenance fixes and minor enhancements. Production released software represents Microchip's most stable and thoroughly tested software.

There will always be a period of time when the Production Released software is not reflected by products being shipped until stocks are rotated. You should always check the BBS for the current production release.



DESIGNER'S KIT

Microchip Serial EEPROM Designer's Kit



FEATURES

- Includes everything necessary to begin developing Serial EEPROM-based applications
- Microchip *Total Endurance*™ software model
- Microchip *SEEVAL*™ evaluation and programming board
- Microchip *SEEVAL* software
- Microchip Serial EEPROM handbook
- Microchip Serial EEPROM sample pack
- RS-232 serial cable
- Power supply

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows® 3.1
- VGA monitor
- 386 or 486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/85CXX
- Microchip *Smart Serial*™ 24XX65
- Microchip 3-wire 93CXX/93LCXX series
- Microchip 4-wire 59C11

Designer's Kit

DESCRIPTION

Now designers of Serial EEPROM-based applications can enjoy the increased productivity, reduced time to market, and the ability to create a rock-solid design that only a well-thought-out development system can provide. Microchip's new Serial EEPROM Designer's Kit includes everything necessary to quickly develop a robust and reliable Serial EEPROM-based design and greatly reduce the time required for system integration and hardware/software debug.

The **Total Endurance software model** enables designers to quickly choose the best Serial EEPROM for the specific application and perform trade-off analysis with voltage, temperature, write cycle and other system parameters in order to achieve the desired Erase/Write endurance (specific ppm rate) or product lifetime. Total Endurance is the new standard of excellence in understanding and predicting the Erase/Write endurance of Serial EEPROMs. An on-line endurance tutorial is included, along with hypertext help files.

Microchip's **SEEVAL Serial EEPROM evaluation and programming system** will accept any Microchip Serial EEPROM in DIP package and enable the designer or system integrator to read, write, or erase any byte or the entire array. SEEVAL also provides the following advanced features to aid in system integration and debug:

- Program special user functions like *Smart Serial* configurations
- Hexadecimal display of array contents
- Pre-set or user-defined repeating patterns
- User-configurable functions like continuous read/write, programmable delay, etc.
- Upload/download files between the Serial EEPROM and disk

Another industry first, the **Microchip Serial EEPROM Handbook** provides a plethora of information crucial to the designers of Serial EEPROM-based systems. Along with data sheets on Microchip Serial EEPROMs, this resource provides application notes regarding Erase/Write endurance, interfacing with different protocols and many, many others. A cross-reference and selector guide are also included, plus article reprints and qualification reports on Microchip Serial EEPROMs.

USING SEEVAL AND TOTAL ENDURANCE

Both software packages can be loaded from Windows by choosing FILE RUN and entering SETUP.EXE from the Program Manager. The applications will install themselves; then a double mouse-click will start either application. The first step in either program is to select a device from the device list.

In Total Endurance, the user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

In **SEEVAL**, the user may choose to load a file from disk to program the Serial EEPROM, or read data from the EEPROM and save it to disk. The screen displays the contents of a software buffer. The buffer may be manipulated before programming data to the Serial EEPROM, or data can be written to the Serial EEPROM directly on-line. An area of memory can be highlighted (selected) and programmed with a predefined pattern or user-specified pattern. Alternately, the entire device can be programmed with any repeating pattern.

Both SEEVAL and Total Endurance allow the user to save any configuration as default. This configuration (device and application settings) will then automatically load at boot time.

Order Information:

<u>Description</u>	<u>Part Number</u>
Serial EEPROM Designer's Kit	DV243001



PICMASTER™ System

Universal In-Circuit Emulator System with MPLAB IDE



SYSTEM FEATURES

General:

- Complete Hi-Performance PC-based Microcontroller Development System for the PIC16/17 families.
- For use on PC-compatible 386, 486 and Pentium machines under Microsoft® Windows® 3.X environment (also runs in Windows 95).
- Assembler and Simulator Software, Emulator System, and EPROM Programmer unit, sample kit, and demonstration hardware and software provide a complete microcontroller product development environment.

Windows IDE:

- Built-in full-featured programmer's text editor.
- Project manager keeps track of files, automatically downloads new builds to emulator/simulator and updates symbolic information into break/trace/trigger point settings.
- Customizable tool bar.
- Editor, emulator, simulator modes.
- Status bar shows development mode, editor information, current PC and W register and status register values.
- Extensive on-line help
- Customizable IDE desktop so key mappings, colors, symbol width, and toolbar position can be changed and saved.

PICMASTER™ System

Emulator System:

- Hi-Performance In-Circuit Emulation of Microchip Microcontrollers.
- Real-time instruction emulation.
- Program Memory emulation and memory mapping capability up to 64K words. Instruction execution can be mapped into either emulation memory or user prototype memory.
- Real-time trace memory capture of 40 bits of information for each instruction cycle in an 8Kx40 trace buffer. Trace region can range from 0 to 64K in any address combinations.
- Real-time trace data can be captured and displayed without halting emulation.
- Unlimited number of hardware breakpoints can be set anywhere in the program memory.
- External Break with "AND"/"OR" capability with internal breakpoints.
- Multiprocessor emulation capability. Up to four PICMASTER emulators can be synchronized on a single PC, for multi-processor development.
- Extended 48-bit stopwatch measures execution time.
- Trigger Output available on any range of addresses.
- Full Symbolic Debug Capability. Symbolic display and alter of all register files, special purpose registers, stack registers, and bank registers.
- Selectable Internal Emulator Clock or User Target (Prototype) System Clock.
- User selectable internal or external Power Supply (provided).
- Pass counter for break and trace points.

EPROM Programmer System:

- PRO MATE™ Device Programmer unit for all current PIC16/17 products.
- Operates as a Stand-alone Unit or in Conjunction with a PC-compatible host system.
- Performs READ, PROGRAM, and VERIFY functions in Stand-alone mode.
- PC Host Software provides file display and editing, file transfer to and from programmer unit, device serialization, and program voltage calibration.

Macro Assembler:

- Provides translation of Assembler source code to object code for the PIC16/17 family of microcontrollers.
- Macro-assembly and conditional assembly capability.
- Produces Object files, Listing files, Symbol files, and special files required for symbolic debug with the PICMASTER Emulator System.
- Binary / Hex output formats: INHX8S, INHX8M, INHX16, and PICMASTER.

Simulator:

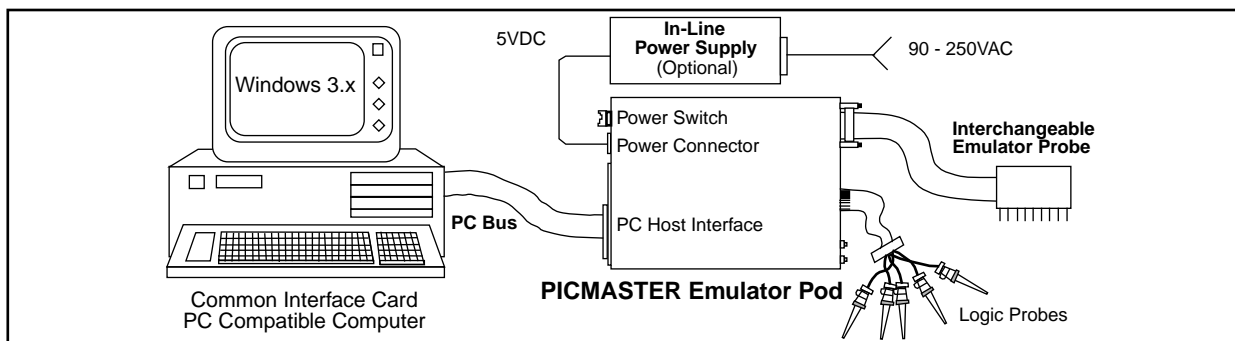
MPSIM is a discrete event software simulator designed to imitate operation of PIC16/17 microcontrollers. It allows the user to debug software that will use any of these microcontrollers. At any instruction boundary, MPSIM also allows the user to examine and/or modify any data area within the processor, or provide external stimulus to any of the pins. The simulator has the exact same look and feel as the PICMASTER emulator but provides the most cost-effective debugging tools. Once operation of the system with the simulator has been mastered, upgrading to the PICMASTER emulator is an easy transition, since the operation is virtually identical.

Demo Board:

The PICDEM Demonstration Board provides a user with a simple hardware tool through which software can be exercised and debugged. A step-by-step tutorial enables first-time users of PICMASTER to become familiar with all the features of the emulator. A generous prototype area allows the user to build additional hardware for their project.

SYSTEM DESCRIPTION

The PICMASTER Universal In-Circuit Emulator System is intended to provide the product development engineer with a complete microcontroller design tool set for all microcontrollers in the PIC16/17 family. The PICMASTER system currently supports the PIC16C54, PIC16C54A, PIC16C55, PIC16C56, PIC16C57 and PIC16C58A at clock frequencies of 20MHz; the PIC16C61, PIC16C62, PIC16C620, PIC16C621, PIC16C622, PIC16C64, PIC16C65, PIC16C71, PIC16C73, PIC16C74, PIC16C84 to 10MHz; and the PIC17C42, PIC17C42A, PIC17C43, and PIC17C44 to 16 MHz.



Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the PICMASTER allows expansion to support all new microcontroller architectures with data and program memory paths to 16 bits.

Provided with the PICMASTER System is the MPLAB integrated development environment, a high performance, real-time In-Circuit Emulator, a microcontroller programmer unit, a macro assembler program, and an Unusually sophisticated simulator program. Sample programs are provided to help quickly familiarize the user with the development system and the PIC16/17 microcontroller families. The MPLAB IDE user interface features a built-in text editor, project management, and full integration with MPASM and ByteCraft's C compiler, MPC, for rapid product development.

A "Quick Start" Product Sample Pak containing user programmable parts is included for additional convenience (only devices supported by the specified probe header).

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS system.

Host System Requirements:

MPLAB and PICMASTER have been designed as a real-time emulation system with advanced features generally found on more expensive development tools. The IBM PC-compatible platform and Windows 3.X environment was chosen to best make these features available to you the end user. To properly take advantages of these features, MPLAB and PICMASTER require installation on a system having the following minimum configuration:

- PC-compatible machine: 386DX, 486 or Pentium with ISA or EISA Bus.
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended).
- MS-DOS® / PC-DOS version 5.0 or greater.
- Microsoft Windows version 3.10 or greater operating in 386 enhanced mode).
- 4 Mbyte RAM.
- One 3.5" floppy disk drive.
- Approximately 4 Mbytes of free hard disk space.
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)
- Microsoft mouse or compatible (highly recommended).

Emulator System Components:

The PICMASTER Emulator Universal System consists primarily of four major components:

- **Host-Interface Card:** The PC Host Interface Card connects the emulator system to a PC

compatible system. This high-speed parallel interface requires a single half-size standard AT / ISA slot in the host system. A 37-conductor cable connects the interface card to the external Emulator Control Pod.

- **Emulator Control Pod:** The Emulator Control Pod contains all emulation and control logic common to all microcontroller devices. Emulation memory, trace memory, event and cycle timers, and trace/breakpoint logic are contained here. The Pod controls and interfaces to an interchangeable target-specific emulator probe via a 14" precision ribbon cable.
- **Target-specific Emulator Probe:** A probe specific to microcontroller family to be emulated is installed on the ribbon cable coming from the control pod. This probe configures the universal system for emulation of a specific microcontroller. Currently, the PIC16C5X family, PIC16CXX family, and the PIC17C4X family microcontrollers are supported. Future microcontroller probes will be available as they are released.
- **MPLAB PC Software:** Host software necessary to control and provide an integrated development environment (IDE) working user interface is the last major component of the system. This software contains a built-in text editor, project manager, PICMASTER emulator drivers, MPSIM simulator drivers (available soon), the software runs in the Windows 3.X environment and provides the user with quick editing, debugging, optimization and control of the system under emulation. MPLAB software is universal to all microcontroller families.

Dynamic Data Exchange (DDE), a feature of Windows 3.X, can be utilized in this version to allow data to be dynamically transferred between two or more Windows programs. Data collected with MPLAB can be automatically transferred to a spreadsheet or database program for further analysis.

Up to four PICMASTER emulators can run simultaneously on the same PC making development of multi-microcontroller systems possible (e.g., a system containing a PIC16CXX processor and a PIC17CXX processor).

PRO MATE Device Programmer:

The PRO MATE Programmer system included in the PICMASTER Development System provides the product developer with the ability to program (transfer) the developer's software into PIC16/17 microcontrollers.

The programmer unit comes complete with accessories for use with a PC host computer. Supplied are interface cables and connectors to a standard PC serial port, a power supply unit, and host operating software.

PICMASTER™ System

The PRO MATE Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user.

STAND-ALONE MODE

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY:

VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM:

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ:

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE

When the PRO MATE is connected to a host PC system, many more options and conveniences are available to the user. Host mode allows full interactive control over the PRO MATE unit. A full screen, user-friendly software program is provided to fully assist the user.

As in stand-alone mode, parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified. In addition, other features available in host-mode are:

Editing

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program and fuse data can be loaded and saved to DOS disk files. Files generated by the Assembler program are directly loadable into programmer memory.

VDD and VPP Adjust

The programming environment voltage settings of VDD max, VDD min, and VPP can be set and altered only on PC host mode. The voltage settings allow the user to

program the part in the environment that the part will be used. The part will be programmed at VDD max and verified at VDD min. VPP is the programming voltage.

PICMASTER PROBE Specifications

Table 1 shows the current probe specifications for the PICMASTER In-Circuit Emulator. The devices are supported regardless of program memory type (ROM, EPROM or EEPROM), process technology or voltage range. That is, selecting the PROBE that supports the PIC16C54 (Probe-16D) also supports the PIC16CR54, PIC16C54A and the PIC16LC54A devices. The probe would also support other variations as they become available (such as PIC16CR54A).

TABLE 1: PICMASTER PROBE SPECIFICATIONS

PICMASTER Probe	Devices Supported	Probe	
		Maximum Frequency	Operating Voltage
PROBE-16B	PIC16C71	10 MHz	4.5V - 5.5V
PROBE-16C	PIC16C84	10 MHz	4.5V - 5.5V
PROBE-16D	PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C58A	20 MHz	4.5V - 5.5V
PROBE-16E	PIC16C64 PIC16C62	10 MHz	4.5V - 5.5V
PROBE-16F	PIC16C63 PIC16C65 PIC16C74 PIC16C73	10 MHz	4.5V - 5.5V
PROBE-16G	PIC16C61	10 MHz	4.5V - 5.5V
PROBE-16H	PIC16C620 PIC16C621 PIC16C622	10 MHz	4.5V - 5.5V
PROBE - 17B	PIC17C42 PIC17C43 PIC17C44	16 MHz	4.5V - 5.5V

Sales and Support

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the factory or the listed sales offices.

PART NUMBER	DESCRIPTION
EM167011	Complete PICMASTER-16B System for PIC16C71
EM167012	Complete PICMASTER-16B System for PIC16C71 without Programmer
EM167013	Complete PICMASTER-16C System for PIC16C84
EM167014	Complete PICMASTER-16C System for PIC16C84 without Programmer
EM167015	Complete PICMASTER-16D System for PIC16C5X
EM167016	Complete PICMASTER-16D System for PIC16C5X without Programmer
EM167017	Complete PICMASTER-16E System for PIC16C64
EM167018	Complete PICMASTER-16E System for PIC16C62, PIC16C64 without Programmer
EM167019	Complete PICMASTER-16F System for PIC16C65, PIC16C74/C73
EM167020	Complete PICMASTER-16F System for PIC16C63, PIC16C65, PIC16C74/C73 without Programmer
EM167021	Complete PICMASTER-16G System for PIC16C61
EM167022	Complete PICMASTER-16G System for PIC16C61 without Programmer
EM167023	Complete PICMASTER-16H System for PIC16C62X
EM167024	Complete PICMASTER-16H System for PIC16C62X without Programmer
EM177007	Complete PICMASTER-17B System for PIC17C42
EM177008	Complete PICMASTER-17B System for PIC17C42, PIC17C43, PIC17C44 without Programmer

PICMASTER™ System

NOTES:



PRO MATE™

Universal Device Programmer

SYSTEM FEATURES

Device Programmer System:

- PRO MATE Programmer unit for the PIC16C5X, PIC16CXX, PIC17CXX Microcontroller family.
- Operates as a Stand-alone Unit or in Conjunction with a PC Compatible host system.
- READS, PROGRAMS, and VERIFIES in Stand-alone mode.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit
- Communicates with PC via RS-232
- Modular socket modules provide easy migration from one PIC16/17 microcontroller product to another.

SYSTEM DESCRIPTION

PRO MATE Programmer:

The PRO MATE Programmer system provides the product developer with the ability to program user software into PIC16C5X, PIC16CXX, PIC17CXX CMOS microcontrollers.

PRO MATE is also supplied with a discrete event software simulator (MPSIM) and a Universal PIC16/17 Macro assembler (MPASM).

The programmer unit comes complete with accessories to be used with the PC host computer. Supplied are interface cables and connectors to a standard PC serial port, a universal input power supply unit, and host operating software.

The PRO MATE Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user.

The modular socket module design allows users to easily migrate between PIC16/17 devices at the lowest possible cost.



STAND-ALONE MODE

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY

VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE

The PRO MATE provides a very user friendly user interface which allows complete control over the programming session.

The PRO MATE host software is a DOS windowed environment with full mouse support to allow the user to point and click when entering commands.

The Host Software communicates with the PRO MATE via the serial port of the PC. Any of the four (COM 1-4) ports may be used. The communication is done at 19200 baud to insure fast throughput. Communication will be established with the PRO MATE Device Programmer prior to any transfers taking place.

Serialization is done by generating a serialization file, and then using that file to serialize locations in the PIC16/17 microcontroller. Once a serialization file is generated, it may be used over different programming sessions. Serial numbers are automatically marked as used when a PIC16/17 is programmed successfully with that serial number.

Complete control over the programming environment is also provided. Control over the programming and verify voltage of VDD insures that the Microcontroller will perform in the desired environment. Programming (VPP) voltage is also adjustable to insure complete compatibility with future programming algorithms.

Macro Assembler:

- Provides translation of Assembler source code to object code for all PIC16/17 microcontroller product family.
- Macro-Assembly capability.
- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16/17 Emulator System.
- Output formats: INHX8S and INHX8M.

Simulator:

- Instruction-level Simulator of the PIC16/17 microcontroller product family.
- For PC-compatible systems running the MS-DOS® operating system.
- Full screen simulation user interface.
- Symbolic debugging capability.
- I/O stimulus input capability.

PRO MATE SOCKET MODULE CROSS-REFERENCE

	Pin Count	DIP	SOIC	SSOP	PLCC	MQFP	TQFP
PIC16C54	18/20	AC164001	AC164002	AC164015	—	—	—
PIC16C54A	18/20	AC164001	AC164002	AC164015	—	—	—
PIC16CR54	18/20	AC164001	AC164002	AC164015	—	—	—
PIC16CR54A	18/20	AC164001	AC164002	AC164015	—	—	—
PIC16C55	28	AC164001	AC164002	AC164015	—	—	—
PIC16C56	18/20	AC164001	AC164002	AC164015	—	—	—
PIC16C57	28	AC164001	AC164002	AC164015	—	—	—
PIC16CR57A	28	AC164001	AC164002	AC164015	—	—	—
PIC16C58A	18/20	AC164001	AC164002	AC164015	—	—	—
PIC16C620	18/20	AC164010	AC164010	AC164018	—	—	—
PIC16C621	18/20	AC164010	AC164010	AC164018	—	—	—
PIC16C622	18/20	AC164010	AC164010	AC164018	—	—	—
PIC16C61	18	AC164010	AC164010	—	—	—	—
PIC16C62	28	AC164012	AC164017	—	—	—	—
PIC16C63	28	AC164012	AC164017	—	—	—	—
PIC16C64	40/44	AC164012	—	—	AC164013	AC164014	AC164020
PIC16C65	40/44	AC164012	—	—	AC164013	AC164014	—
PIC16C71	18	AC164010	AC164010	—	—	—	—
PIC16C73	28	AC164012	AC164017	—	—	—	—
PIC16C74	40/44	AC164012	—	—	AC164013	AC164014	AC164020
PIC16C84	18	AC164010	AC164010	—	—	—	—
PIC17C42	40/44	AC174001	—	—	AC174002	AC174004	—
PIC17C43	40/44	AC174001	—	—	AC174002	—	AC174005
PIC17C44	40/44	AC174001	—	—	AC174002	—	AC174005

SALES AND SUPPORT

To order or obtain information (e.g., pricing or delivery), please use listed part numbers and refer to listed sales offices.

Programmer Part Number	Description
DV007001	Programmer Kit as described above
Socket Part Number	Description
AC164001	PIC16C54 through C58A 18- & 28-Lead PDIP Socket Module
AC164002	PIC16C54 through C58A 18- & 28-Lead SOIC Socket Module
AC164010	PIC16C61, PIC16C62X, PIC16C71, PIC16C84, 18-Lead PDIP/SOIC Socket Module
AC164012	PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C73, PIC16C74, 40-Lead PDIP Socket Module
AC164013	PIC16C64, PIC16C65, PIC16C74, 44-Lead PLCC Socket Module
AC164014	PIC16C64, PIC16C65, PIC16C74, 44-Lead PQFP Socket Module
AC164015	PIC16C54 through PIC16C58A, 20 and 28-Lead SSOP Socket Module
AC164017	PIC16C62, PIC16C63, PIC16C73, 28-Lead SOIC Socket Module
AC164018	PIC16C62X 20-Lead SSOP Socket Module
AC164020	PIC16C64, PIC16C74, 44-Lead TQFP Socket Module
AC174001	PIC17C42, PIC17C43, PIC17C44, 40-Lead PDIP Socket Module
AC174002	PIC17C42, PIC17C43, PIC17C44, 44-Lead PLCC Socket Module
AC174004	PIC17C42 44-Lead QFP Socket Module
AC174005	PIC17C43, PIC17C44, 44-Lead TQFP Socket Module

PRO MATE CROSS REFERENCE BY SOCKET PART NUMBER

Device	Pin Count	AC164001 DIP	AC164002 SOIC	AC164010 DIP/SOIC	AC164012 DIP	AC164013 PLCC	AC164014 MQFP	AC164015 SSOP	AC164017 SOIC	AC164018 SSOP	AC174001 DIP	AC174002 PLCC	AC172004 MQFP
PIC16C54	18/20	✓	✓				✓						
PIC16C54A	18/20	✓	✓				✓						
PIC16CR54	18/20	✓	✓				✓						
PIC16C55	28	✓	✓				✓						
PIC16C56	18/20	✓	✓				✓						
PIC16C57	28	✓	✓				✓						
PIC16CR57A	28	✓	✓				✓						
PIC16C58A	18/20	✓	✓				✓						
PIC16C61	18		✓										
PIC16C62	28			✓				✓					
PIC16C63	28			✓				✓					
PIC16C64	40/44			✓	✓	✓							
PIC16C65	40/44			✓	✓	✓							
PIC16C620	18/20		✓						✓				
PIC16C621	18/20		✓						✓				
PIC16C622	18/20		✓						✓				
PIC16C71	18		✓										
PIC16C73	28			✓				✓					
PIC16C74	40/44			✓	✓	✓							
PIC16C84	18		✓										
PIC17C42	40/44									✓	✓		✓
PIC17C43	40/44									✓	✓		
PIC17C44	40/44									✓	✓		



PICSEE™ TOOLS

PICSEE Product Development Tools

INTRODUCTION

The PICSEE Development Systems provide the product development engineer with cost effective and timely design tool solutions for the MTA8XXXX family of 8-bit CMOS microcontrollers with Serial EEPROM. They are designed specifically for the MTA8XXXX family. These tools work in conjunction with existing hardware and software design tools for the PIC16CXX microcontroller family. This allows the development engineer to efficiently implement systems utilizing these multichip modules with a minimal learning curve and capital investment.

PICSEEKIT-81A — P/N AC812001

- Supports MTA81010
- Programming Adapters for PDIP and SOIC packages
- Daughter card for PICPROBE-16D
- I²C™ bus Serial Communication Application Software

This kit supports the MTA81010 multichip module. It contains programming adapters, a PICMASTER™ emulator daughter board and MTA81010 product samples in a 28-lead PDIP. The kit also includes an MS-DOS®, PC-compatible 3.5-inch software diskette that contains example source code for implementing the I²C serial bus protocol to communicate with a Serial EEPROM. Documentation is provided for all of the included hardware and software.

Programming Support

Two programming adapters are provided to allow the MTA81010's internal program EPROM as well as its data EEPROM to be programmed on existing programmers. Any programmer that supports Microchip's PIC16C54 can program the MTA81010's internal EPROM. Also, any programmer that supports Microchip's 24LC01B Serial EEPROM can program the MTA81010's internal Serial EEPROM. There is one adapter for MTA81010's in DIP packages and another for SOIC packages. Both DIP and SOIC programming adapters interface to programmers via a 300 mil DIP header.

Emulation Support

The emulator daughter board allows the developer to use Microchip's PICMASTER in-circuit emulator to emulate the MTA81010 Microcontroller with Serial EEPROM. This daughter board replaces Microchip's PIC16C5X Emulator Probe Header (P/N AC162009) emulator probe to support the MTA81010. The daughter board provides the required translation from a PIC16C54 pin out to the MTA81010 pin out. It also contains a discrete 24LC01B Serial EEPROM to provide the same functions as the MTA81010's internal EEPROM. This provides a cost-effective emulation solution to customers who may wish to purchase a PICMASTER in-circuit emulator or those that already have a PICMASTER.

Software Support

Example source code for I²C Bus communication with a Serial EEPROM is included in the kit. This pre-tested code can be used directly or modified by the developer to meet their specific needs. This example code is provided royalty free and license free.

PICSEESTART-81A — P/N DV813001

- Complete Low-Cost Development Solution for MTA81010
- Combines PICSEEKIT-81A (AC812001) and PICSTART-16B1 (DV163003)
- MPASM Assembler
- MPSIM Simulator
- Low-Cost Programmer
- Programming Adapter Sockets
- I²C Bus Applications Software

This kit combines the PICSEEKIT (P/N AC812001) with a PICSTART™-16B1 (P/N DV163003) to form a complete low-cost development system for the MTA81010 multichip module. It is designed to support the MTA81010 during the software development and initial prototype phases of new product development. It contains tools for software development and debugging, as well as programmer for programming the MTA81010's internal EPROM program memory. For a more detailed description, please refer to the PICSEEKIT P/N AC812001 and PICSTART P/N DV163003 product descriptions.

PICSEEKIT-85A — P/N AC852001

- Supports MTA85XXX
- Programming Adapter for SSOP package
- PICMASTER Adapter Socket
- I²C bus Serial Communication Application Software

This kit supports the MTA85XXX multichip module. It contains a programming adapter, a PICMASTER adapter socket, and product samples. Also included is an MS-DOS, PC-compatible 3.5-inch software diskette that contains example source code for implementing the I²C serial bus protocol to communicate with a Serial EEPROM. Documentation is provided for all of the included hardware and software.

Programming Support

A programming adapter socket is provided to allow the MTA85XXX's internal program EPROM as well as its data EEPROM to be programmed on existing programmers. Any programmer that supports Microchip's PIC16C54 can program the MTA85XXX's internal EPROM. Also, any programmer that supports Microchip's 24LC01B and 24LC02B Serial EEPROM can program the MTA85XXX's internal Serial EEPROM. There is one adapter for the SSOP package for MTA85XXX.

Emulation Support

An adapter socket allows the developer to use Microchip's PICMASTER in-circuit emulator to emulate the MTA85XXX Microcontroller with Serial EEPROM. The adapter socket provides the required translation from a PIC16C54A/58A pin out to the MTA85XXX pin out. It also contains a discrete 24LC02B Serial EEPROM to provide the same functions as the MTA85XXX's internal EEPROM. This provides a cost-effective emulation solution to customers who may wish to purchase a PICMASTER in-circuit emulator or those that already have a PICMASTER.

Software Support

Example source code for I²C Bus communication with a Serial EEPROM is included in the kit. This pre-tested code can be used directly or modified by the developer to meet their specific needs. This example code is provided royalty free and license free.

PICSEESTART-85A — P/N DV853001

- Complete Low-Cost Development Solution for MTA85XXX
- Combines PICSEEKIT-85A (AC852001) and PICSTART-16B1 (DV163003)
- MPASM Assembler
- MPSIM Simulator
- Low-Cost Programmer
- Programming Adapter Sockets
- I²C Bus Applications Software

This kit combines the PICSEEKIT (P/N AC852001) with a PICSTART-16B1 (P/N DV163003) to form a complete low-cost development system for the MTA85XXX multichip module. It is designed to support the MTA85XXX during the software development and initial prototype phases of new product development. It contains tools for software development and debugging, as well as programmer for programming the MTA85XXX's internal EPROM program memory. For a more detailed description, please refer to the PICSEEKIT-85A P/N AC852001 and PICSTART-16B1 P/N DV163003 product descriptions.

FIGURE 1: PICSEKIT-81A INTRODUCTION DESIGN KIT



Description of Contents

1. PICSEE PDIP and SOIC to PIC16C54 or 24LC01B Programming Adapter Sockets
2. Header Interface for PICMASTER-16D and PICPROBE-16D
3. Serial EEPROM Example Software Disk
4. MTA81010 Product Samples
5. 8- and 18-Pin Programming Adapter Plugs
6. Complete Systems Documentation

FIGURE 2: PICSEESTART-81A DEVELOPMENT KIT



Description of Contents

1. PICSEE PDIP and SOIC to PIC16C54 or 24LC01B Programming Adapter Sockets
2. Header Interface for PICMASTER-16D and PICPROBE-16D
3. Serial EEPROM Example Software Disk
4. MTA81010/PIC16CXX Product Samples
5. PIC16CXX Device Programmer Board
6. PIC16CXX Assembler, Simulator and Host Software
7. 8- and 18-Pin Programming Adapter plug
8. Power Supply
9. RS-232 Cable
10. Complete Systems Documentation

FIGURE 3: PICSEKIT-85A DEVELOPMENT KIT



Description of Contents

1. 20-Lead SSOP Programming Adapter Socket
2. PICMASTER Adapter Socket
3. Serial EEPROM Communications Software Disk
4. Product Samples
5. 8- and 18-pin Programming Adapter Plugs
6. Complete Systems Documentation

FIGURE 4: PICSEESTART-85A DEVELOPMENT KIT



Description of Contents

1. 20-lead SSOP Programming Adapter Socket
2. PICMASTER Adapter Socket
3. Serial EEPROM Communications Software Disk
4. Product Samples
5. PIC16CXX Device Programmer Board
6. PIC16/17 Assembler, Simulator, and Host Software
7. 8- and 18-pin Programming Adapter Plugs
8. Power Supply
9. RS-232 Cable
10. Complete Systems Documentation

NOTES:

SALES AND SUPPORT

To order or to obtain information (e.g., on pricing or delivery), please use the listed part numbers, and refer to the listed sales offices.

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
AC812001	PICSEEKIT-81A FOR MTA81010
DV813001	PICSEESTART-81A for MTA81010
AC814003	PDIP PROGRAMMING SOCKET SOIC PROGRAMMING SOCKET
AC852001	PICSEEKIT-85A FOR MTA85XXX
AC854001	20-LEAD SSOP PROGRAMMING ADAPTER SOCKET AND ADAPTER PLUGS ONLY
DV853001	PICSEESTART-85A FOR MTA85XXX



PICSTART™-16B1

PIC16CXX Low-Cost Microcontroller Development System

SYSTEM FEATURES

EPROM Programmer System:

- EPROM Development Programmer unit for the PIC16C5X and selected PIC16CXX Microcontroller family members. Supports PIC16C54, PIC16C54A, PIC16C55, PIC16C56, PIC16C57, PIC16C58A, PIC16C61, PIC16C62X, PIC16C71, PIC16C84.
- Operates with a PC-compatible host system.
- READS, PROGRAMS, VERIFIES EPROM Memory.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit.
- Universal power supply
- RS-232 interface cable

Macro Assembler:

- Provides translation of Assembler source code to object code for all PIC16CXX microcontroller product family.

- For PC-compatible systems running the MS-DOS® operating system.
- Macro-Assembly capability.
- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16CXX Emulator System.
- Output formats: INHX8S and INHX8M.

Simulator:

- Instruction-level Simulator of the PIC16CXX microcontroller product family.
- For PC-compatible systems running the MS-DOS operating system.
- Full screen simulation user interface.
- Symbolic debugging capability.
- I/O stimulus input capability.

“Quick Start” Sample Kit:

- Provides the User / Developer with a sample kit of PIC16CXX parts for initial prototype use.



PICSTART™-16B1

SYSTEM DESCRIPTION

The PICSTART-16B1 Development System provides the product development engineer with an alternative low-cost introductory microcontroller design tool set for the PIC16CXX family where full real-time emulation is not required. The equipment in the PICSTART-16B1 system operates on any PC compatible machine running the MS-DOS/PC-DOS operating system.

Provided in the System is an MS-DOS-based Software Simulator program (MPSIM), a microcontroller EPROM programmer, and a macro assembler program (MPASM).

Sample software programs to be run on the simulator are provided to help the user to quickly become familiar with the development system and the PIC16CXX microcontroller line.

The user need only provide his or her own preferred text editor and the system is ready for development of end products using the PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C58A, PIC16C61, PIC16C62X, PIC16C71 or PIC16C84 microcontrollers.

A "Quick Start" PIC16CXX Product Sample Pak containing user programmable parts is also included.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS.

PICSTART-16B1 Development Programmer:

The Microchip device programmer system included in the PICSTART-16B1 Development System provides the product developer with the ability to program user software into PIC16CXX EPROM microcontrollers. It is designed to be a development programmer and not recommended for use in a production environment.

The programmer unit connects to a standard PC serial port.

A full screen, user-friendly software program is provided for full interactive control over the programmer. Parts may be Read, Programmed, Blank checked and Verified. Also, all fuses and ID locations may be specified.

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program data can be loaded and saved to DOS disk files. Files generated by the MPASM Assembler program are directly loadable into programmer memory.

MPSIM Simulator:

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging PIC16/17 assembler code.

The simulator is meant for use with smaller projects not requiring precise, more extensive development equipment. Many applications can be developed by using a simulator program alone.

The MPSIM Simulator has the following features to assist in the debugging of software/firmware for the user.

Program Load/Save

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display & Alter

Provisions are made to display and alter Program Memory, Register Files, and status register bits. Also simulator information such as cycle times, elapsed time, and step count can be displayed.

Utility Functions

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurrences of instructions, register use, and ASCII data.

Disassembler

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Symbolic Debugging

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace

During program execution, address ranges, registers, register contents, and others can be traced.

Breakpoints

The user may specify up to 512 breakpoints at any one time.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part number, and refer to the listed sales offices.

PART NUMBER
DV163003

DESCRIPTION
PICSTART-16B1 DEVELOPMENT SYSTEM



PICSTART™-16C

PIC16CXX Low-Cost Microcontroller Development System

SYSTEM FEATURES

EPROM Programmer System:

- EPROM Programmer unit for the PIC16CXX Microcontroller family. Supports the PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C73 and the PIC16C74.
- Operates with a PC-compatible host system.
- READS, PROGRAMS, VERIFIES EPROM Memory.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit.
- Universal power supply
- RS-232 interface cable

Macro Assembler:

- Provides translation of Assembler source code to object code for all PIC16CXX microcontroller product family.
- Macro-Assembly capability.

- For PC-compatible systems running the MS-DOS® operating system.
- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16CXX Emulator System.
- Output formats: INHX8S and INHX8M.

Simulator:

- Instruction-level Simulator of the PIC16/17 microcontroller product family.
- For PC-compatible systems running the MS-DOS operating system.
- Full screen simulation user interface.
- Symbolic debugging capability.
- I/O stimulus input capability.

“Quick Start” Sample Kit:

- Provides the User / Developer with a sample kit of the supported PIC16CXX parts for initial prototype use.



PICSTART™-16C

SYSTEM DESCRIPTION

The PICSTART-16C Development System provides the product development engineer with an alternative low-cost introductory microcontroller design tool set for the PIC16CXX family where full real-time emulation is not required. The equipment in the PICSTART-16C system operates on any PC compatible machine running the MS-DOS/PC-DOS operating system.

Provided in the System is an MS-DOS-based Software Simulator program (MPSIM), a microcontroller EPROM programmer, and a macro assembler program (MPASM).

Sample software programs to be run on the simulator are provided to help the user to quickly become familiar with the development system and the PIC16CXX microcontroller line.

The user need only provide his or her own preferred text editor and the system is ready for development of end products using the PIC16C64, PIC16C65, PIC16C73, or the PIC16C74.

A "Quick Start" PIC16CXX Product Sample Pak containing user programmable parts is also included.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS.

PICSTART-16C Development Programmer:

The Microchip device programmer system included in the PICSTART-16C Development System provides the product developer with the ability to program user software into PIC16CXX EPROM microcontrollers. It is designed to be a development programmer and not recommended for use in a production environment.

The programmer unit connects to a standard PC serial port.

A full screen, user-friendly software program is provided for full interactive control over the programmer. Parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified.

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program data can be loaded and saved to DOS disk files. Files generated by the MPASM Assembler program are directly loadable into programmer memory.

MPSIM Simulator:

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging PIC16/17 assembler code.

The simulator is meant for use with smaller projects not requiring precise more extensive development equipment. Many applications can be developed by using a simulator program alone.

The MPSIM Simulator has the following features to assist in the debugging of software/firmware for the user.

Program Load/Save

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display & Alter

Provisions are made to display and alter Program Memory, Register Files, and status register bits. Also simulator information such as cycle times, elapsed time, and step count can be displayed.

Utility Functions

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurrences of instructions, register use, and ASCII data.

Disassembler

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Symbolic Debugging

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace

During program execution, address ranges, registers, register contents, and others can be traced.

Breakpoints

The user may specify up to 512 breakpoints at any one time.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part number, and refer to the listed sales offices.

PART NUMBER

DV163002

DESCRIPTION

PICSTART-16C DEVELOPMENT SYSTEM

Low-Cost PIC16/17 Demonstration Board

PRODUCT INFORMATION

The PICDEM-1 is a simple board which demonstrates the capabilities of several Microchip microcontrollers. The microcontrollers supported are: PIC16C5X (PIC16C54 to PIC16C58), PIC16C62X, PIC16C61, PIC16C62X, PIC16C71, PIC16C84, PIC17C42, PIC17C43 and PIC17C44. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. The users can program the samples (one each of PIC17C42, PIC16C71 and PIC16C55) provided with the PICDEM-1, on a PRO MATE™ or PICSTART™ programmer and easily debug/test the sample code, or the user can connect the PICDEM-1 with the PICMASTER™ emulator and download the sample code to the emulator and debug/test the code. Additionally, a generous 200-hole prototype area is available for the user to build some additional hardware and connect it to the microcontroller socket(s).

FEATURES:

Hardware:

- 40-pin, 28-pin and 18-pin Precision sockets for all supported microcontrollers.
- On board +5V regulator and filter rectifier for direct input from 9V AC/DC wall adapter.
- RS-232 socket and associated hardware for direct connection to RS-232 interface.
- 5K pot to simulate analog input for PIC16C71.
- Three push button Key for external stimulus and RESET.
- Eight bright LEDs connect to PORTB, help in displaying 8-bit binary values on PORTB.
- Socket for "canned" crystal Oscillator.
- Unpopulated holes provided for Xtal connection
- Jumper to disconnect on board RC Oscillator.
- 200-hole prototype area for user's hardware.



PICDEM-1

Software:

- Program for PIC16C71 to demonstrate on-chip A/D features.
- Program for PIC16C84 to demonstrate on-chip EEPROM.
- Program for PIC17C42 to demonstrate on-chip USART.
- Program for PIC16C5X to demonstrate key input capability.
- All demo programs supplied on 3.5" disk,
- Additional programs available on Microchip's BBS.

DOCUMENTATION

- A comprehensive User's Guide with easy to follow step-by-step Getting Started and a Tutorial.
- Schematics for the entire circuit.

SAMPLES

Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:

- PIC17C42
- PIC16C71
- PIC16C55

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER

DM163001

DESCRIPTION

Low-cost Demonstration Board for
PIC16C5X, PIC16C61, PIC16C62X, PIC16C71,
PIC16C84, PIC17C42, PIC17C43 and PIC17C44.

Low-Cost PIC16CXX Demonstration Board

PRODUCT INFORMATION

The PICDEM-2 is a simple board which demonstrates the capabilities of several Microchip microcontrollers, including PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C73 and the PIC16C74. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PRO MATE™ or PICSTART™ programmer and easily debug and test the sample code. The PICDEM-2 is also usable with the PICMASTER™ emulator, and all of the sample programs can be run and modified using the PICMASTER. Additionally, a generous prototype area is available for user hardware.

FEATURES:

Hardware:

- 40- and 28-pin DIP sockets
- On board +5V regulator for direct input from 9V AC/DC wall adapter or 9V battery.
- RS-232C socket and associated hardware for direct connection to RS-232C interface.
- 5K pot for analog inputs for the PIC16C73/74
- Three push button keys for external stimulus and RESET.
- Eight bright LEDs connected to PORTB for displaying 8-bit binary values.
- Socket for "canned" crystal oscillator.
- Unpopulated holes provided for crystal connection
- 128 x 8 Serial EEPROM.
- LCD module header.
- Keyboard header.
- Unpopulated holes for ACCESS.bus™ connector.



PICDEM-2

Hardware (continued):

- Jacks for connection of 9V battery.
- Jumper to disconnect on-board RC oscillator.
- Prototype area for user hardware.

Software:

- Program for PIC16C74 to demonstrate on-chip A/D feature.
- Program for PIC16C64 to demonstrate I²C™ Serial EEPROM usage.
- All demo programs supplied on 3.5" disk.
- Additional programs available on Microchip's BBS.

DOCUMENTATION:

- A comprehensive User's Guide with easy to follow, step-by-step Getting Started and Tutorial.
- Full schematics.

Samples:

Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:

- PIC16C64
- PIC16C74

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER

DM163002

DESCRIPTION

Low-cost Demonstration Board for
PIC16C64, PIC16C65, PIC16C73 AND PIC17C74



*fuzzy*TECH[®]-MP

Fuzzy Logic Development System for PIC16/17

*fuzzy*TECH-MP FOR MICROCHIP PIC16/17

This product brief describes the technical aspects of the *fuzzy*TECH-MP Fuzzy Logic Development System for PIC16/17 microcontrollers developed by INFORM Software Corporation specifically for Microchip.

The *fuzzy*TECH-MP Development System comes in two versions. The first, the Explorer, contains everything you need to gain a comprehensive working knowledge about fuzzy-logic system design. It is easy-to-use, all graphic editors and tools guide you step-by-step through the development phases of fuzzy systems. The Explorer supports two input variables and one output variable.

The full-featured *fuzzy*TECH-MP Edition offers all of the capabilities of the Explorer, plus it has the additional flexibility of eight input variables and four output variables for designing more complex systems. The full

features are enabled with a hardware key lock attached to the parallel port of the PC.

Included in both versions is *fuzzy*LAB[™], a fully functional demonstration board, to give customers hand-on experience with fuzzy logic systems implementation. *fuzzy*LAB is a simple heating thermostat consisting of a PWM-controlled resistor configured to heat a thermistor to a preset temperature. Using the two fuzzy algorithms provided, a designer can set a target temperature and observe the thermostat response to the set point.

Both systems generate assembly code compatible with the MPASM, Microchip's Universal Assembler, that can be integrated into your application. Examining this code provides you with further insights into the fabrics of fuzzy logic systems.



fuzzyTECH-MP System Requirements

fuzzyTECH-MP will run on any IBM PC[®] (386 or higher) or compatible computer, running DOS 4.1 or later, and Microsoft[®] Windows[®] 3.0 or later. Because *fuzzyTECH-MP* makes extensive use of graphics, a color graphic monitor (VGA) is required, and higher resolutions of 800 x 600 or 1024 x 768 are recommended.

What is Fuzzy Logic?

Fuzzy logic is a technology that enhances mode-based system designs using both intuition and engineering heuristics. Fuzzy logic uses elements of everyday language to represent desired system behavior, thus circumventing the need for rigorous mathematical modeling.

It is an efficient way of designing, optimizing and maintaining highly complex systems transparently.

Fuzzy Logic Applications

Fuzzy logic finds its home in unique applications:

- When no adequate mathematical model for a given problem is readily apparent.
- When non-linearities, time constraints or multiple parameters exist.
- When engineering know-how about the given problem is available or can be acquired during the design process.

The *fuzzyTECH-MP* Implementation

fuzzyTECH-MP provides the following standard features:

- Windows Compatible with full graphical user interface
- 8-Input variables (2 for the Explorer version)
- 4-Output variables (1 for the Explorer version)
- 8-Bit resolution on input and output variables
- 16-Bit computation resolution for the PIC16CXX and PIC17CXX microcontrollers
- No theoretical limit on rules, antecedents and linguistic conjunctions (chip limitations will place a practical limit on these)
- MAX-MIN and MAX-DOT inference methods
- CoM and MoM defuzzification methods
- MPASM Compatible
- PICMASTER[™] Compatible

fuzzyTECH-MP

fuzzyTECH-MP is available directly from Microchip Technology and its authorized distributors. Contact your local sales office for more information.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER

DV005001

DV005002

DESCRIPTION

fuzzyTECH-MP EXPLORER

fuzzyTECH-MP EDITION



MPASM Universal Assembler

Universal PIC16/17 Microcontroller Assembler Software

This product brief describes the technical aspects of the PIC16/17 Assembler. The MPASM Cross Assembler is a PC hosted symbolic assembler. It supports all microcontroller series, including the PIC16C5X, PIC16CXX and PIC17CXX families.

MPASM offers fully featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPASM allows full symbolic debugging from the Microchip Universal Emulator System (PICMASTER™).

MPASM REQUIREMENTS

MPASM will run on any IBM PC/AT® or compatible computer running DOS 5.0 or later.

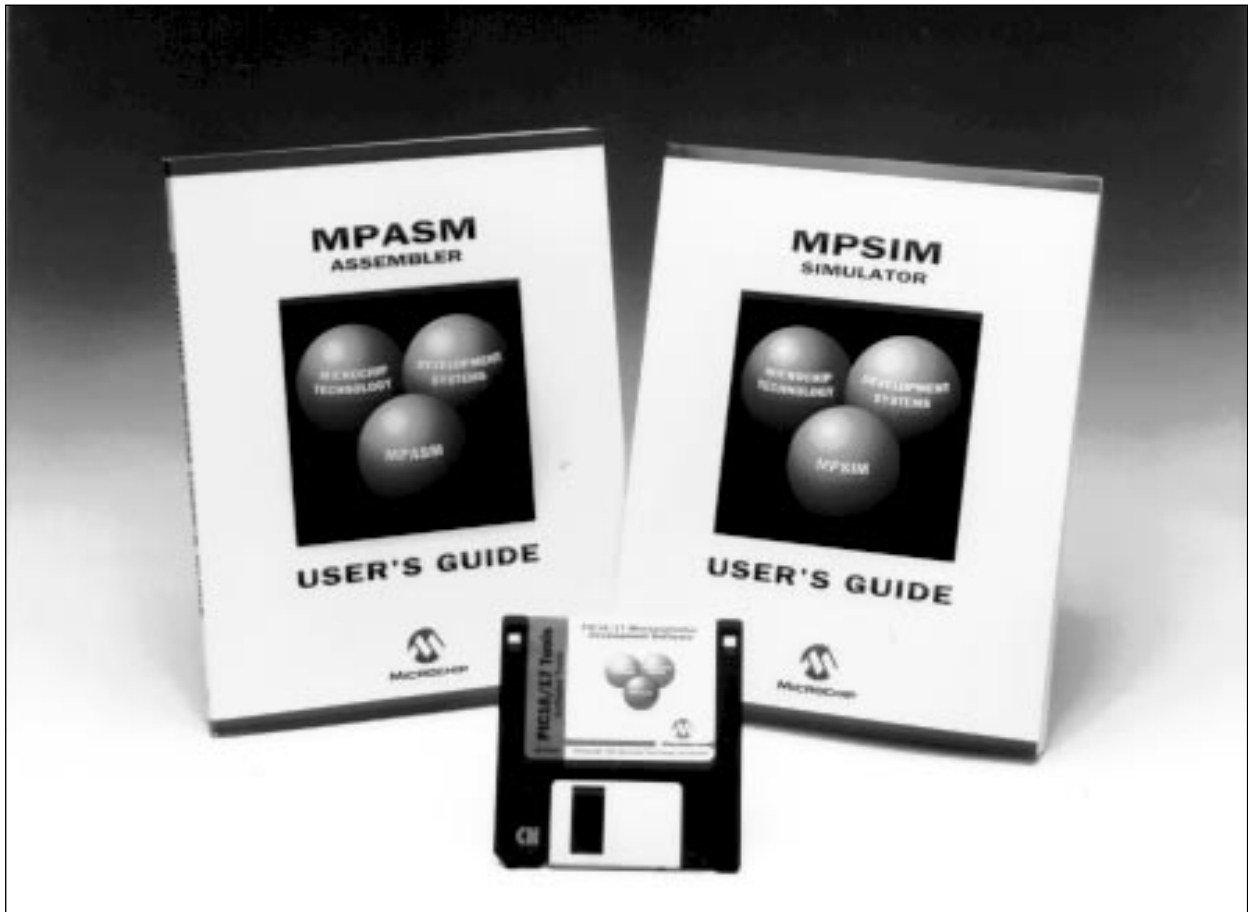
MPASM ASSEMBLER FEATURES

MPASM supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and the 16-bit PIC17CXX cores.

All instructions are single-word and single-cycle, except for branches, which execute in two cycles. Most instructions operate on one or more operands.

MPASM have the following features to assist in developing software for specific user applications:

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro Assembly Capability
- Provides Object, Listing, Symbol and special files required for debugging with one of the Microchip Emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.
- Output formats: INHX8S, INHX8M, INHX32 and relocatable objects.



MPASM Universal Assembler

MPASM DIRECTIVE LANGUAGE

MPASM provides a full featured directive language represented by four basic classes of directives:

- Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, by meaningful names.
- Listing Directives control the MPASM listing display. They allow the specification of titles and subtitles, page ejects and other listing control.
- Control Directives permit sections of conditionally assembled code.
- Macro Directives control the execution and data allocation within macro body definitions.

MPASM INSTRUCTION SET

MPASM supports the entire instruction set of the PIC16C5X, PIC16CXX and PIC17CXX microcontrollers, as represented in the following four classes of instructions:

- Data Move Operations
- Arithmetic and Logical Operations
- Bit Manipulation Operations
- Special Control Operations

The Microchip microcontroller set is used to operate on data located in any of the file registers, including the I/O registers. There are:

- Data Transfer Operations
- Logical Operations
- Rotate Operations

MPASM provides bit level file register operations to manipulate and test individual bits in any addressable register, literal and control operations permitting operations on literals and branches to subroutines in program memory.

The Microchip microcontroller instruction sets allow read and write of special function registers such as the PC and status registers.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
SW165002	MPSIM/MPASM Simulator and Assembler Software and Documentation



MPSIM Simulator

PIC16/17 Microcontroller Simulator

MPSIM is a discrete event simulator software application designed to imitate operation of the PIC16/17 microcontrollers. It allows the user to debug software that will use any of these microcontrollers.

At any instruction boundary, you may examine and/or modify any data area within the processor, or provide external stimulus to any of the pins. MPSIM gives you a solid, low cost, source-level debug tool to help you through the early design verification stages of your project.

MPSIM REQUIREMENTS

MPSIM requires an IBM PC/AT[®] or compatible computer running DOS version 5.0 or later. The PC needs a 3-1/2" floppy disk drive and at least 256K of main memory; MPSIM.EXE occupies roughly 150K. Recommended is a hard disk drive with 5 Mb of available storage.

MPSIM SIMULATOR

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging Microchip microcontroller assembler code.

The simulator is meant for use with smaller projects not requiring precise, more extensive development equipment. Many applications can be developed by using a simulator program alone.

The PIC16CXX and PIC17CXX families support various peripherals and interrupts. MPSIM generally simulates interrupts and most peripheral functions. However, certain peripheral functions are not supported (such as A/D conversion or serial I/O).

The MPSIM Simulator has the following features to assist in the debugging of software / firmware for the user:



MPSIM Simulator

Program Load / Save

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display and Alter

Provisions are made to display and alter Program Memory, Register Files and status register bits. Also, simulator information such as cycle times, elapsed time, and step count can be displayed.

Disassembler

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Utility Functions

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurrences of instructions, register use and ASCII data.

Symbolic Debugging

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace

During program execution, a number of items can be traced. Address ranges, registers and register contents and others.

Breakpoints

The user may specify up to 512 breakpoints at any one time.

Assembler Support

MPSIM works with Microchip's MPASM Universal Assembler.

SALES AND SUPPORT

To order or to obtain information (e.g., on the pricing or delivery), please use the listed part numbers, and refer to the listed sales offices.

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
SW165002	MPSIM/MPASM Simulator and Assembler Software and Documentation

C Compiler

MP-C C COMPILER FOR PIC16/17

This product brief describes the technical aspects of the **MP-C** Code Development System for PIC16/17 micro-controllers developed by Byte Craft Limited.

The **MP-C** Code Development System is a complete 'C' compiler and integrated development environment for Microchip's PIC16/17 family of microcontrollers. The compiler provides powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compiler provides symbol information that is compatible with the PICMASTER™ Universal Emulator memory display (emulator software versions 1.13 and later).

MP-C is fast and efficient. You can quickly produce stand-alone single-chip microcontroller applications. These, taken with its other advantages make the Byte Craft **MP-C** Code Development System the first choice in intelligent compiler technology.

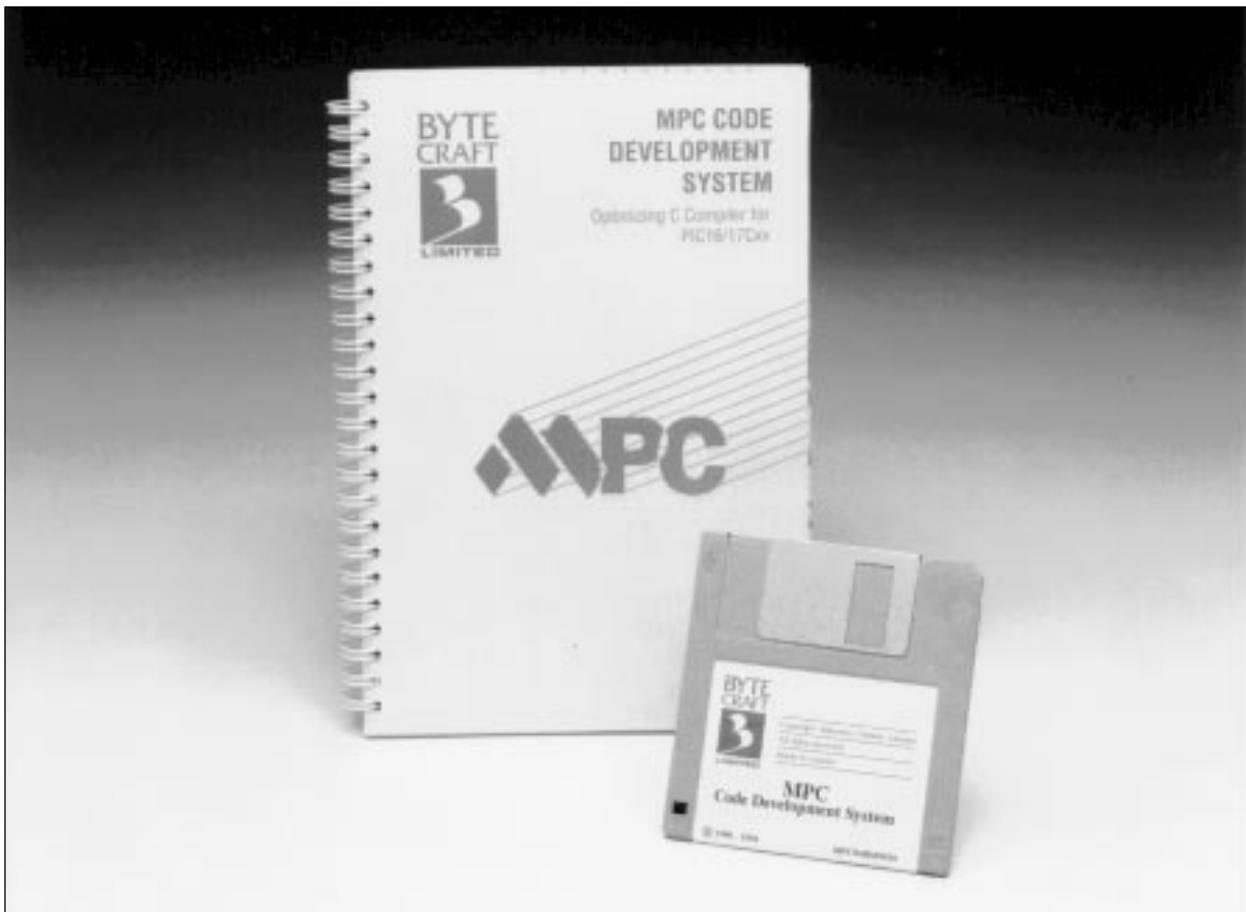
MP-C Requirements

The compiler will run on any IBM PC, PC/XT®, PC/AT® or compatible computer, running DOS 5.0 or later.

MP-C Code Development System Features

MP-C supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and 16-bit PIC17CXX cores. It is a rule-based compiler with expert systems tailored to each of these platforms for optimal efficiency.

The compiler generates executable code directly from the compile process. There is no need for an extra step to assemble code generated by the compiler.



MP-C

MP-C has the following features to assist in developing PIC16/17 software for specific user applications:

- Provides Object, Listing, Symbol and special files required for debugging with other Microchip Development systems.
- Supports interrupt routines
- Checks source against target hardware definitions
- Generates efficient, tight object code
- Includes a linker and built-in macro assembler
- 'C' enhancements specific to the PIC16/17 families' instruction sets.
- Output formats: INHX8S, INHX8M, and INHX32.

MP-C Microprocessor Specific Extensions

The **MP-C** Code Development System includes common 'C' enhancements such as ROM arrays, binary constants and case statements together with functions specific to the PIC16/17 architecture.

- **Binary Constants** of the form 0b0101110 which are logical extensions to the conventional 0x1a3b style of hexadecimal constants. You may also use 0B as leading characters.
- **Case Statements** are supported well by the PIC16/17 instruction set and the compiler provides a superset of the standard 'C' case statement. For example, case 4,5:, case '0':9', and complex case statements are allowed.
- **Processor Specific Functions** that are specific to the PIC16/17 family. For example NOP() and SLEEP() produce the equivalent PIC16/17 instruction.
- **"At" or @ Extension** allows you to fix a variable to a specific address in memory, for example: int N @ 0x0C.

SALES AND SUPPORT

The MP-C system is supported by Microchip via the Microchip technical support hotline.

The **MP-C** Code Development System is supplied directly by Byte Craft Limited of Waterloo, Ontario, Canada.

If you have any questions please contact your regional Microchip FAE or Microchip technical support personnel at (602) 786-7627.



MICROCHIP

TOTAL ENDURANCE™

Microchip Serial EEPROM Endurance Model



FEATURES

- IBM® PC compatibility
- Windows® 3.1 or DOS 3.1 compatibility
- Automatic or manual recalculation
- Real-time update of data
- Full-screen or windowed graphical view
- Hypertext on-screen help
- Key or slide-bar entry of parameters
- On-screen editing of parameters
- Single-click copy of plot to clipboard
- Numeric export to delimited text file
- On-disk Endurance Tutorial

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows 3.1
- 1MB memory
- 386 or 486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/24AAXX/85CXX
- Microchip 3-wire 93CXX/93LCXX/93AAXX Series
- Microchip 4-wire 59C11

Total Endurance™

DESCRIPTION

Microchip's revolutionary Total Endurance Model provides electronic systems designers with unprecedented visibility into Serial EEPROM-based applications. This advanced software model (with a very friendly user interface) eliminates time and guesswork from Serial EEPROM-based designs by accurately predicting the device's performance and reliability within a user-defined application environment. Design trade-off analysis which formerly consumed days or weeks can now be performed in minutes...with a level of accuracy that delivers a truly robust design.

Users may input the following application parameters:

- Serial EEPROM device type
- Bytes to be written per cycle
- Cycling mode - byte or page
- Data pattern type - random or worst-case
- Temperature in °C
- Erase/Write cycles per day
- Application lifetime or target PPM level

The model will respond with FIT rate, PPM level, application life and a plot of the PPM level vs. number of cycles. The model is available in both DOS and Windows versions.

BACKGROUND

Microchip's research into the Erase/Write endurance of Serial EEPROMs has resulted in the conclusion that endurance depends upon three primary effects: the physical properties of the EEPROM cell, the internal error-correction technology employed, and the application environment. EEPROM endurance specified as a "typical" value in device data sheets must therefore be evaluated on a case-by-case basis, taking into account the manner in which the device will be used in the application. The Microchip Total Endurance™ software applies the user-defined application parameters to a complex mathematical model in order to emulate the EEPROM's performance and reliability in the system.

USING THE MODEL

The user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

ACCURACY OF THE MODEL

The accuracy of the Microchip Total Endurance model has been verified against test data to within ten percent of the actual values. However, Microchip makes no warranty as to its accuracy or applicability of the information to any given application. It is intended to be used as a guide to aid designers of Serial EEPROM-based systems in performing trade-off analysis and developing robust and reliable designs.

Order Information:

<u>Description</u>	<u>Part Number</u>
Total Endurance Software Disk	SW242001



TRUEGAUGE™

TrueGauge Intelligent Battery Management Development Tool

INTRODUCTION

The MTA11200 TrueGauge Intelligent Battery Management IC is supported by a user friendly tool for system development. The DV114001 operates under Microsoft Windows®. This development tool enables for management of all phases of product development, including inception, debugging and maintenance. System design verification can be accomplished before a hardware prototype needs to be built, thus reducing time and cost. The user interface provides a graphically-oriented development environment. The data logging feature saves measured data into a file that can be imported to Excel®. Battery parameters can be changed, down-loaded to the TrueGauge, then battery performance can be evaluated.

SUMMARY OF FEATURES

The TrueGauge development tool is a tool for system development under Windows. The development tool kit contains the following:

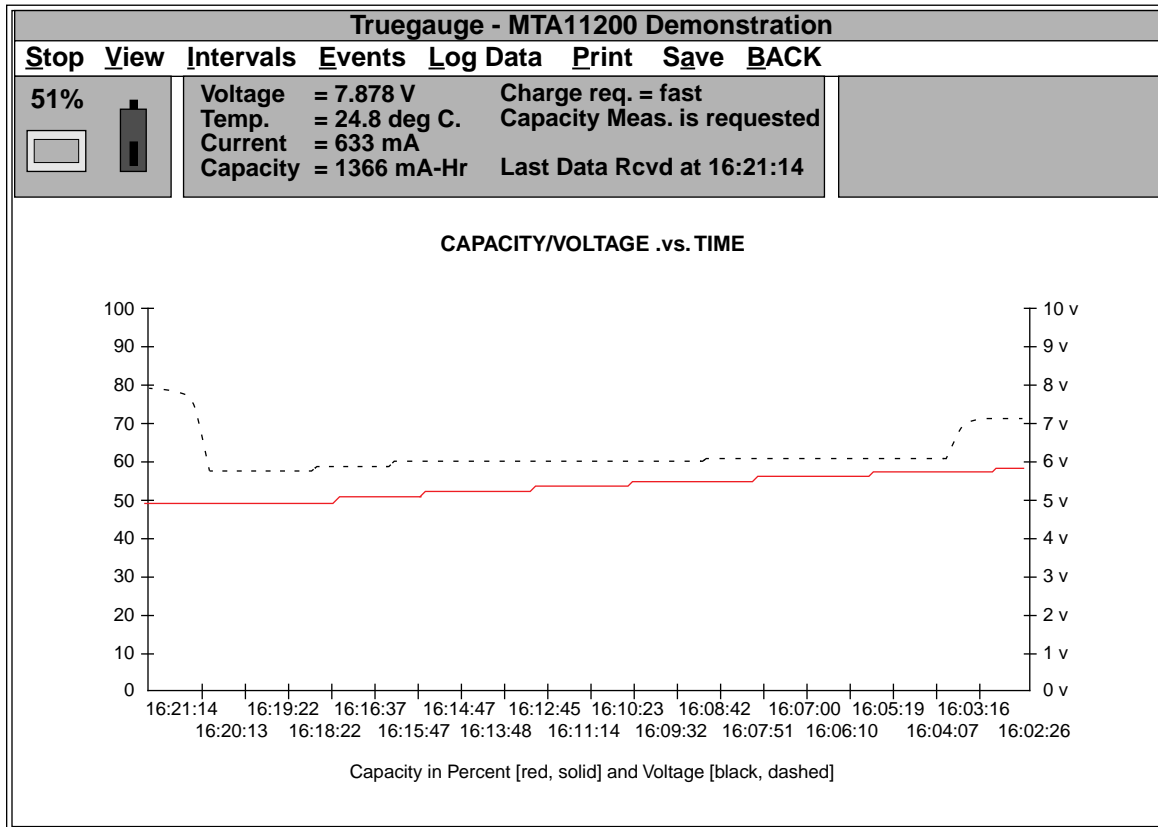
- NiCd battery with TrueGauge module
- NiMH battery with TrueGauge module
- Stand-alone TrueGauge module
- Charger/discharger Interface Board
- Universal Power Supply with power cord
- PC Interface Cable with DB9-DB25 converter
- Design/Verification software on a 3.5" diskette, including calibration routines
- MTA11200B and 24LC01B product samples
- MTA11200B Datasheet
- TrueGauge Development Tool User's Guide
- Version 2.5 supports MTA11200B

FIGURE 1: TRUEGAUGE DEVELOPMENT TOOL KIT



Battery status information is plotted on the computer allowing for real-time monitoring of battery management parameters (Figure 2).

FIGURE 2: CAPACITY/VOLTAGE OVER TIME



Parameters can be changed easily and downloaded to the TrueGauge module (Figure 3 and Figure 4).

FIGURE 3: CONFIGURATION CONTROL PANEL

MTA11200 Configuration Data

Rev:

Fast Charge Termination

Delta Temp Delta Time

0.50 degrees C / min

Nominal Battery Capacity

1800 milliamp-hours

Number of Cells: 6

Measured Battery Capacity

1800 milliamp-hours

Battery Warning Levels (% of Full Charge)

Low

Critical

Shutdown

Fast Charge Termination (Maximum Limits)

Volt (volts)

Time (min)

Charge Termination Temperatures (deg. C)

Max Fast

Min Fast

Max Maint

Min Maint

Charge Turn On Threshold (% of Charge Capacity)

% Charge

End of Discharge Voltage

Volts

FIGURE 4: ADVANCED CONFIGURATION DISPLAY

Advanced Configuration

REPINTRVL & REPMODE: Broadcast Interval

Interval ← →

LED Mode: BCD 10% Levels Discrete 20% Levels

Broadcast Data Select: Single Byte Sixteen Byte

Counts

TOERRS: Charge Time ← →

LOTERRS: Low Temp. ← →

HITERRS: High Temp. ← →

HIVERRS: High Volt. ← →

CCCR: Charge Cycle ← →

TCC: Charge Cycle ← →

USER: User Storage

User Value # 1 ← →

User Value # 2 ← →

User Value # 3 ← →

User Value # 4 ← →

FIGURE 5: REV. B CONTROL OPTIONS (BOPT)

MTA11200 Rev. B Control Options

Rev B Control Options (BOPT)

Fast Charge Termination: Does not affect % capacity gauge Forces % Capacity gauge to 100%

Primary Fast Charge Termination Method: Stop fast charge when gauge at 100% or dT/dt detect Stop fast charge on dT/dt detect

Negative Delta Voltage (-dV) control panel: Use current monitor to prevent false -dV detects Use voltage monitor to prevent false -dV detects

Wake up from Standby if current flow is detected: Enable wake up on current flow detect Disable wake up on current flow detect

EODV# pin function: EODV pin indicates >=0% charge, P0#, (like P20#, P40#, P60#, P80# pins) EODV pin indicates battery voltage below EODV limit

Wake up current threshold: IWAKE: (mA) ← →

Current monitor -dV detector threshold: DIMON: (mA) ← →

Standby current deadzone threshold: STBYIDZ: (mA) ← →

FIGURE 6: CHARGE EFFICIENCY vs STATE OF CHARGE COMPENSATION

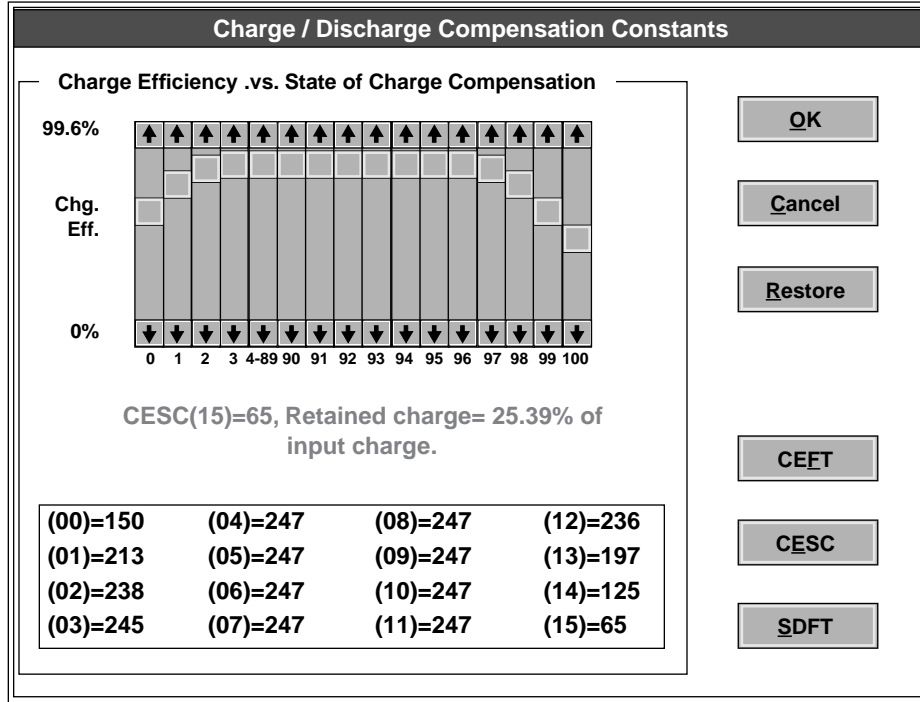
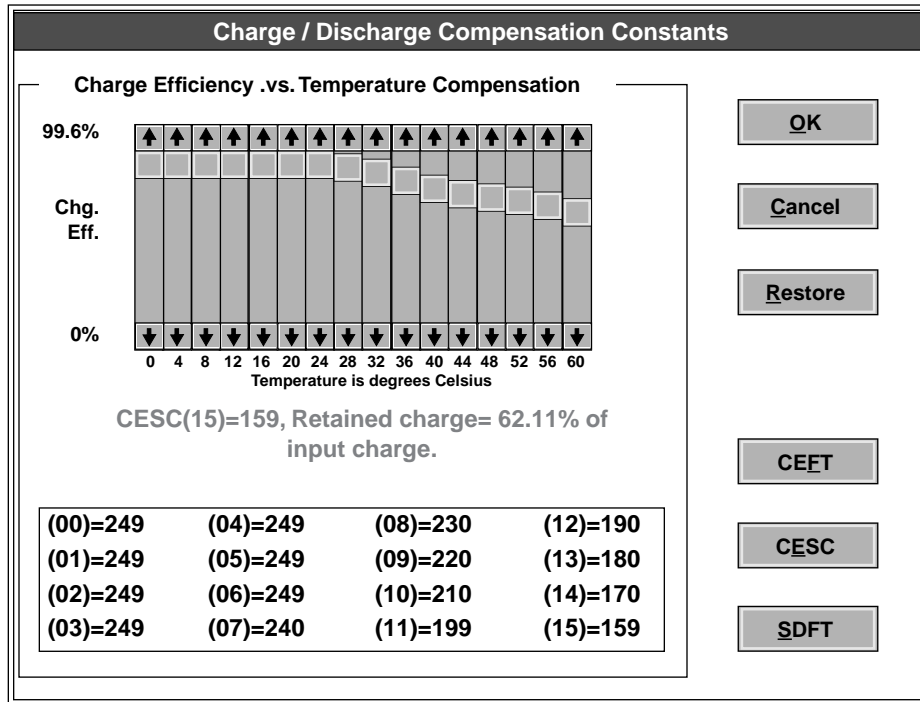


FIGURE 7: CHARGE EFFICIENCY vs TEMPERATURE COMPENSATION



System design verification can be accomplished before hardware implementation (Figure 8, Figure 9, and Figure 10).

FIGURE 8: TRUEGAUGE VOLTAGE AND CAPACITY vs. TIME

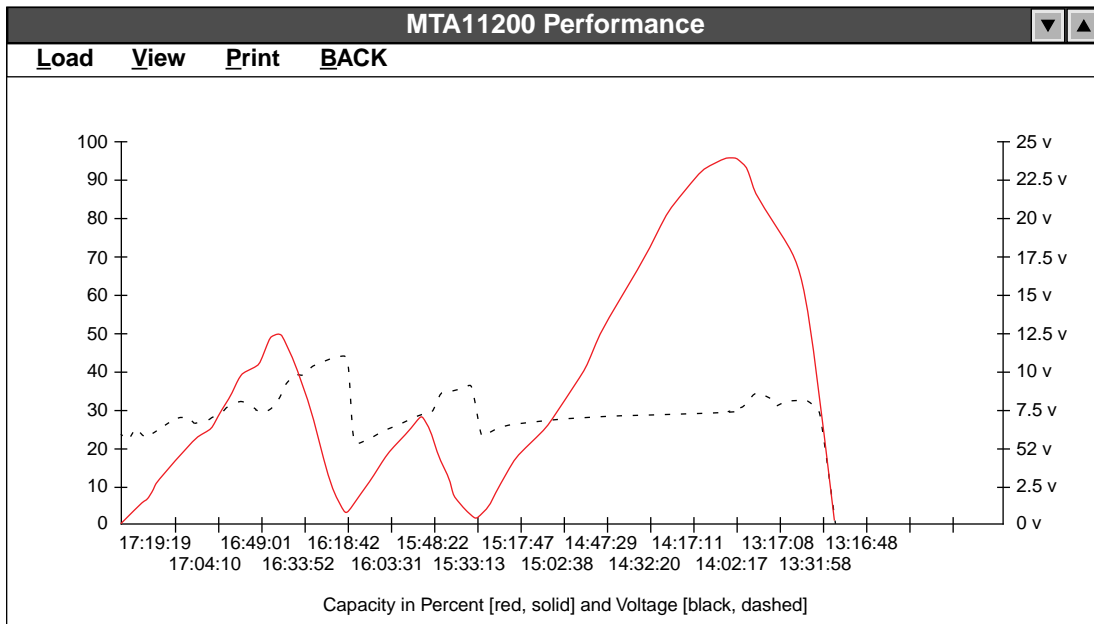


FIGURE 9: TEMPERATURE vs. TIME

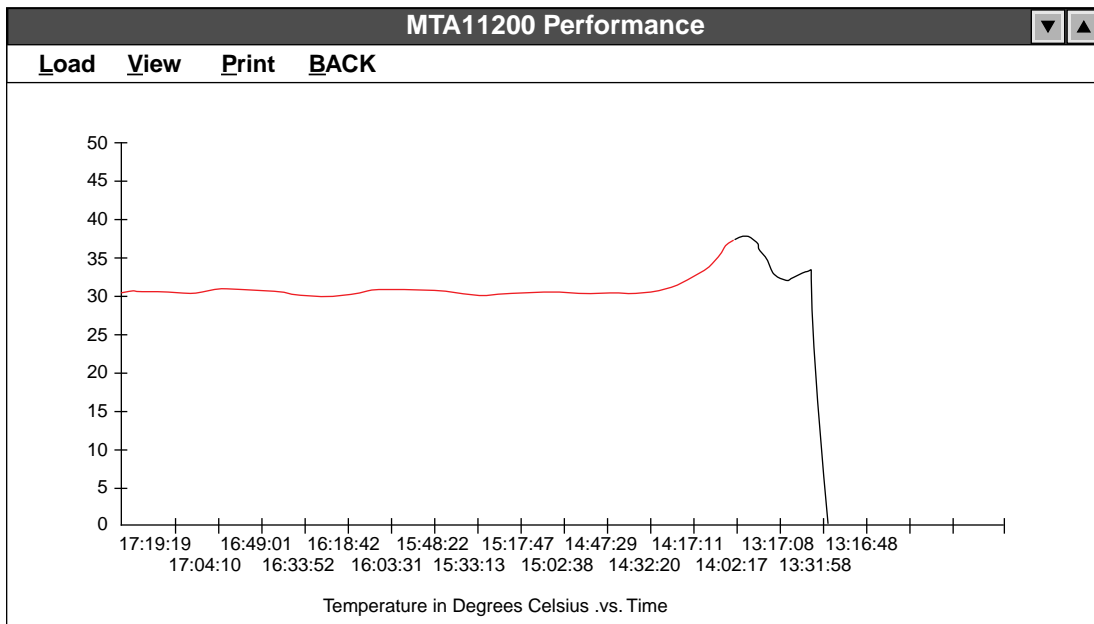
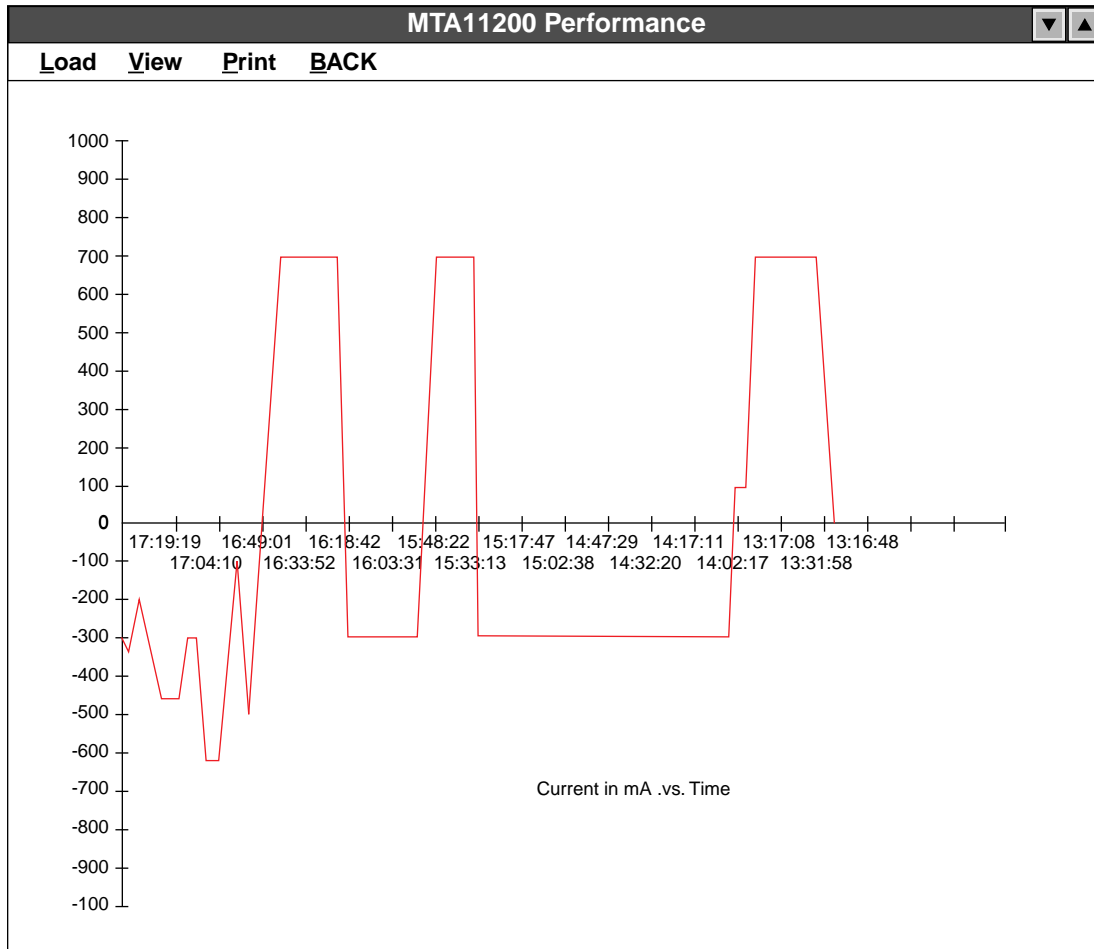


FIGURE 10: CURRENT vs. TIME



A data logging feature saves measured data into a file that can be imported into Microsoft Excel (Figure 11)

FIGURE 11: EXAMPLE OF DATA LOG FILE

Time	Remaining Capacity (%)	Voltage	Temperature (C)	Current (mA)	Total Capacity (mA-Hr)	Flag_byte	Error_byte
22:39:13	9	8.645	23.88281	647	1200	BB	0
22:39:15	9	8.645	23.88672	647	1200	BB	0
22:39:23	9	8.646	23.90234	648	1200	BB	0
22:39:34	9	8.646	23.92188	647	1200	BB	0
22:39:44	9	8.647	23.9375	647	1200	BB	0
22:39:55	9	8.647	23.95313	647	1200	BB	0
22:40:04	10	8.647	23.96484	647	1200	BB	0
22:40:14	10	8.648	23.96875	647	1200	BB	0
22:40:25	10	8.648	23.98047	647	1200	BB	0

SALES AND SUPPORT

To order or to obtain information (e.g., on pricing or delivery), please use the listed part numbers, and refer to the listed sales office.

PART NUMBER

DV114001

AC113001

DESCRIPTION

TRUEGAUGE DEVELOPMENT TOOL

TRUEGAUGE UPGRADE KIT
(3 MODULES and SOFTWARE DISK)

SECTION 7

SALES AND SERVICE LOCATIONS

Factory Representatives

Africa	7-1
Asia/Pacific	7-1
Canada	7-1
Europe	7-2
Mexico	7-2
South America	7-3
United States	7-4

Distributors/Service

Africa	7-7
Asia/Pacific	7-7
Canada	7-8
Europe	7-10
Mexico	7-12
South America	7-12
United States	7-13

Factory Sales	7-19
----------------------------	------



Factory Representatives

AFRICA

Tempe Technologies Pty Ltd
P.O. Box 2480
Edenvale 1610
South Africa
Tel: 27/11-3341427
Fax: 27/11-3340103

CANADA

Alberta

Enerlec Sales Ltd.
#103 155 Glendeer Circle S.E.
Calgary, Alberta T2H 2S8 Canada
Tel: 403-777-1550
Fax: 403-777-1553

British Columbia

Enerlec Sales Ltd.
#7 3671 Viking Way
Richmond, B.C. V6V 1W1 Canada
Tel: 604-273-0882
Fax: 604-273-0884

Manitoba

Enerlec Sales Ltd.
#7 3671 Viking Way
Richmond, B.C. V6V 1W1 Canada
Tel: 604-273-0882
Fax: 604-273-0884

Montreal

Electronic Sales Professionals
10690 Pelouin Street
Suite 210
Montreal, Quebec, Canada H2C 2K3
Tel: 514-388-6596
Fax: 514-388-8402

Ottawa

Electronic Sales Professionals
215 Stafford Road
Unit 104
Nepean, Ontario, Canada K2H 9C1
Tel: 613-828-6881
Fax: 613-828-5725

Saskatchewan

Enerlec Sales Ltd.
#7 3671 Viking Way
Richmond, B.C., Canada V6V 1W1
Tel: 604-273-0882
Fax: 604-273-0884

Toronto

Electronic Sales Professionals
137 Main Street
Suite 204
Markham, Ontario, Canada L3P 1Y2
Tel: 905-294-3520
Fax: 905-294-3806

Factory Representatives

EUROPE

Finland

Memec Finland
Asemankello
Vernissankatu 6
Fin 01300 Vantaa
Finland
Tel: 358-07001-9830
Fax: 358-07001-9839

Germany

Active Rep GmbH
Beningastr. 24
D-26721 Emden
Germany
Tel: 49/4921-979071
Fax: 49/4921-979072

Active Rep GmbH
Blumental 1-3
D-42653 Solingen
Germany
Tel: 49/212-25823-0
Fax: 49/212--282323

Active Rep GmbH
Kennedy-StraBe 5
D-75438 Knittlingen
Germany
Tel: 49/0743-940011
Fax: 49/0704-333492

Active Rep GmbH
Reichenastr. 18
D-81234 Muenchen
Germany
Tel: 49/89-89689181
Fax: 49/89-89689183

Ireland

Eltech Agencies, Ltd.
27 Maccurtain Street
Cork
Ireland
Tel: 353-21-509366
Fax: 353-21-509344

Israel

Elina Electronics Ltd.
14 Raoul Wallenberg St.
P.O. Box 13190
Tel Aviv 61131
Israel
Tel: 972 3 49 85 43
Fax: 972 3 49 87 45

Netherlands

Sonetech
P.O. Box 258
5670 AG Nuenen
Netherlands
Tel: 31-40-837075
Fax: 31-40-832300

Norway

Component-74 Eidsvold AS
Postboks 9
N-2070 Raholt
Tel: 47-63-951174
Fax: 47-63-953066

Scotland

Juniper Solutions
32 Enterprise House
Springkerse Business Park
Sterling FK7 7UF
Scotland
Tel: 44-1786-446220/1
Fax: 44-1786-446223

Sweden

Memec Scandinavia AB
Kvarnholmsvagen 52
131 31 Nacka
Sweden
Tel: 46-8643-4190
Fax: 46-8643-1195

Turkey

Inter Muehendislik Danismanlik
Ve Ticaret A.S. 1
Hasircbasi Caddesi No. 55
81310 Kadikoy
Istanbul
Turkey
Tel: 90 216 349 94 00
Fax: 90 216 349 94 31

MEXICO

CompTech de Mexico
Av. Morelos Sur 809
MB5 Suite 200
Las Palmas
Cuernavaca, MOR 62050
Tel: 52 73 122733
Fax: 52 73 185500

Factory Representatives

SOUTH AMERICA

Argentina

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Bolivia

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Brazil

Aplicacoes Eletronicas
Artimar Ltda.
Rua Marques de Itu N°. 70-10And.
CEP 01223
Sao Paulo, Brazil
Tel: 55-11-231-0277
Fax: 55-11-255-0511

Caribbean

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Chile

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Columbia

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Ecuador

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Paraguay

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Peru

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Uruguay

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Venezuela

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Factory Representatives

UNITED STATES

Alaska

Trinity Technologies
1261 Oakmead Parkway
Sunnyvale, CA 94086
Tel: 408-733-9000
Fax: 408-733-9970

Alabama

Concord Component Reps Inc.
190 Lime Quarry Road, Suite 102
Madison, AL 35758
Tel: 205-772-8883
Fax: 205-772-8262

Arizona

Western High Tech Marketing, Inc.
9414 E. San Salvador, Suite 206
Scottsdale, AZ 85258
Tel: 602-860-2702
Fax: 602-860-2712

Arkansas

CompTech Sales, Inc.
2401 Gateway Drive
Suite 114
Irving, TX 75063
Tel: 214-751-1181
Fax: 214-550-8113

California

Costa Mesa

Competitive Technology, Inc.
200 Baker Street, Suite 101
Costa Mesa, CA 92626
Tel: 714-540-5501
Fax: 714-540-5171

Escondido

Eagle Technical Sales Assoc. Inc.
1900 Sunset Drive, Suite A
Escondido, CA 92025
Tel: 619-743-6550
Tel: 619-743-6585

Sunnyvale

Trinity Technologies
1261 Oakmead Parkway
Sunnyvale, CA 94086
Tel: 408-733-9000
Fax: 408-733-9970

Colorado

Western Region Marketing
9176 Marshall Place
Westminster, CO 80030
Tel: 303-428-8088
Fax: 303-426-8585

Connecticut

Gilford

S-J Associates, Inc.
10 Copper Ridge Circle
Gilford, CT 06437
Tel: 203-458-7558
Fax: 203-458-1181

Naugatuck

S-J Associates, Inc.
15 Coventry Lane
Naugatuck, CT 06770
Tel: 203-723-4707
Fax: 202-723-1629

Delaware

S-J Mid-Atlantic, Inc.
131-D Gaither Drive
Mt. Laurel, NJ 08054
Tel: 609-866-1234
Fax: 609-866-8627

District of Columbia

S-J Chesapeake
900 S. Washington Street
Suite 307
Falls Church, VA 22046
Tel: 703-533-2233
Fax: 703-533-2236

Florida

Altamonte Springs

Electramark Florida, Inc.
401 Whooping Loop, Suite 1565
Altamonte Springs, FL 32701
Tel: 407-830-0844
Fax: 407-830-0847

Boca Raton

Electramark Florida, Inc.
621 NW 53rd Street, Suite 240
Boca Raton, FL 33487
Tel: 407-998-8820
Fax: 407-998-8821

Tampa

Electramark Florida, Inc.
2910 W. Waters Ave.
Tampa, FL 33614
Tel: 813-915-1177
Fax: 813-915-1188

Georgia

Concord Component Reps Inc.
6825 Jimmy Carter Boulevard
Norcross, GA 30071
Tel: 404-416-9597
Fax: 404-441-0790

Hawaii

Trinity Technologies
1261 Oakmead Parkway
Sunnyvale, CA 94086
Tel: 408-733-9000
Fax: 408-733-9970

Idaho

Micro Sales
2122 - 112th Ave. N.E.
Bellevue, WA 98004
Tel: 206-451-0568
Fax: 206-453-0092

Illinois

Janus Incorporated
650 E. Devon Avenue
Itasca, IL 60143
Tel: 708-250-9650
Fax: 708-250-8761

Indiana

Fort Wayne

Electro Reps, Inc.
125 Airport North Office Park
Fort Wayne, IN 46825
Tel: 219-489-8205
Fax: 219-489-8408

Indianapolis

Electro Reps, Inc.
7240 Shadeland Station, Suite 275
Indianapolis, IN 46256
Tel: 317-842-7202
Fax: 317-841-0230

Iowa

Spectrum Sales
1364 Elmhurst Drive NE
Cedar Rapids, IA 52402
Tel: 319-366-0576
Fax: 319-366-0635

Kansas

Spectrum Sales
5382 W. 95th Street
Prairie Village, KS 66207
Tel: 913-648-6811
Fax: 913-648-6823

Factory Representatives

Kentucky

Northern

Technology Marketing Corporation
7775 Cooper Road
Suite 3
Cincinnati, OH 45242
Tel: 513-984-6720
Fax: 513-984-6874

Southern

Electro Reps, Inc.
7240 Shadeland Station, Suite 275
Indianapolis, IN 46256
Tel: 317-842-7202
Fax: 317-841-0230

Louisiana

CompTech Sales, Inc.
10550 Richmond Avenue
Suite 105
Houston, TX 77042
Tel: 713-781-7420
Fax: 713-781-5865

Maine

S-J New England, Inc.
40 Mall Road
Burlington, MA 01803
Tel: 617-272-5552
Fax: 617-272-5515

Maryland

S-J Chesapeake
900 S. Washington Street
Suite 307
Falls Church, VA 22046
Tel: 703-533-2233
Fax: 703-533-2236

Massachusetts

S-J New England, Inc.
40 Mall Road
Burlington, MA 01803
Tel: 617-272-5552
Fax: 617-272-5515

Michigan

Farmington Hills

J. L. Montgomery Associates, Inc.
34405 West 12 Mile Road, Suite 149
P.O. Box 2726
Farmington Hills, MI 48333-2726
Tel: 810-489-0099
Fax: 810-489-0189

Grand Rapids

J. L. Montgomery Associates, Inc.
2215 Oak Industrial Drive NE
Grand Rapids, MI 49505
Tel: 616-458-5490
Fax: 616-458-5709

Minnesota

Comprehensive Technical Sales Inc.
6513 City West Parkway
Eden Prairie, MN 55344
Tel: 612-941-7181
Fax: 612-941-4322

Mississippi

Concord Component Reps Inc.
190 Lime Quarry Road, Suite 102
Madison, AL 35758
Tel: 205-772-8883
Fax: 205-772-8262

Missouri

Spectrum Sales
5494 Brown Road, Suite 124
St. Louis, MO 63042
Tel: 314-921-1313
Fax: 314-921-0701

Montana

Western Region Marketing
9176 Marshall Place
Westminster, CO 80030
Tel: 303-428-8088
Fax: 303-426-8585

Nebraska

Spectrum Sales
5382 W. 95th Street
Prairie Village, KS 66207
Tel: 913-648-6811
Fax: 913-648-6823

Nevada

Northern

Trinity Technologies
1261 Oakmead Parkway
Sunnyvale, CA 94086
Tel: 408-733-9000
Fax: 408-733-9970

Southern

Western High Tech Marketing, Inc.
9414 E. San Salvador, Suite 206
Scottsdale, AZ 85258
Tel: 602-860-2702
Fax: 602-860-2712

New Hampshire

S-J New England, Inc.
40 Mall Road
Burlington, MA 01803
Tel: 617-272-5552
Fax: 617-272-5515

New Jersey

Northern

Parallax
734 Walt Whitman Road
Melville, NY 11747
Tel: 516-351-1000
Fax: 516-351-1606

Southern

S-J Mid-Atlantic, Inc.
131-D Gaither Drive
Mt. Laurel, NJ 08054
Tel: 609-866-1234
Fax: 609-866-8627

New Mexico

Western High Tech Marketing, Inc.
9414 E. San Salvador, Suite 206
Scottsdale, AZ 85258
Tel: 602-860-2702
Fax: 602-860-2712

New York

Melville

Parallax
734 Walt Whitman Road
Melville, NY 11747
Tel: 516-351-1000
Fax: 516-351-1606

Rochester

Apex Associates, Inc.
1210 Jefferson Road
Rochester, NY 14623
Tel: 716-272-7040
Fax: 716-272-7756

North Carolina

Zucker Associates
4070 Barrett Drive
Raleigh, NC 27609
Tel: 919-782-8433
Fax: 919-782-8476

North Dakota

Comprehensive Technical Sales Inc.
6513 City West Parkway
Eden Prairie, MN 55344
Tel: 612-941-7181
Fax: 612-941-4322

Factory Representatives

Ohio

Cincinnati

Technology Marketing Corporation
7775 Cooper Road
Suite 3
Cincinnati, OH 45242
Tel: 513-984-6720
Fax: 513-984-6874

Middleburg Heights

Technology Marketing Corporation
One Independence Place
4807 Rockside Road, Suite 360
Independence, OH 44131
Tel: 216-520-0150
Fax: 216-520-0190

Oklahoma

CompTech Sales, Inc.
18700 Woodbriar Lane
Catoosa, OK 74015
Tel: 918-266-1966
Fax: 918-266-1808

Oregon

Micro Sales
1865 N.W. 169th Pl. #210
Beaverton, OR 97006
Tel: 503-645-2841
Fax: 503-645-3754

Pennsylvania

Eastern

S-J Mid-Atlantic, Inc.
131-D Gaither Drive
Mt. Laurel, NJ 08054
Tel: 609-866-1234
Fax: 609-866-8627

Western

Technology Marketing Corporation
7775 Cooper Road
Suite 3
Cincinnati, OH 45242
Tel: 513-984-6720
Fax: 513-984-6874

Rhode Island

S-J New England, Inc.
40 Mall Road
Burlington, MA 01803
Tel: 617-272-5552
Fax: 617-272-5515

South Carolina

Zucker Associates
4070 Barrett Drive
Raleigh, NC 27609
Tel: 919-782-8433
Fax: 919-782-8476

South Dakota

Comprehensive Technical Sales Inc.
6513 City West Parkway
Eden Prairie, MN 55344
Tel: 612-941-7181
Fax: 612-941-4322

Tennessee

Eastern

Zucker Associates
4070 Barrett Drive
Raleigh, NC 27609
Tel: 919-782-8433
Fax: 919-782-8476

Western

Concord Component Reps Inc.
6825 Jimmy Carter Boulevard
Norcross, GA 30071
Tel: 404-416-9597
Fax: 404-441-0790

Texas

Austin

CompTech Sales, Inc.
11130 Jollyville Road, Suite 200
Austin, TX 78759
Tel: 512-343-0300
Fax: 512-345-2530

Brownsville

CompTech Sales, Inc.
2390 Central Blvd., Suite P
Brownsville, TX 78520
Tel: 210-504-9693
Fax: 210-504-9982

El Paso

CompTech Sales, Inc.
3120 Wheeling Rd.
El Paso, TX 79930
Tel: 915-566-1022
Fax: 915-566-1030

Houston

CompTech Sales, Inc.
10550 Richmond Avenue
Suite 105
Houston, TX 77042
Tel: 713-781-7420
Fax: 713-781-5865

Irving

CompTech Sales, Inc.
2401 Gateway Drive
Suite 114
Irving, TX 75063
Tel: 214-751-1181
Fax: 214-550-8113

Utah

Western Region Marketing
3539 South Main, Suite 210
Salt Lake City, UT 84115
Tel: 801-268-9768
Fax: 801-268-9796

Vermont

S-J New England, Inc.
40 Mall Road
Burlington, MA 01803
Tel: 617-272-5552
Fax: 617-272-5515

Virginia

S-J Chesapeake
900 S. Washington Street
Suite 307
Falls Church, VA 22046
Tel: 703-533-2233
Fax: 703-533-2236

Washington

Micro Sales
2122 - 112th Ave. N.E.
Bellevue, WA 98004
Tel: 206-451-0568
Fax: 206-453-0092

West Virginia

S-J Chesapeake
900 S. Washington Street
Suite 307
Falls Church, VA 22046
Tel: 703-533-2233
Fax: 703-533-2236

Wisconsin

Janus Incorporated
375 Williamstowne
Delafield, WI 53018
Tel: 414-646-5420
Fax: 414-646-2421

Wyoming

Western Region Marketing
9176 Marshall Place
Westminster, CO 80030
Tel: 303-428-8088
Fax: 303-426-8585



MICROCHIP

Distributors

AFRICA

South Africa

Pace Electronic Components Ltd.
Cnr. Vanacht & Gewel Streets
P.O. Box 701
Isando 1600, Transvaal
Republic of South Africa
Tel: 27/11-9741211
Fax: 27/11-9741271

ASIA/PACIFIC

Australia

Har-Tec Australia
205 Middleborough Road
Box Hill, Victoria Australia 3128
Tel: 61-3-890-0970
Fax: 61-3-899-5191

China

Wuhan Liyuan
107-2 Luo Yu Road
Wuhan 430070
Peoples Republic of China
Tel: 86-27-7802986
Fax: 86-27-7802985

Fuzhou Dingxu
3/F, No3 Building
Chang Ting Industrial Area
Xianjin Rd
Fuzhou
Peoples Republic of China
Tel: 86-591-372-9174
Fax: 86-591-371-7704

Goldenchip Electronics
7/F, 275 Wushi Road
Fuzhou
Peoples Republic of China
Tel: 86-591-784-4159
Fax: 86-591-784-4160

Hong Kong

Goldenchip Research
11/F, 1116-7 Kwai on Fty Bldg
103 Tai Lin Pai Road
Kwai Chung, N.T. Hong Kong
Tel: 852-2426-3968
Fax: 852-2481-7403

Infintron
Rm 802, 8/F Kinok Centre
9 Hung To Rd, Kwun Tong
Kowloon
Hong Kong
Tel: 852-2341-6611
Fax: 852-2950-0987

Excelpoint Systems
Rm. 2108, Fortress Tower
250 King's Road
North Point
Hong Kong
Tel: 852-2503-2212
Fax: 852-2503-1558

India

Excelpoint Systems
11/1 & 11/2 Dickenson Road
Bangalore 560042 India
Tel: 91-80-5586719
Fax: 91-80-5586606

Japan

Dianichi Contronics Inc.
Dainichi Bldg. 1-7 Karaku-Chrome,
Bunkyo-Ku
Tokyo 112, Japan
Tel: 81-3-818-8081
Fax: 81-3-3818-8088

Marubeni Hytech Co., Ltd.
Marubeni Hytech Building
4-20-22, Koishikawa
Bunkyo-Ku
Tokyo 112 Japan
Tel: 81-3-3817-4921
Fax: 81-3-3817-4880

Japan (continued)

Nippon Precision Device Corp.
Nichibei Time 24 Bldg.
35 Tansu-Cho, Shinjuku-Ku
Tokyo 162 Japan
Tel: 81-3-3260-1411
Fax: 81-3-3260-7100

Ryoden Trading Co.
3-15-15, Higashi Ikebukuro
Toshima-Ku
Tokyo 170 Japan
Tel: 81-3-5396-6197
Fax: 81-3-5396-6443

Unidux
5-1-21, Kyonan-cho
Musashino-shi
Tokyo 180 Japan
Tel: 81-422-32-4111
Fax: 81-422-31-2050

Korea

ProCHIPS Inc.
779-12, Daelim 3-Dong
Youngdeungpo-Ku,
Seoul, Korea
Tel: 822-849-8567
Fax: 822-849-8659

New Zealand

Har-Tec NZ Limited
50 O'Rourke Road
P.O. Box 12055
Penrose, Auckland New Zealand
Tel: 64-9-525-1096
Fax: 64-9-525-1097

Distributors

ASIA/PACIFIC (continued)

Singapore

Gates Engineering
1123 Serangoon Road
#03-01 UMW Building
Singapore 1232
Tel: 65-299-9937
Fax: 65-299-7636

Excelpoint Systems
12 Tannery Lane #06-01/02
Beam Building
Singapore 1334
Tel: 65-741-8980
Fax: 65-741-8980

Taiwan, R.O.C.

Solomon Technology Corp.
5th Floor, No. 293, Sec. 5
Chung Hsiao E. Rd.
Taipei, Taiwan, R.O.C.
Tel: 886-02-788-8989
Fax: 866-02-788-8029

Pinnacle Technologies Co. Ltd.
4F, No. 270, Sec. 3
Nan-Kang Road
Taipei, Taiwan, R.O.C.
Tel: 02 788 4800
Fax: 02 788 5969

CANADA

Vancouver, British Columbia

Farnell Electronic Services
8525 Baxter Place, Unit 101
Production Court
Burnaby, B.C. V5A 4V7
Canada
Tel: 604-421-6222
Fax: 604-421-0582

Semad Electronics
3700 Gilmore Way, #212
Burnaby, B.C. V5G 4M1
Canada
Tel: 604-451-3444
Fax: 604-451-3445

Pioneer/Zentronics
4455 No. 6 Road
Richmond, B.C. V6V 1P8
Canada
Tel: 604-273-5575
Fax: 604-273-2413

Future Electronics
1695 Boundary Road
Vancouver, B.C. V5K 4X7
Canada
Tel: 604-294-1166
Fax: 604-294-1206

Calgary, Alberta

Farnell Electronic Services
3015 - 5th Ave. NE
Suite 210
Calgary, Alberta T2A 6T8
Canada
Tel: 403-273-2780
Fax: 403-273-7458

Future Electronics
2015 32nd Ave. N.E., Unit 1
Calgary, Alberta T2E 6Z3
Canada
Tel: 403-250-5550
Fax: 403-291-7054

Pioneer/Zentronics
#560, 1212-31st Avenue N.E.
Calgary, Alberta T2E 7S8
Canada
Tel: 403-291-1988
Fax: 403-291-0740

Semad Electronics
6815 8th St. N.E.
Suite 175
Calgary, Alberta T2E 7H7
Canada
Tel: 403-252-5664
Fax: 800-565-9779

Edmonton, Alberta

Future Electronics
4606-97th Street
Edmonton, Alberta T6E 5NG
Canada
Tel: 403-438-2858
Fax: 403-434-0812

Pioneer/Zentronics
Plaza 124 #708
10216-124 Street
Edmonton, Alberta T5N 4A3
Canada
Tel: 403-482-3038
Fax: 403-482-1336

London, Ontario

Pioneer/Zentronics
148 York Street, Suite 209
London N6A 1A9
Canada
Tel: 519-672-4666
Fax: 519-672-3528

Toronto, Ontario

Farnell Electronic Services
300 N. Rivermede Road
Concord, Ontario L4K 2Z4
Canada
Tel: 416-798-4884
Fax: 416-798-4889

Semad Electronics
85 Spy Court
Markham, Ontario L3R 4Z4
Canada
Tel: 416-475-3922
Fax: 416-475-4158

Future Electronics
5935 Airport Road, Suite 200
Mississauga, Ontario L4V 1W5
Canada
Tel: 905-612-9200
Fax: 905-612-9185

Pioneer/Zentronics
3415 American Drive
Mississauga, Ontario L4V 1T4
Canada
Tel: 905-405-8300
Fax: 905-405-6423

Ottawa, Ontario

Farnell Electronic Services
39 Robertson Road, Suite 506
Bell Mews
Nepean, Ontario K2H 8R2
Canada
Tel: 613-596-6980
Fax: 613-596-6987

Pioneer/Zentronics
223 Colonnade Road, Suite 112
Nepean, Ontario K2E 7K3
Canada
Tel: 613-226-8840
Fax: 613-226-6352

Future Electronics
1050 Baxter Road
Ottawa, Ontario K2C 3P2
Canada
Tel: 613-820-8313
Fax: 613-820-3271

Semad Electronics
2781 Lancaster
Suite 302
Ottawa, Ontario K1B 1A7
Canada
Tel: 613-526-4866
Fax: 613-523-4372

Montreal, Quebec

Farnell Electronic Services
6600 Trans Canada Highway
Suite 620
Pointe Claire, Quebec H9R 4S2
Canada
Tel: 514-697-8149
Fax: 514-697-1210

Future Electronics/Branch
237 Hymus Boulevard
Pointe Claire, Quebec H9R 5C7
Canada
Tel: 514-694-7710
Fax: 514-695-3707

Semad Electronics
243 Place Frontenac
Pointe Claire, Quebec H9R 4Z7
Canada
Tel: 514-694-0860
Fax: 514-694-0965

Pioneer/Zentronics
520 McCaffrey Street
Ville St. Laurent, Quebec H4T 1N1
Canada
Tel: 514-737-9700
Fax: 514-737-5212

Quebec City, Quebec

Future Electronics
1000 Ave. St. Jean Baptiste
Suite 100
Quebec G2E 5G5
Canada
Tel: 418-877-6666
Fax: 418-877-6671

Pioneer/Zentronics
2954 Blvd. Laurier
Suite 100
Ste-Foy, Quebec G1V 4T2
Canada
Tel: 418-654-1077
Fax: 418-654-2958

Winnipeg, Manitoba

Pioneer/Zentronics
540 Marjorie Street
Winnipeg, Manitoba R3H 0S9
Canada
Tel: 204-989-1957
Fax: 204-633-9255

Future Electronics
106 King Edward
Winnipeg, Manitoba R3H 0N8
Canada
Tel: 204-944-1446
Fax: 204-783-8133

Farnell Electronic Services
Unit 250, 1625 Dublin Ave.
Winnipeg, Manitoba R3H 0W3
Canada
Tel: 204-786-2589
Fax: 204-786-2637

Distributors

EUROPE

Austria

Elbatex GmbH
Eitnergasse 6
A-1230 Wien
Austria
Tel: 43/1-86642-0
Fax: 43/1-86642-201

Baltic

Memec Baltic Ltd.
Brivibas Str. 148A / 709
LV-1012 Riga
Lithuania
Tel: 371-2-364684
Fax: 371-8-828062

Belgium, Netherlands

Sonetech Nederland BV
P.O. Box 259
5670 AG Nuenen
Netherlands
Tel: 31-40-837075
Fax: 31-40-832300

Sonetech Nederland BV
Office Belgium
Limburg Stirum 243, B-2
B-1780 Wommel
Tel: 32-2-4600707
Fax: 32-2-4601200

Memec Benelux, Bruxelles
Rue Saint Lambert 135
1200 Bruxelles
Belgium
Tel: 010-32-2778-9850
Fax: 010-32-2778-9858

Memec Benelux BV
Postbus 4903
NL-5604 CC Eindhoven
Netherlands
Tel: 31-40-659399
Fax: 31-40-659393

Denmark

Arrow Exatec AS
Mileparken 20E
2740 Skovlund
Denmark
Tel: 45-44-92-7000
Fax: 45-44-92-6020

England

Polar Electronics Ltd
Cherrycourt Way
Leighton Buzzard
Bedfordshire LU7 8YY
England
Tel: 44-1525-858000
Fax: 44-1525-858255

Eiger Technologies Ltd
6 Harvard Court
Winwick Quay
Warrington
Cheshire WA2 8LT
England
Tel: 44-1925-626626
Fax: 44-1925-626600

Farnell Electronics Svs Ltd
Edinburgh Way
Harlow
Essex CM20 2DF
England
Tel: 44-1279-441144
Fax: 44-1279-443417

Future Electronics Ltd.
Future House
Poyle Road, Colnbrook
Berkshire SL3 0EZ
England
Tel: 44-1753-763000
Fax: 44-1753-689100

Hawke Components Ltd.
26 Campbell Court
Bramley NR. Bassingstoke
Hantsire RG26 5EG
England
Tel: 44-1256-880800
Fax: 44-1256-880325

H.B. Electronics Ltd.
Lever Street
Bolton
Lancshire BL3 6BJ
England
Tel: 44-1204-555000
Fax: 44-1204-384911

Finland

Memec Finland
Asemankello
Vernissankatu 6
Fin 01300 Vantaa
Finland
Tel: 358-07001-9830
Fax: 358-07001-9839

France

Mecodis
Parc d'Activites
3 Allee des Erables
94042 CRETEIL Cedex
France
Tel: 33 1 43 99 44 00
Fax: 33 1 43 99 98 28

Farnell Electronic Services
BP 69 Saint Aubin
91192 GIF Sur Yvette Cedex
France
Tel: 33 1 69 85 83 00
Fax: 33 1 69 85 83 99

Scaib SA
6 rue Ambroise Croizat
ZI des Glaises - BP 58
91122 Palaiseau Cedex
France
Tel: 33 1 69 19 89 00
Fax: 33 1 69 19 89 20

Germany

Rutronik RSC-Halbleiter GmbH
Industriestr. 2
D-75228 Ispringen
Germany
Tel: 49/7231-801-0
Fax: 49/7231-82282

Future Electronics Deutschland GmbH
Muenchner Str. 18
D-85774 Unterfoehring
Germany
Tel: 49/89-95727-0
Fax: 49/89-95727-140

Semitron W. Roeck GmbH
Im Gut 1
D-79790 Kuessaberg
Germany
Tel: 49/7742-8001-0
Fax: 49/7742-6901

Avnet E2000 GmbH
Stahlgruberring 12
D-81829 Muenchen
Germany
Tel: 49/89-4 5110-01
Fax: 49/89-4 5110-210

Metronik GmbH
Leonhardsweg 2
D-82008 Unterhaching
Germany
Tel: 49/89-61108-0
Fax: 49/89-61108-155

Greece

P. Caritato & Associates S.A.
Ilia Iliou 31
Athens 11743
Greece
Tel: 30 1 902 01 15
Fax: 30 1 901 70 24

Hungary

Humansoft Ltd.
Angol u. 24/b
H-1149 Budapest
Hungary
Tel: 36/1162-2879
Fax: 36/1251-3673

Israel

Elina Electronics Ltd.
14 Raoul Wallenberg St.
P.O. Box 13190
Tel Aviv 61131 Israel
Tel: 972 3 49 85 43
Fax: 972 3 49 87 45

Italy

Eurelettronica SPR
Via E. Fermi 8
20090 Assago Milano
Italy
Tel: 39-2-657841
Fax: 39-2-26126270

Farnell Spa
Viale Milanofiori E/5
20090 Assago Milano
Italy
Tel: 39-2-824701
Fax: 39-2-82470278

Kevin
Via Dei Gradenigo, 3
20148 Milano
Italy
Tel: 39-2-48706300
Fax: 39-2-48706500

Norway

Berendsen
P.O. Box 9376 Gronlund
N0135
Oslo Norway
Tel: 47-22-677290
Fax: 47-22-677380

Portugal

Digicontrol
Dpt Comercial
Aven. Eng Arantes E. Oliveira 52d
1900 LISBOA
Portugal
Tel: 35 11 80 57 30
Fax: 35 11 849 03 73

Russia

Gamma Ltd.
Grazdanski PR-T
111 KOM 427, 429
Elektronmash Building
R-195265 St. Petersburg
Russia
Tel: 7/812-5311-180
Fax: 7/812-5311-402

Spain

Sagitron
Corazon de Maria 80/82
28002 Madrid
Spain
Tel: +34 1 416 92 61
Fax: +34 1 415 86 52

Sweden

Memec Scandinavia AB
Kvarnholmsvagen 52
131 31 Nacka
Sweden
Tel: 011-46-8643-4190
Fax: 011-46-8643-1195

Switzerland

Elbatex Gruppe AG
Hardstr. 72
CH-5430 Wettingen
Switzerland
Tel: 41/56-275100
Fax: 41/56-275454

Semitron W:Roeck & Co.
Promenadenstr.6
CH-5330 Zurzach
Switzerland
Tel: 41/56-493383
Fax: 41/56-493569

Avnet E2000 AG
Boehnrainstr. 11
CH-8801 Thalwil
Switzerland
Tel: 41/1-7221330
Fax: 41/1-7221340)

Turkey

Inter Muehendislik Danismanlik
Ve Ticaret A.S. 1
Hasircbasi Caddesi No. 55
81310 Kadikoy
Istanbul
Turkey
Tel: 90 216 349 94 00
Fax: 90 216 349 94 31

Distributors

MEXICO

Mexico, D.F.

Semiconductores Profesionales
Madrid No. 55
Col. Del Carmen Coyoacan
04100 Mexico, D. F.
Tel: 525-658-60-11
Fax: 525-658-60-44

Edo de Mexico

Electronica Seta
Galeana No. 114-20 Piso
La Loma Tlalnepantla
54060 Edo de Mexico
Tel: 525-390-77-13
Fax: 525-390-94-68

Guadalajara

Future Electronics de Mexico
S.A. de C.V.
Prol. Americas 1600
2 do Pisco
Guadalajara, Jalisco
Mexico 44610
Tel: 523-678-9281
Fax: 523-678-9200

SOUTH AMERICA

All Other Countries Except Brazil:

Ibars Electronics Corporation
10020 N.W. 6th Ct.
Pembroke Pines, Florida 33024
USA
Tel: 305-430-3740
Fax: 305-430-3763

Argentina

CIKA Electronica SRL
Av. de Los Incas, 4821
1427 Buenos Aires
Tel: 521-3188/8108
Fax: 523-6068

Electronica Elemon SA
Franklin D. Roosevelt, 5415
1431 Buenos Aires
Tel: 523-5555/3909
Fax: 522-7335

Brazil

Aplicacoes Eletronicas
Artimar Ltda.
Rua Marques de Itu N°. 70-10And.
CEP 01223
Sao Paulo, Brazil
Tel: 55-11-231-0277
Fax: 55-11-255-0511

Chile

Victronics Ltda.
Ricardo Matte Perez, 0307
Providencia
Santiago
Tel: 225-3788
Fax: 341-5578

UNITED STATES

Alabama

Future Electronics
4825 University Square, Suite 12
Huntsville, AL 35816
Tel: 205-830-2322
Fax: 205-830-6664

Pioneer Technologies
4835 University Square, Suite 5
Huntsville, AL 35816
Tel: 205-837-9300
Fax: 205-837-9358

Arizona

Bell Industries
10611 North Hayden Road
Building D, Suite 103
Scottsdale, AZ 85260
Tel: 602-905-2355
Fax: 602-905-2356

Future Electronics
4636 E. University Dr., Suite 245
Phoenix, AZ 85034
Tel: 602-968-7140
Fax: 602-968-0334

Pioneer Standard
1438 W. Broadwa, Suite B140
Tempe, AZ 85282
Tel: 602-350-9335
Fax: 602-350-9376

California

San Jose Area

Bell Industries
1161 N. Fairoaks Ave.
Sunnyvale, CA 94089
Tel: 408-734-8570
Fax: 408-734-8875

Future Electronics
1024 Marilyn Dr.
Mountain View, CA 94040
Tel: 408-232-1998
Fax: 408-433-0822

Future Electronics
2220 O'Toole Ave.
San Jose, CA 95131
Tel: 408-434-1122
Fax: 408-433-0822

Pioneer Technical Products
333 River Oaks Parkway
San Jose, CA 95134
Tel: 408-954-9100
Fax: 408-954-9113

Roseville

Bell Industries
300 Douglas Blvd., Suite 205
Roseville, CA 95661
Tel: 916-781-8070
Fax: 916-781-2954

Future Electronics
755 Sunrise Avenue, Suite 150
Roseville, CA 95661
Tel: 916-783-7877
Fax: 916-783-7988

Agoura Hills

Bell Industries
30101 Agoura Court, Suite 118
Agoura Hills, CA 91301
Tel: 818-865-7900
Fax: 818-991-7695

Future Electronics
27489 West Agoura Road, Suite 300
Agoura Hills, CA 91301
Tel: 818-865-0040
Fax: 818-865-1340

Pioneer Standard
5126 Clareton Dr., Suite 160
Agoura Hills, CA 91301
Tel: 818-865-5800
Fax: 818-865-5814

Irvine

Bell Industries
220 Technology Dr. #100
Irvine, CA 92718
Tel: 714-727-4500
Fax: 714-453-4610

Future Electronics
258 Technology, Suite 200
Irvine, CA 92718
Tel: 714-453-1515
Fax: 714-453-1226

Pioneer Standard
217 Technology Drive #110
Irvine, CA 92718
Tel: 800-753-5090
Fax: 714-753-5074

San Diego

Aegis Electronic Group, Inc.
1015 Chestnut Ave., Suite G2
Carlsbad, CA 92008
Tel: 619-729-2026
Fax: 619-729-9295

Bell Industries
5520 Ruffin Rd., Suite 209
San Diego, CA 92123
Tel: 619-576-3294
Fax: 800-444-0139

Future Electronics
5151 Shoreham Place, Suite 220
San Diego, CA 92122
Tel: 619-625-2800
Fax: 619-625-2810

Pioneer Standard
9449 Balboa Ave., Suite 114
San Diego, CA 92123
Tel: 619-514-7700
Fax: 619-514-7799

Distributors

Colorado

Bell Industries
1873 S. Bellaire St.
Denver, CO 80222
Tel: 303-691-9270
Fax: 303-790-4991

Future Electronics
12600 W. Colfax Avenue
Suite B110
Lakewood, CO 80215
Tel: 303-232-2008
Fax: 303-232-2009

Pioneer Technologies
5600 Greenwood Plaza Blvd.
Suite 201
Englewood, CO 80111
Tel: 303-773-8090
Fax: 303-773-8194

Connecticut

Bell Industries
1064 East Main Street
Meriden, CT 06450
Tel: 203-639-6000
Fax: 203-639-6005

Future Electronics
700 W. Johnson Ave.
Westgate Office Center
Cheshire, CT 06410
Tel: 203-250-0083
Fax: 203-250-0081

Pioneer Standard
Two Trap Falls #101
Shelton, CT 06484
Tel: 203-929-5600
Fax: 203-929-9791

Florida

Altamonte Springs

Bell Industries
650 S. North Lake Blvd. #400
Altamonte Springs, FL 32701
Tel: 407-339-0078
Fax: 407-339-0139

Future Electronics
237 S. Westmonte Drive, Suite 307
Altamonte Springs, FL 32714
Tel: 407-865-7900
Fax: 407-865-7660

Pioneer Technologies
337 South-North Lake, Suite 1000
Altamonte Springs, FL 32701
Tel: 407-834-9090
Fax: 407-834-0865

Deerfield Beach

Future Electronics
1400 E. Newport Center Drive
Suite 200
Deerfield Beach, FL 33442
Tel: 305-426-4043
Fax: 305-426-3939

Pioneer Technologies
674 S. Military Trail
Deerfield Beach, FL 33442
Tel: 305-428-8877
Fax: 305-481-2950

Largo

Future Electronics
2200 Tall Pines Drive, Suite 108
Largo, FL 34641
Tel: 813-530-1222
Fax: 813-538-9598

Georgia

Pioneer Technologies
4250 C Rivergreen Pkwy.
Duluth, GA 30136
Tel: 404-623-1003
Fax: 404-623-0665

Bell Industries
3020 Business Park Drive, Suite A
Norcross, GA 30071
Tel: 404-446-7167
Fax: 404-446-7264

Future Electronics
3150 Holcomb Bridge Rd.
Suite 130
Norcross, GA 30071
Tel: 404-441-7676
Fax: 404-441-7580

Illinois

Bell Industries
870 Cambridge Drive
Elk Grove Village, IL 60007
Tel: 708-640-1910
Fax: 708-640-1942

Future Electronics
3150 W. Higgins Rd., Suite 160
Hoffman Estates, IL 60195
Tel: 708-882-1255
Fax: 708-490-9290

Pioneer Standard
2171 Executive Drive #200
Addison, IL 60101
Tel: 708-495-9680
Fax: 708-495-9831

Indiana

Fort Wayne

Bell Industries
525 Airport North Office Park
Fort Wayne, IN 46825
Tel: 219-490-2100
Fax: 219-490-2104

Pioneer Standard
237 Airport N. Office Park
Fort Wayne, IN 46825
Tel: 219-489-0283
Fax: 219-489-6262

Indianapolis

Bell Industries
5230 W. 79th St.
P.O. Box 6885
Indianapolis, IN 46268
Tel: 317-875-8200
Fax: 317-875-8219

Future Electronics
8425 Woodfield Crossing
Suite 175
Indianapolis, IN 46240
Tel: 317-469-0447
Fax: 317-469-0448

Pioneer Standard
9350 N. Priority Way W. Dr.
Indianapolis, IN 46240
Tel: 317-573-0880
Fax: 317-573-0979

Kansas

Future Electronics
8826 Sante Fe Drive, Suite 150
Overland Park, KS 66212
Tel: 913-649-1531
Fax: 913-649-1786

Maryland

Bell Industries
8945 Guilford Rd., Suite 130
Columbia, MD 21046
Tel: 800-274-6953
Fax: 410-290-8006

Future Electronics
6716 Alexander Bell Drive
Suite 101
Columbia, MD 21046
Tel: 410-290-0600
Fax: 410-290-0328

Seymour Electronics
Columbia Business Center
6440 Dobbin Road, Suite B
Columbia, MD 21045
Tel: 410-992-7474
Fax: 410-992-7410

Pioneer Technologies
9100 Gaither Road
Gaithersburg, MD 20877
Tel: 301-921-3953
Fax: 301-921-4255

Massachusetts

Bell Industries
100 Burt Road #106
Andover, MA 01810
Tel: 508-474-8880
Fax: 508-474-8902

Future Electronics
41 Main Street
Bolton, MA 01740
Tel: 508-779-3000
Fax: 508-779-3050

Pioneer Standard
44 Hartwell Ave.
Lexington, MA 02173
Tel: 617-861-9200
Fax: 617-863-1547

Michigan

Grand Rapids

Future Electronics
4505 Broadmoor S.E.
Grand Rapids, MI 49512
Tel: 616-698-6800
Fax: 616-698-6821

Pioneer Standard
4467 Byron Ctr Rd SW
Grand Rapids, MI 49509
Tel: 616-534-3145
Fax: 616-534-3922

Detroit Area

Future Electronics
35200 Schoolcraft Road, Suite 106
Livonia, MI 48150
Tel: 313-261-5270
Fax: 313-261-8125

Pioneer Standard
44190 Plymouth Oak Blvd.
Plymouth, MI 48170
Tel: 313-525-1800
Fax: 313-427-3720

Distributors

Minnesota

Bloomington

Bell Industries
9400 James Ave. So. #142
Bloomington, MN 55431
Tel: 612-888-7247
Fax: 612-888-7757

Eden Prairie

Future Electronics
10025 Valley View Road, Suite 196
Eden Prairie, MN 55344
Tel: 612-944-2200
Fax: 612-944-2520

Pioneer Standard
7625 Golden Triangle
Eden Prairie, MN 56344
Tel: 612-944-3355
Fax: 612-944-3794

Thief River Falls

Digi-Key Corporation
701 Brooks Ave. So.
P.O. Box 677
Thief River Falls, MN 55344
Tel: 218-681-6674
Fax: 218-681-3380

Missouri

Future Electronics
12125 Woodcrest Executive Dr.
Suite 220
St. Louis, MO 63141
Tel: 314-469-6805
Fax: 314-469-7226

Pioneer Standard Electronics
111 Westport Plaza
Suite 600
St. Louis, Mo. 63146
Tel: 314-542-3077
Fax: 314-542-3078

New Jersey

Northern

Bell Industries
271 Route 46 West
Suites F202-203
Fairfield, NJ 07004
Tel: 201-227-6060
Fax: 201-227-2626

Pioneer Standard
14A Madison Road
Fairfield NJ 07006
Tel: 201-575-3510
Fax: 201-575-3454

Future Electronics
1259 Route 46 East
Parsippany, NJ 07054
Tel: 201-299-0400
Fax: 201-299-1377

Phase 1 Technology
295 Molnar Drive
Elmwood Park, NJ 07407
Tel: 201-791-2990
Fax: 201-791-2552

Seymour Electronics Corporation
357 Crossways Park Drive
Woodbury, NY 11797-2042
Tel: 201-465-7474
Fax: 201-465-0890

New Jersey (continued)

Southern

Bell Industries
158 Gaither Drive, Suite 110
Mt. Laurel, NJ 08054
Tel: 609-439-9009
Fax: 609-439-0570

Phase 1
12-B Ellipse Bldg. #221
4201 Church Road
Mount Laurel, NJ 08054
Tel: 609-234-3237
Fax: 609-234-5012

Pioneer Technologies
500 Enterprise Road
Keith Valley Business Center
Horsham, PA 19044
Tel: 215-674-4000
Fax: 215-674-3107

Seymour Electronics
520 Fellowship Road, Suite A104
Mt. Laurel, NJ 08054
Tel: 609-235-7474
Fax: 609-235-4992

Future Electronics
12 East Stow Road, Suite 200
Marlton, NJ 08053
Tel: 609-596-4080
Fax: 609-596-4266

Distributors

New Mexico

Bell Industries
11728 Linn N.E.
Albuquerque, NM 87123
Tel: 505-292-2700
Fax: 505-275-2819

New York

Binghamton

Pioneer Standard
1249 Front Street, Suite 201
Binghamton, NY 13901
Tel: 607-722-9300
Fax: 607-722-9562

Long Island

Future Electronics
801 Motor Parkway
Hauppauge, NY 11788
Tel: 516-234-4000
Fax: 516-234-6183

Phase 1 Technology
46 Jefryne Blvd.
Deer Park, NY 11729
Tel: 516-254-2600
Fax: 516-254-2693

Pioneer Standard
60 Crossways Park West
Woodbury, NY 11797
Tel: 516-921-8700
Fax: 516-921-2143

Seymour Electronics
357 Crossways Park Drive
Woodbury, NY 11797-2042
Tel: 516-496-7474
Fax: 516-496-0857

Upstate New York

Pioneer Standard
840 Fairport Park
Fairport, NY 14450
Tel: 716-381-7070
Fax: 716-381-5955

Future Electronics
300 Linden Oaks
Rochester, NY 14625
Tel: 716-387-9550
Fax: 716-387-9563

Future Electronics
200 Salina Meadows Parkway
Suite 130
Syracuse, NY 13212
Tel: 315-451-2371
Fax: 315-451-7258

North Carolina

Future Electronics
8401 University Executive PK.
Suite 108
Charlotte, NC 28262
Tel: 704-547-1107
Fax: 704-547-9650

Future Electronics
5225 Capital Blvd.
1 North Commerce Center
Raleigh, NC 27604
Tel: 919-790-7111
Fax: 919-790-9022

Pioneer Technologies
2200 Gateway Centre Blvd.
Suite 215
Morrisville, NC 27560
Tel: 919-460-1530
Fax: 919-460-1540

Ohio

Cleveland

Future Electronics
6009-E Landerhaven Dr.
Mayfield Heights, OH 44124
Tel: 216-449-6996
Fax: 216-449-8987

Bell Industries
31200 Solon Road Unit 11
Solon, OH 44139
Tel: 216-498-2002
Fax: 216-498-2006

Pioneer Standard
29125 Solon Road
Solon, OH 44139
Tel: 216-248-8710
Fax: 216-248-9166

Dayton

Bell Industries
446 Windsor Park Drive
Dayton, OH 45459
Tel: 513-434-8231
Fax: 513-434-8103

Pioneer Standard
4433 Interpoint Blvd.
Dayton, OH 45424
Tel: 513-236-9900
Fax: 513-236-8133

Future Electronics
1430 Oak Ct., Suite 203
Beavercreek, OH 45430
Tel: 513-426-0090
Fax: 513-426-8490

Worthington

Pioneer Standard
100 Old Wilson Bridge, Suite 105
Worthington, OH 43085
Tel: 614-848-4854
Fax: 614-848-4889

Oklahoma

Pioneer Standard
9717 E. 42nd St., Suite 105
Tulsa, OK 74146
Tel: 918-665-7840
Fax: 918-665-1891

Distributors

Oregon

Bell Industries
9275 S.W. Nimbus
Beaverton, OR 97005
Tel: 503-644-3444
Fax: 503-520-1948

Future Electronics
15236 N.W. Greenbrier Pkwy.
Beaverton, OR 97006
Tel: 503-645-9454
Fax: 503-645-1559

Pioneer Technologies
8905 SW Nimbus, Suite 160
Beaverton, OR 97008
Tel: 503-626-7300
Fax: 503-626-5300

Pennsylvania

Philadelphia Area

Bell Industries
158 Gaither Drive, Suite 110
Mt. Laurel, NJ 08054
Tel: 609-439-9009
Fax: 609-439-0570

Phase 1
12-B Ellipse Bldg. #221
4201 Church Road
Mount Laurel, NJ 08054
Tel: 609-234-3237
Fax: 609-234-5012

Pioneer Technologies
500 Enterprise Road
Keith Valley Business Center
Horsham, PA 19044
Tel: 215-674-4000
Fax: 215-674-3107

Seymour Electronics
520 Fellowship Road, Suite A104
Mt. Laurel, NJ 08054
Tel: 609-235-7474
Fax: 609-235-4992

Future Electronics
12 East Stow Road, Suite 200
Marlton, NJ 08053
Tel: 609-596-4080
Fax: 609-596-4266

Pittsburgh

Pioneer Standard
259 Kappa Drive
Pittsburgh, PA 15238
Tel: 412-782-2300
Fax: 412-963-8255

Texas

Austin

Future Electronics
9020 II Capital of Texas Hwy N
Suite 610
Austin, TX 78759
Tel: 512-502-0991
Fax: 512-502-0740

Dallas

Bell Industries
1701 Greenville Ave. #306
Richardson, TX 75081
Tel: 214-690-9096
Fax: 214-690-0467

Future Electronics
800 E. Campbell Rd., Suite 130
Richardson, TX 75081
Tel: 214-437-2437
Fax: 214-669-2347

Pioneer Standard
13765 Beta Road
Dallas, TX 75244
Tel: 214-386-7300
Fax: 214-490-6419

Houston

Future Electronics
10333 Richmond Ave., Suite 970
Houston, TX 77042
Tel: 713-785-1155
Fax: 713-785-4558

Pioneer Standard
10530 Rockley Road, Ste 100
Houston, TX 77099
Tel: 713-495-4700
Fax: 713-495-5642

Utah

Bell Industries
6912 S. 185 West, Suite B
Midvale, UT 84047
Tel: 801-561-9691
Fax: 801-255-2477

Future Electronics
3540 S. Highland Drive #301
Salt Lake City, UT 84106
Tel: 801-467-4448
Fax: 801-467-3604

Washington

Bell Industries
1715 114th Ave SE #208
Bellevue, WA 98004
Tel: 206-646-8750
Fax: 206-646-8559

Pioneer Technologies
2800 156th Ave., SE, Suite 100
Bellevue, WA 98007
Tel: 206-644-7500
Fax: 206-644-7300

Future Electronics
19102 N. Creek Parkway South
Suite 118
Bothell, WA 98011
Tel: 206-489-3400
Fax: 206-489-3411

Wisconsin

Future Electronics
250 N. Patrick Boulevard, Suite 170
Brookfield, WI 53045
Tel: 414-879-0244
Fax: 414-879-0250

Pioneer Standard
120 Bishop's Way, Suite 163
Brookfield, WI 53005
Tel: 414-784-3480
Fax: 414-784-8207

Bell Industries
W 226 N 900 Eastmound
Waukesha, WI 53186
Tel: 414-547-8879
Fax: 414-547-6547

Authorized Distributor for Obsolete Products

Rochester Electronics, Inc.
10 Malcolm Hoyt Drive
Newburyport, MA 01950
Tel: 508-462-9332
Fax: 508-462-9512



MICROCHIP

Factory Sales

AMERICAS

Corporate Office

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200 Fax: 602 786-7277
Technical Support: 602 786-7627
Web: <http://www.mchip.com/biz/mchip>

Atlanta

Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770 640-0034 Fax: 770 640-0307

Boston

Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508 480-9990 Fax: 508 480-8575

Chicago

Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 708 285-0071 Fax: 708 285-0075

Dallas

Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 214 991-7177 Fax: 214 991-8588

Dayton

Microchip Technology Inc.
35 Rockridge Road
Englewood, OH 45322
Tel: 513 832-2543 Fax: 513 832-2841

Los Angeles

Microchip Technology Inc.
18201 Von Karman, Suite 455
Irvine, CA 92715
Tel: 714 263-1888 Fax: 714 263-1338

New York

Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516 273-5305 Fax: 516 273-5335

San Jose

Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408 436-7950 Fax: 408 436-7955

ASIA/PACIFIC

Hong Kong

Microchip Technology
Unit No. 3002-3004, Tower 1
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T. Hong Kong
Tel: 852 2 401 1200 Fax: 852 2 401 3431

Korea

Microchip Technology
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku,
Seoul, Korea
Tel: 82 2 554 7200 Fax: 82 2 558 5934

Singapore

Microchip Technology
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65 334 8870 Fax: 65 334 8850

Taiwan

Microchip Technology
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2 717 7175 Fax: 886 2 545 0139

EUROPE

United Kingdom

Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44 0 1628 851077 Fax: 44 0 1628 850259

France

Arizona Microchip Technology SARL
2 Rue du Buisson aux Fraises
91300 Massy - France
Tel: 33 1 69 53 63 20 Fax: 33 1 69 30 90 79

Germany

Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Muenchen, Germany
Tel: 49 89 627 144 0 Fax: 49 89 627 144 44

Italy

Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Pegaso Ingresso No. 2
Via Paracelso 23, 20041
Agrate Brianza (MI) Italy
Tel: 39 039 689 9939 Fax: 39 039 689 9883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81 45 471 6166 Fax: 81 45 471 6122

9/05/95

Factory Sales



MICROCHIP

Printed in the USA, 9/95
© 1995, Microchip Technology Inc.

Microchip Technology Inc. • 2355 W. Chandler Blvd. • Chandler, Arizona USA • 602-786-7200 • FAX: 602-899-9210