

Инфракрасное дистанционное управление.

(с)2007, А. Топрес, Нетания.
<http://altor.sytes.net>

Общее.

Инфракрасное дистанционное управление (ИКДУ) применяется последние лет 30 практически во всей бытовой электронной аппаратуре. Иногда используется двусторонняя связь объекта управления и пульта (например – в кондиционерах с пульта передаются команды, а обратно – текущие параметры), но мы не будем ее рассматривать в данной статье. Также мы не будем рассматривать вопросы типа «что такое инфракрасный свет» (ИК) и т.п. Достаточно знать что это свет. Весьма небольшой мощности и невидимом человеческому глазу диапазоне (но прекрасно наблюдаемому любой видеокамерой или цифровым фотоаппаратом). Как и всякий свет, в земных условиях он распространяется по прямой, т.е. не огибает предметы, не проходит через непрозрачные объекты, но в той или иной степени от них отражается. Как и всякое излучение, характеризуется своей величиной (интенсивностью и т.п., но мы договорились не вдаваться в физику), которая зависит от типа источника и режима его работы, и падает с расстоянием. Поэтому расстояние, на котором можно уверенно принимать сигнал зависит от мощности источника и чувствительности приемника, плюс несколько побочных факторов.

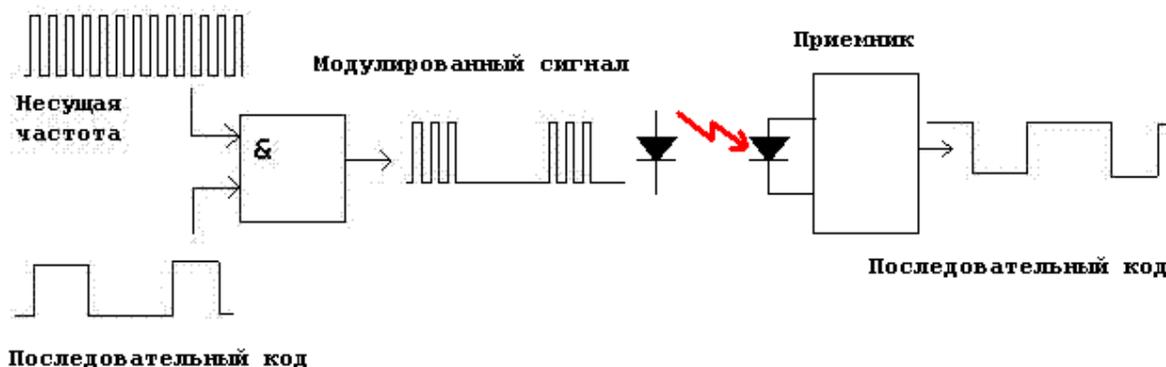


Для излучения используются ИК-светодиоды, работающие в импульсном режиме (ток в импульсе достигает нескольких ампер), для приема – фотодиоды с усилителями. Последние как правило, в аппаратуре закрываются ИК-фильтрами (на практике – красным или дымчатым пластиком) для снижения чувствительности к обычному солнечному или электрическому свету. Как всякий фильтр, он частично задерживает и полезное ИК излучение, поэтому от фильтра также зависит дальность работы ИКДУ

Почему в свое время было выбрано для массового использования именно ИКДУ а не что-то другое? Да, были (и есть) системы, основанные на радиосвязи. Возможно даже в будущем они станут преобладать над ИК-системами, благодаря развитию таких способов коммуникаций как Bluetooth, ZigBee, MiWi (это не описка, я не имел ввиду Wi-Fi!) и подобных. Но три с лишним десятилетия назад – ИК оказалось самым надежным, и что самое важное – самым дешевым видом беспроводной односторонней связи на короткие расстояния в пределах одного помещения. Ведь прежде всего оно начало использоваться в бытовых телевизорах. Наиболее массово ИК используется и поныне, однако среди ИКДУ царит полный балаган – каждая фирма разрабатывала свои собственные протоколы и работала на разных частотах модуляции. В результате мы имеем около 2-х десятков(!) совершенно несовместимых между собою систем, из которых наиболее массово, к счастью, используются 6-7.

Самые первые ИК-системы работали по принципу: «есть излучение/нет излучения». Т.е. передавалась одна команда – «включить» или «выключить». Естественно, такой вариант совершенно не функционален, и не обладает приемлемой помехозащищенностью. Первое – потребовало разработки специальных последовательных протоколов связи. Второе – использованию модуляции, т.е. грубо говоря – излучение (ток) светодиода модулируется дважды, сначала несущей частотой (в районе 30-60кГц), которая в свою очередь, модулируется последовательностью битов команд. Замечу что существовали ИКДУ системы и без модуляции, например протокол ИТТ.

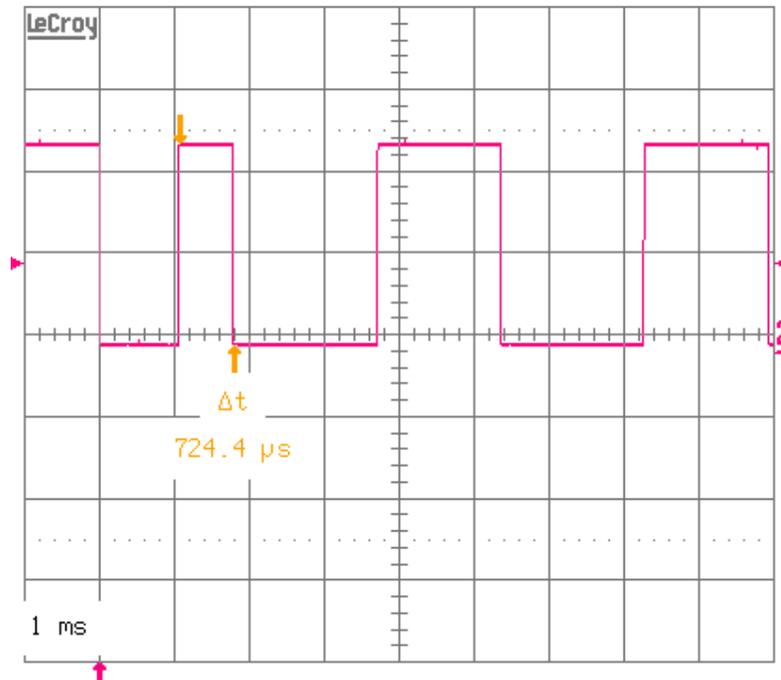
Исторически сложилось, что когда передается “1” это называют “Mark”, когда передача отсутствует – это называют “Space”. В рассматриваемых далее системах ИКДУ во время Space на светодиод ничего не подается, он выключен, а во время Mark – подается несущая частота. Т.е. светодиод вспыхивает с частотой 30-50кГц.



Со стороны приемника – сигнал усиливается и демодулируется. В результате на выходе имеем исходную последовательность импульсов. (Как правило, все современные приемники выдают инверсный сигнал, т.е. состояние Mark соответствует логическому нулю, а состояние Space – логической единице). В реальности – «ИК-приемник» несколько более сложное устройство чем просто фотодиод и усилитель. В нем имеется также ограничитель уровня (лимитер) и автоматическая регулировка усиления (АРУ), полосовой фильтр и демодулятор (детектор) с компаратором на выходе. Выпускаются готовые ИК-приемники такими фирмами как Vishay, Siemens, Sharp, а также многочисленными китайскими “тружениками”. Учтите, что с протоколами без модуляции такие ИК-приемники использовать нельзя, но мы такие и не будем рассматривать в этой статье (по разным причинам. В т.ч. и из-за присущих без модуляционным системам недостатков). Для правильной работы АРУ, в начале посылки практически во всех модуляционных протоколах передается “преамбула”.

Вряд ли стоит останавливаться на кодировании, т.е. изготовлении самого пульта ИКДУ – во первых, это сложно чисто конструктивно, ибо изготовить нормальный красивый пластмассовый корпус с кнопками и батарейным отсеком – задача не простая (и не для радиолюбителей). Во вторых – эта задача существенно проще задачи декодирования, достаточно взять описание протокола, и по нему все сделать. В декодировании сложнее, особенно, когда надо обеспечить работу с заранее неизвестным пультом (даже если известен стандарт, по которому он работает). В первую очередь потому, что разброс у пультов достаточно велик, и иногда даже выходит за рамки дозволенного. Простой пример – пульт от моего CD-плеера Philips CD800. Работает он по самому что ни на есть стандартному протоколу RC5, использующего манчестерское кодирование.

При таком кодировании, импульсы могут иметь только две длительности – T и $2T$, тем не менее, начальный импульс, который должен иметь одинаковые Mark и Space длительностью T , получается весьма несимметричный. Как видно на осциллограмме ниже – первый импульс чуть больше 1мс , затем пауза 0.74мс , затем широкий импульс примерно 1.8мс . По стандарту – первые два должны были бы быть 0.88мс .



Во вторую очередь – когда встает вопрос универсальности. Т.е. когда хочется иметь свой аппарат, который можно настроить на любой имеющийся пульт. Здесь есть несколько вариантов: или просто пульту говорится примерно следующее: «ты работаешь в кодировке NEC для телевизора», после чего микроконтроллер пульта все делает “по стандарту”. К сожалению, это имеет много недостатков. Например, я не смог настроить таким методом один такой “универсальный” пульт на мой телевизор – точнее вроде бы все работает, только кнопки “Громкость –” и “Громкость +” работают наоборот. Согласитесь, это несколько напрягает. (Там был еще какие-то мелочи, нестыковки, я уже не помню какие).

Другие варианты работают по принципу – “посвети имеющимся пультом в приемник”. В “универсальных пультах” как правило, используется тупая оцифровка импульсов с приемника, с выделением начала и конца посылки, и запоминание этой последовательности (иногда делаются простые преобразования для уменьшения объема данных) в памяти. Т.е. своего рода “цифровой магнитофон”. И лишь в некоторых делается по настоящему “разбор полетов” - анализ типа протокола и полное декодирование. Оба эти способа имеют как свои достоинства, так и недостатки. Первый способ “оцифровка” более простой, но требует достаточно большого запоминающего устройства для хранения оцифрованных данных (есть способы сильно сократить объем, но все равно он не мал), и хорошо работает когда нет ограничения на размер памяти и “хранителя” - например именно такой способ я использовал когда-то для управления видеоманитофоном от компьютера для записи рентгенограмм и УЗИ в медицинской диагностике. Аналогичное решение (и также на видеоманитофоне) использовали разработчики стримера “Арвид”.

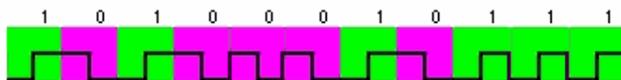
Для использования готовых “чужих” пультов своих конструкциях, (и при отсутствии памяти для запоминания оцифрованных последовательностей), наиболее интересна настройка на имеющийся пульт с “разбором полетов”, т.е. с полным декодированием. Поэтому в данной статье я постараюсь описать некоторые наиболее часто встречающиеся протоколы (кодировки), и способы программного декодирования на простых микроконтроллерах.

Описания протоколов.

В Интернете масса описаний очень распространенной филипповской кодировки RC5, но практически отсутствует информация по другим протоколам. Или же она настолько мизерна и разбросана по разным местам, что понять что-то сложно. Я решил собрать все что можно воедино, и дать насколько это возможно, наиболее полную информацию для того. Чтобы этим можно было пользоваться.

Не будем вдаваться в 7-уровневую модель OSI, разделим условно каждый протокол на две части – битовое кодирование и логический состав. Т.е. первое – это как кодируется передаваемый бит информации, а второе – что он означает. Рассмотрим сначала битовое кодирование. Все протоколы можно условно разделить на две категории – использующих бифазное кодирование (“Манчестерский” код) и использующие временной код.

При бифазном кодировании (используется в филипповских RC5 и RC6, Nokia NRC17 и других), каждый бит передается в течении фиксированного времени, значение бита – определяется направлением перехода в середине этого времени. На диаграмме ниже видно как это происходит – показана передача как разных бит, так и нескольких одинаковых бит подряд. Как мы видим – в середине битового интервала всегда есть переход, направление которого определяет значение бита. Между битами – переход есть только при передаче одинаковых бит. Это доставляет некоторые неудобства при декодировании, т.к. не всегда известен момент перехода от одного бита к другому. Было бы все просто, если бы длительности импульсов были всегда одинаковы (как это требуется по стандарту). Но на практике это не так, и разброс весьма существенный.

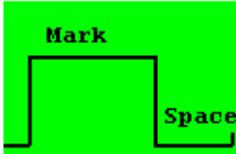


Кодирование временными интервалами (Sony, NEC, Samsung, etc.) предполагает ВСЕГДА наличие импульса на каждом передаваемом бите, а за ним паузы (или наоборот), что абсолютно четко определяет начало и конец передаваемого бита, а его значение определяется соотношением времени импульса и времени паузы (собственно, это разновидность ШИМ – широтно-импульсной модуляции с переменной частотой). В отличие от манчестерского кодирования, длительность передачи нуля и единицы разная, следовательно и общая длительность передаваемой команды может быть разной (есть способы этого избежать, но об этом ниже).



Почти во всех протоколах, в начале каждой команды передается стартовый бит (его еще часто называют «преамбула» или “AGC pulse” – импульс, для установки автоматической регулировки усиления в приемнике). Это импульс и пауза определенной длительности. К огромному счастью, у большинства протоколов временные параметры стартового импульса различны, что позволяет по первому же импульсу детектировать тип протокола (но учтите, что это не в 100% случаев!)

Ниже представлена таблица известных (мне J) длительностей преамбул:

	Протокол	Mark time, ms	Space time, ms	Примечание.
	NEC	9	4.5	Используется и многими другими
	NEC-Repeate	9	2.25	При повторе
	JVC	8.4	4.2	При повторе преамбула отсутствует
	Samsung	4.5	4.5	
	RCA	4	4	
	Panasonic (JAP)	3.6	1.5	
	Funai-2	3.2	3.2	
	RC6	2.67	0.89	
	Sony (SIRC)	2.4	0.6	
	RC5	0.89	0.89	
	Nokia NRC17	0.5	2.5	
	RC-MM	0.42	0.28	

В следующей таблице даны длительности информационных бит при кодировании временными интервалами:

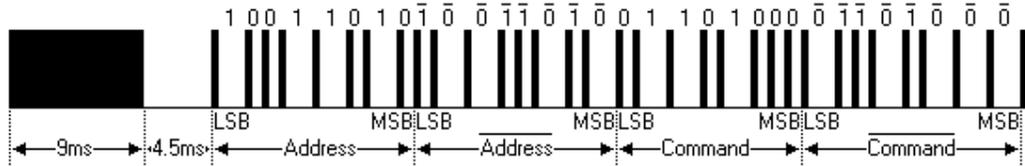
Протокол	1		0	
	Mark time, ms	Space time, ms	Space time, ms	Mark time, ms
NEC	0.56	1.6	0.56	0.56
JVC	0.53	1.6	0.53	0.53
Samsung	0.65	1.5	0.65	0.65
RCA	0.5	2	0.5	0.5
Panasonic (JAP)	0.4	1.2	0.4	0.4
Funai-2	0.8	2.4	0.8	0.8
RC6	0.45	0.45	0.45	0.45
RC6 Trailer	0.9	0.9	0.9	0.9
Sony (SIRC)	1.2	0.6	0.6	0.6
RC5	0.9	0.9	0.9	0.9
Nokia NRC17	0.5	0.5	0.5	0.5
RC-MM	0.17	0.28/0.44/0.62/0.78 (см. Описание протокола)		

В конце статьи будет представлена совмещенная таблица всех длительностей, для удобства.

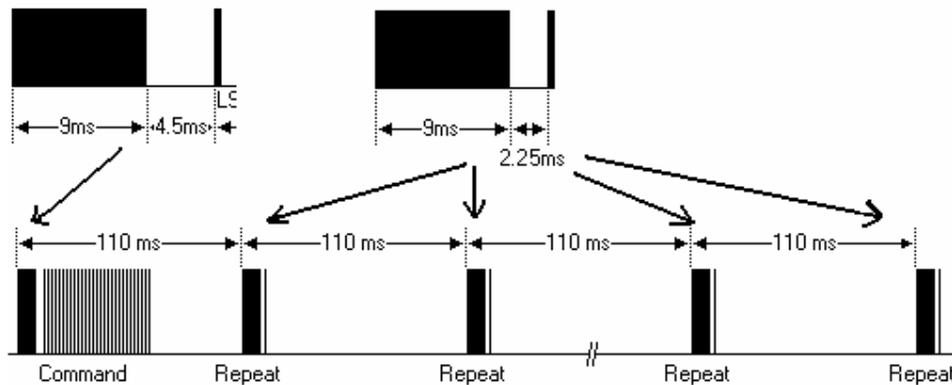
Протокол NEC.

Один из самых распространенных протоколов. Встречается в аппаратуре таких фирм, как Funai, Akai, Fisher, Goldstar, Hitachi, Kenwood, Onkio, Teac, Yamaha, Sanyo, Canon, Orion, Apex, Eltax, и много других. Этот протокол настолько распространен в аппаратуре из страны Восходящего Солнца, что его часто называют «японский протокол».

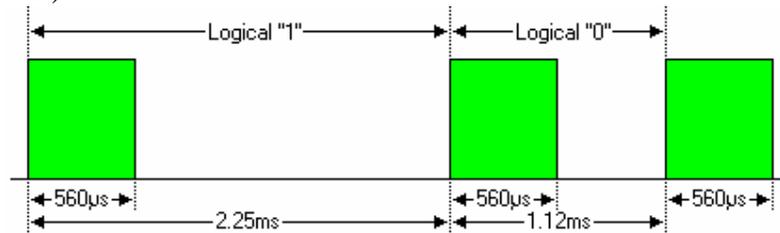
В каждой послылке передается 8 бит адреса и 8 бит команды, причем адрес и команда передаются дважды – в прямом и инверсном виде (это кроме проверки валидности передачи, делает одинаковой общую длительность любой послылки).



При непрерывном нажатии на кнопку, только первая послылка передается как показано на рисунке выше. Все остальные передаются в виде преамбулы с укороченной паузой и завершающим импульсом, с периодом 110мс:

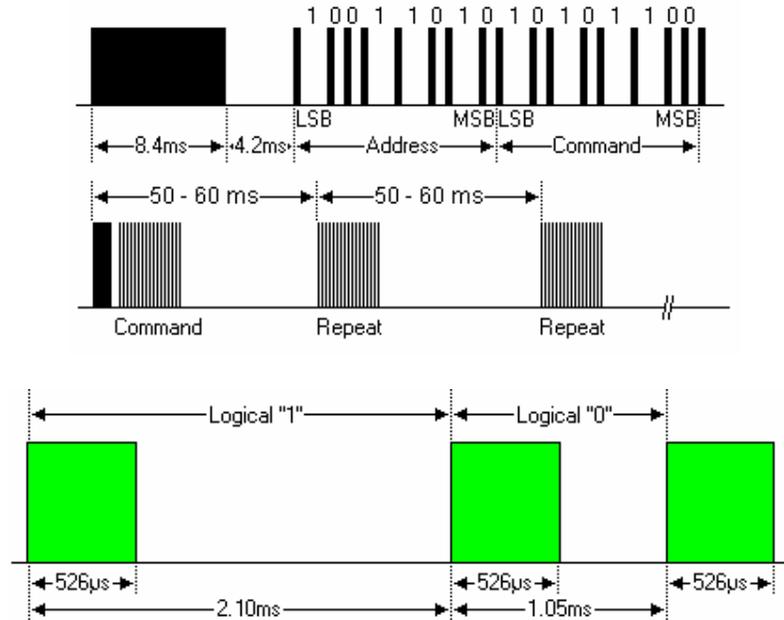


Кодирование бит – длительностью паузы (1.6мс или 0.56мс) между импульсами фиксированной длительности (0.56мс):



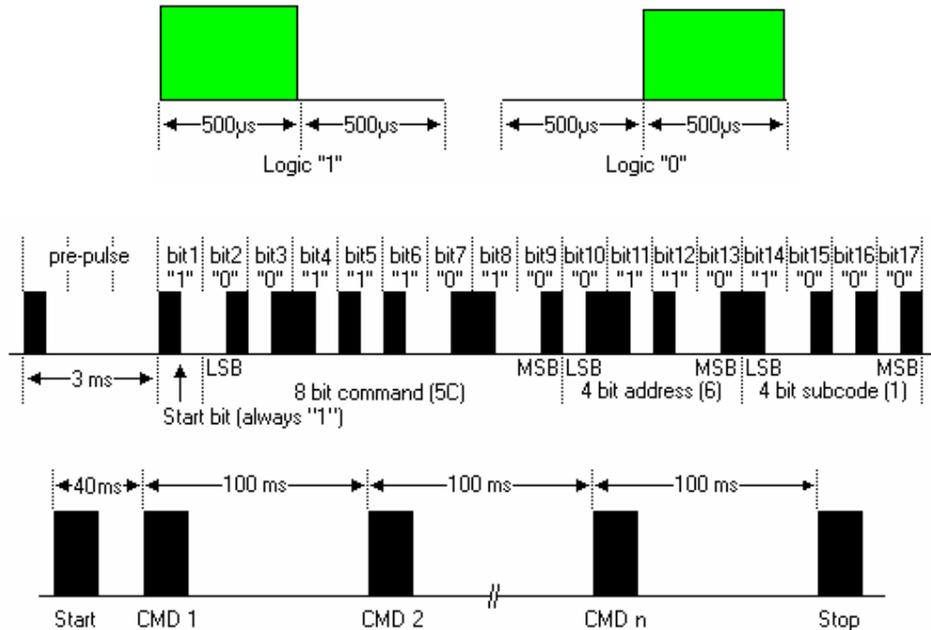
Протокол JVC.

Очень похож на NEC, и даже имеет сходные длительности. Но адрес и команда передаются только один раз. При повторе – посылка повторяется без преамбулы:



Протокол NOKIA NRC17

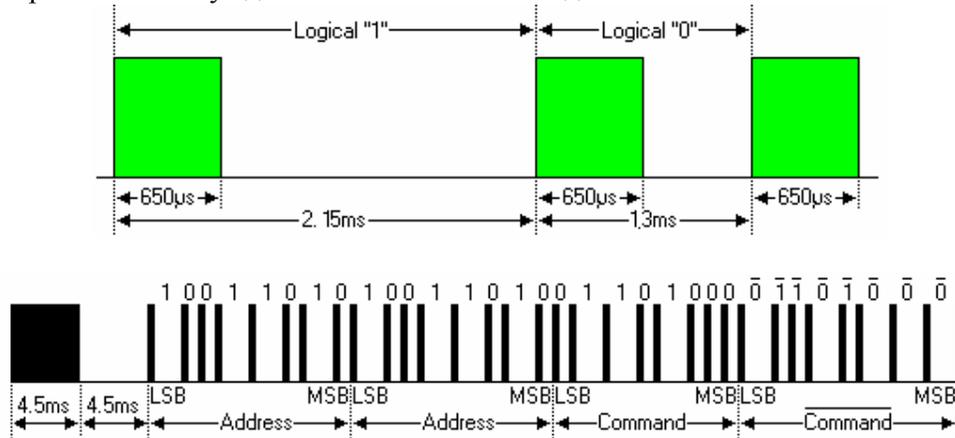
8 бит команды, 4 бита адреса и 4 бита “суб-кода” передаются бифазным (манчестерским) кодированием передаваемых бит:



Особенность протокола – для индикации разряженной батареи, преамбула передается с удлинненной на 1мс паузой (т.е. пауза не 2.5 а 3.5мс и длительность преамбулы 4мс вместо 3мс).

Протокол SAMSUNG.

Этот протокол также очень похож на NEC, но инверсным повторяется только команда. Адрес просто повторяется. Почему сделано именно так – загадка.



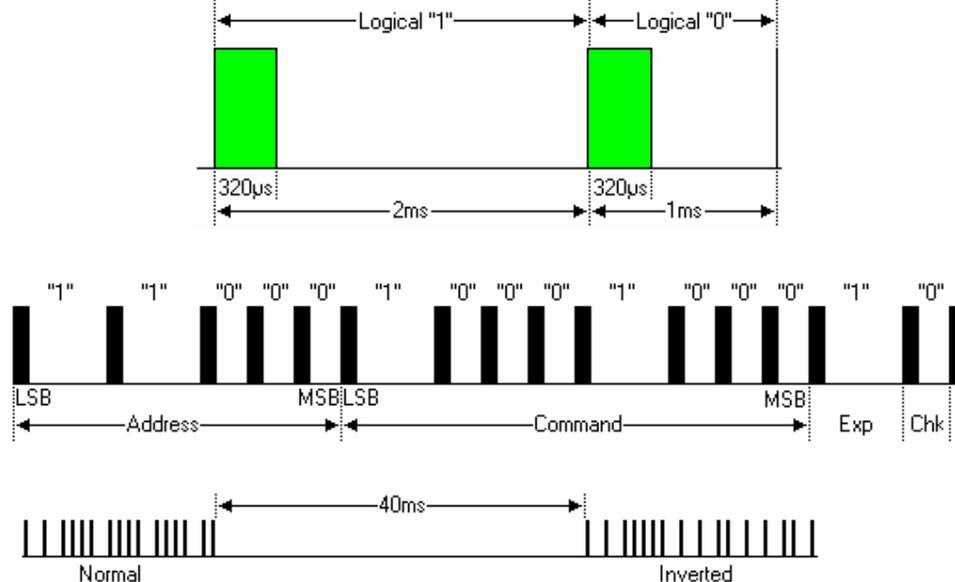
В качестве повтора используется преамбула и два импульса.:



Данный протокол очень любят использовать наши коллеги из “сарайчика дядюшки Ляо” в телевизорах марки “Noname”.

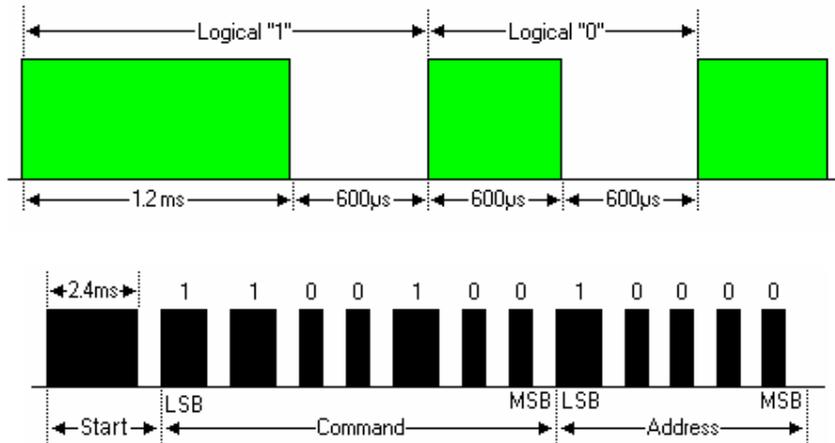
Протокол SHARP.

В отличие от других протоколов, в этом во 1-х отсутствует преамбула, а во 2-х - каждая команда передается двумя посылками, с периодом 40мс, при этом во второй посылке биты адреса и команды инверсные. Кодирование битов – ШИМ. В конце каждой посылки передаются дополнительные 3 бита – один для расширения системы команд и два контрольных.



Протокол SONY (SIRC).

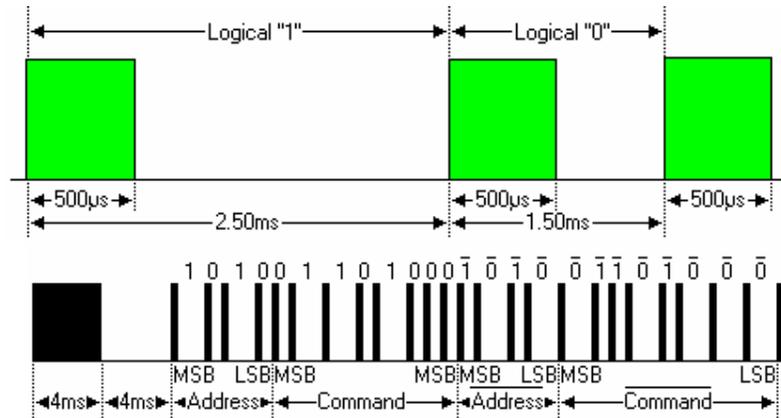
Существуют три версии данного протокола использующие соответственно 12, 15 и 20 информационных бит. Наиболее распространен 12-битный протокол, который состоит из 5 бит адреса и 7 бит команды, передающихся методом ШИМ, и предшествующей им преамбуле..



При непрерывном нажатии на кнопку, передаются одинаковые послылки с периодом 50-мс.

Протокол RCA.

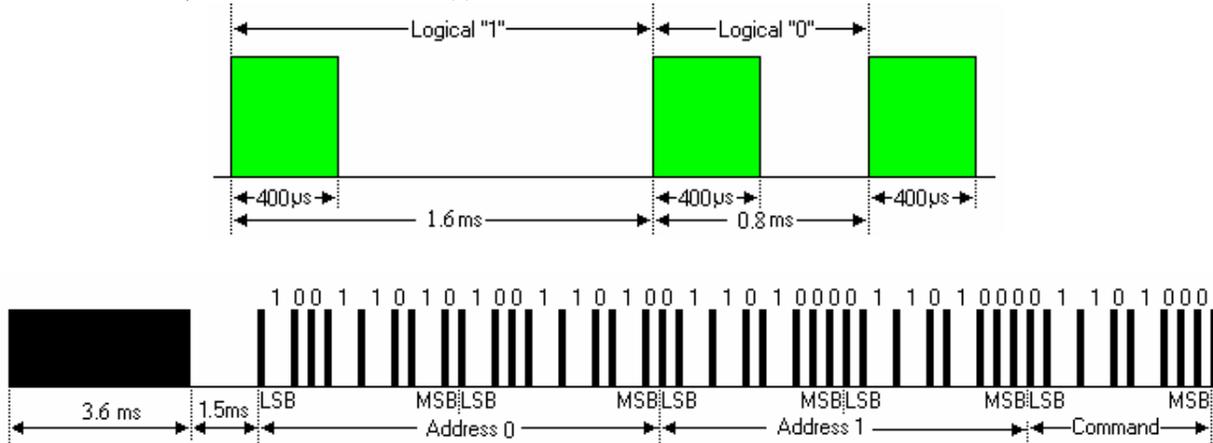
Кодирование – ШИМ, 4 бита адреса и 8 бит команды передаются в каждой послылке дважды, в прямом и инверсном виде. Это позволяет иметь неизменную общую длительность послылки.



При длительном нажатии – повторяется та же послылка с периодом 60мс.

Протокол Panasonic (REC80, JAP).

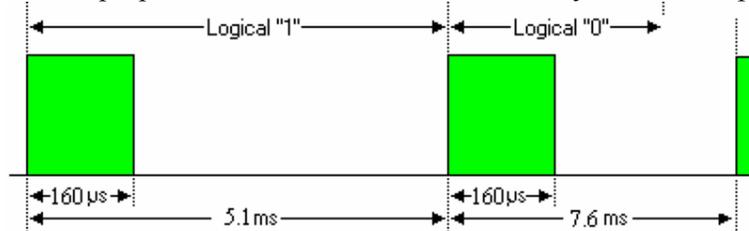
Очень «длинный» протокол – информационная длина 48 бит, из которых первые 32 бита являются постоянными для конкретного пульта, т.е. адресом. И только младшие 8 бит зависят от нажатой кнопки, т.е. являются командой.



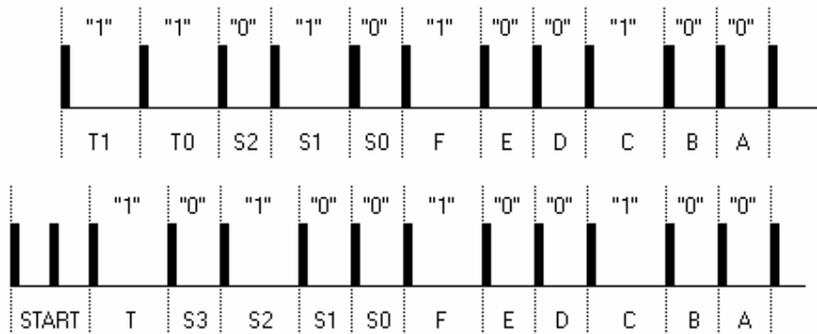
При длительном нажатии кнопки, повторяются одинаковые послылки через 70-80мс.

Протокол Philips RECS-80.

Хотя этот протокол и был разработан Филипсом, он не использует манчестерское кодирование:



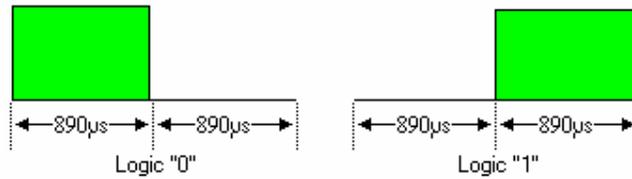
В целом, протокол довольно сложный, и имеет “обычный” и “расширенный” варианты. В обычном варианте передаются стартовый бит, “toggle bit”, 3 бита адреса и 6 бит команды. Расширенный режим отличается наличием двух стартовых бит и четырех бит адреса.



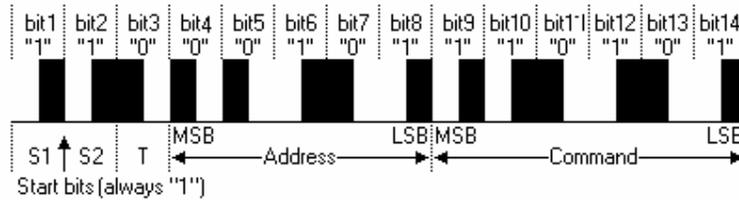
Филиппс как всегда отличился оригинальностью, и ввел почти во все свои протоколы т.н. “toggle bit”. Этот бит меняется каждый раз, когда кнопка была отпущена и снова нажата. При непрерывном нажатии кнопки – постоянно передается один и тот же код. Это позволяет легко отличить длительное нажатие от подряд идущих кратковременных. Даже если была нажата одна и та же кнопка. Способ безусловно неплохой, но как видим – все остальные прекрасно обходятся и без этого.

Протокол Philips RC5.

Один из самых распространенных протоколов в мире, Филипс RC5 использует уже знакомое нам бифазное (манчестерское) кодирование информационных битов.



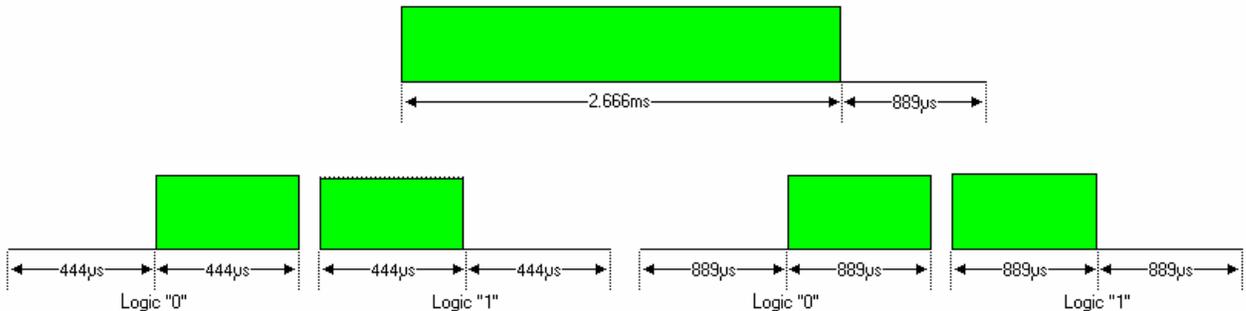
Протокол содержит 5 бит адреса и 6 бит команды, предваряется стартовым битом и битом “toggle”.



При длительном нажатии передаются одинаковые послылки, при отпускании кнопки – в следующей послылке инвертируется и бит toggle.

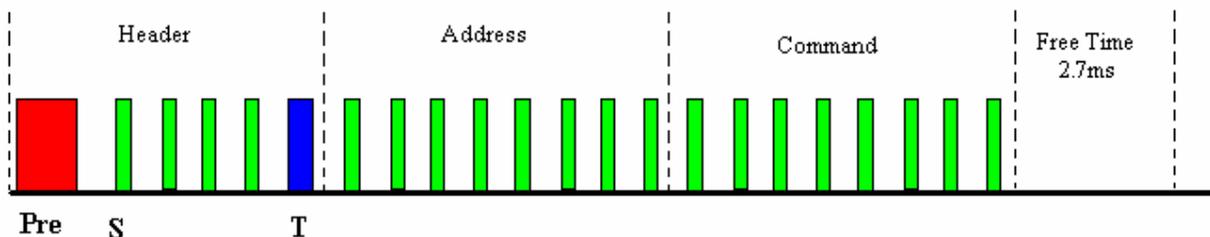
Протокол Philips RC6.

Немного более сложный чем RC5 - филипповский протокол RC6. Он использует преамбулу из длинного импульса и короткой паузы, за которой следуют информационные биты. Которые в свою очередь, могут быть одинарной или двойной длительности:



Каждая послылка состоит из четырех частей – заголовка, адреса, команды паузы. Заголовок начинается с преамбулы, за которой следует стартовый бит и три бита режима работы (все биты кроме преамбулы, перелаются одинарной длительностью, т.е. 444мкс.). Завершается преамбула битом “toggle”, который передается удвоенной длительности (890мкс.).

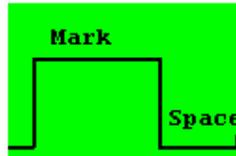
Далее следуют восемь бит адреса и восемь бит команды. Завершается каждая послылка обязательной паузой, длительностью 2.7мс.



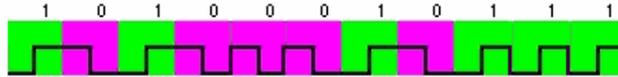
Программное декодирование.

Существуют много способов обработки сигнала с фотоприемника микроконтроллером или компьютером. Проще всего декодировать протоколы с кодированием длительностью – нужно измерить длительность единицы и нуля, после чего можно однозначно сделать вывод о передаваемом бите – единица это или ноль.

Аналогичным способом декодируется и стартовый бит (преамбула). В отличие от манчестерского кодирования, нам точно известно начало каждого передаваемого бита, и известно что должно быть изменение сигнала где-то между ними.



При манчестерском кодировании – обязательно только изменение состояния в середине бита, а вот на границе битов перехода может и не быть – если передаются разные биты:



Существует много алгоритмов декодирования манчестерского код, например алгоритм со «стейт-машиной» : <http://www.clearwater.com.au/rc5/>, можно и другими способами – суть от этого сильно не меняется. Я применил следующий алгоритм: считать длительности нуля и единицы, но запоминать при этом что было в прошлый раз, середина бита или граница между битами. И смотреть какой был предыдущий импульс – длинный (2T) или короткий (T). Мы совершенно точно знаем, что переход из 0 в 1 соответствует передаваемой единице, а переход из 1 в 0 – передаваемому нулю. Но это если он был в середине бита. Если же это переход на границе битов – его следует просто проигнорировать. Для определения где же был этот переход достаточно посмотреть на длительность импульса, а также помнить что детектировалось в последний раз – середина или граница.. Если импульс был длинный – мы однозначно находимся посередине бита (тут можно проверить – перед этим могла быть тоже только середина, иначе это ошибка!), и направление перехода указывает на передаваемый бит. Если импульс был короткий, а перед этим была середина – значит мы однозначно на границе битов и игнорируем переход (но запоминаем что он был на границе!). Если же импульс был короткий а перед этим была граница – значит мы на середине.

Для удобства реализации любых алгоритмов декодирования на микроконтроллерах (MCU), сигнал с фотоприемника нужно завести на вход внешнего прерывания и использовать таймер, присутствующим с большинстве MCU, для подсчета времени между фронтами.

В одних MCU прерывание вызывается по любому фронту сигнала – в этом случае нужно в процедуре обработки прерывания прочитать состояние входа прерывания, чтобы определить направление фронта (из 0 в 1 или из 1 в 0). Процедура обработки прерывания выглядит следующим образом:

```
void interrupt_ext(void){
    Dir = Int_pin;           // читаем состояние входа. Dir=1 после переднего,
                           // и Dir=0 после заднего фронта
    PTime = Timer;         // читаем содержимое таймера, таймер считает от 0 вверх
    ReloadTimer();        // перезапускаем таймер
                           // здесь выполняем остальные действия
}
```

В других – прерывание вызывается только по одному фронту, полярность которого определяется специальным битом внутри MCU. В этом случае – мы точно знаем по какому фронту пришло прерывание, но должны проинвертировать этот бит для приема следующего фронта другого направления:

```
void interrupt_ext(void){
    Dir = Int_Direction;    // читаем состояние бита направления фронта
    Int_Direction^=1;      // следующее прерывание будет по противоположному фронту
    PTime = Timer;         // читаем содержимое таймера, таймер считает от 0 вверх
    ReloadTimer();         // перезапускаем таймер
                            // здесь выполняем остальные действия
}
```

Далее я буду ссылаться именно на второй способ.

Если микроконтроллеру больше делать нечего, кроме как принимать и декодировать код – можно конечно и не использовать ни прерывание, ни таймер, если микроконтроллер будет достаточно быстрым, чтобы время выполнения следующего цикла не приводило к существенным погрешностям:

```
Pin_was = RC_PIN;          // запоминаем текущее состояние фотоприемника
While(Pin_was == RC_PIN){ // ожидаем изменения состояния фотоприемника
    Timer++;                // при этом увеличиваем программный таймер (счетчик)
                            // на единицу
}
PTime = Timer;             // после изменения состояния – запоминаем счетчик.
```

Однако, если прием и декодирование ИКДУ не является единственной задачей MCU – лучше все же использовать прерывание и аппаратный таймер. Как правило, при переполнении таймера может быть также сгенерировано прерывание – его удобно использовать для выхода из всевозможных ошибок. Например посередине посылки что-то случилось в пультом, или кто-то заслонил собою его луч. MCU может долго ждать следующего импульса (причем нужной полярности!), прежде чем он зафиксирует ошибку. Если же выбрать период таймера (т.е. время, за которое он считает от нуля до своего максимума) чуть больше максимально возможного периода импульсов, то при нормальной работе прерывания от таймера возникать не должно. Если же оно возникло – значит был сбой, и надо переинициализировать всю систему и ждать следующей посылки.

```
void interrupt_timer(void){
    Start_IRC();
}
```

В обработчике внешнего прерывания удобно применить стейт-машину, которая будет выполнять различные действия в зависимости от того, какой фронт мы ожидаем в настоящее время. Например:

```
enum stIR {stIRCWait, stIRCSyncStart, stIRCSyncEnd, stIRCDData1, stIRCDData0, stIRCDDataEnd};
```

Первое состояние stIRCWait соответствует ожиданию, точнее концу ожидания – приход первого фронта. Начало преамбулы. При этом нам еще не надо запоминать никаких таймеров, наоборот – нам нужно его запустить, и сказать программе что следующее состояние stIRCSyncStart будет концом первого импульса.

При возникновении прерывания в этом состоянии, нужно уже запомнить таймер – это будет длительность начала импульса преамбулы (S1), перезапустить таймер, и перейти в состояние stIRCSyncStart. С приходом сигнала прерывания в этом состоянии, значение таймера будет отражать длительность паузы преамбулы (S0). Нужно снова перезапустить таймер и перейти к состоянию определения информационных бит. В том случае, если преамбула правильная. Здесь может быть несколько вариантов – или мы сравниваем длительности S1 и S0 с заранее

известными константами, зависящими от того, по какому протоколу мы работаем, или же мы последовательно сравниваем со всеми вариантами, тем самым определяя сам протокол. Реально это выглядит так:

```
// процедура инициализации приема ИКДУ
void Start_IRC(void){
    Int_Direction = 0;           // ожидаем перехода из 1 в 0
    Disable_Timer();           // запрещаем работу таймера
    IR_State = stIRCWait; // устанавливаем начальное состояние стейт-машины
    Enable_Ext_Interrupt(); // разрешаем прерывание от фотоприемника
}

// процедура обработки таймерного прерывания
void interrupt_timer(void){ // если произойдет прерывание от таймера -
    Start_IRC();           // то начинаем все с начала
}

// процедура обработки внешнего прерывания
void interrupt_ext(void){
    Dir = Int_Direction; // читаем состояние бита направления фронта
    Int_Direction ^= 1; // следующее прерывание будет по противоположному фронту
    PTime = Timer; // читаем содержимое таймера, таймер считает от 0 вверх
    ReloadTimer(); // перезапускаем и стартуем таймер

    Switch (IR_State){
        case stIRCWait:
            IR_State=stIRCSyncStart;
            Break;

        case, stIRCSyncStart:
            S1 = PTime; // длительность импульса преамбулы
            IR_State=stIRCSyncEnd;
            Break;

        case stIRCSyncEnd:
            S0 = PTime; // длительность паузы преамбулы
            If ( // проверяем что это "наша" преамбула
                (S1>S1min) && (S1<S1max) &&
                (S0>S0min) && (S0<S0max))
                IR_State=stIRCDData1;
            Else // иначе – начинаем все с начала
                Start_IRC();
            Break;

        case stIRCDData1: // здесь делаем прием информационных бит согласно
        case stIRCDData0: // выбранному протоколу
            Break;

        case stIRCDDataEnd:
            IRC_Ready=1; //
            Start_IRC();
            Break;
    }
}
```

В примере выше производится сравнение преамбулы с заранее известной преамбулой выбранного протокола. Можно построить “лестницу” проверок, и выделить чему соответствует принятый первый импульс посылки:

```

Prot=prNone;
if((S1>=ST1_RC5_MIN)&&(S1<=ST1_RC5_MAX)&&
(S0>=ST0_RC5_MIN)&&(S0<=ST0_RC5_MAX)) Prot=prRC5; else

if((S1>=ST1_SIR_MIN)&&(S1<=ST1_SIR_MAX)&&
(S0>=ST0_SIR_MIN)&&(S0<=ST0_SIR_MAX)) Prot=prSIR; else

if((S1>=ST1_JAP_MIN)&&(S1<=ST1_JAP_MAX)&&
(S0>=ST0_JAP_MIN)&&(S0<=ST0_JAP_MAX)) Prot=prJAP; else

if((S1>=ST1_SAM_MIN)&&(S1<=ST1_SAM_MAX)&&
(S0>=ST0_SAM_MIN)&&(S0<=ST0_SAM_MAX)) Prot=prSAM; else

if((S1>=ST1_NEC_MIN)&&(S1<=ST1_NEC_MAX)&&
(S0>=ST0_NEC_MIN)&&(S0<=ST0_NEC_MAX)) Prot=prNEC;

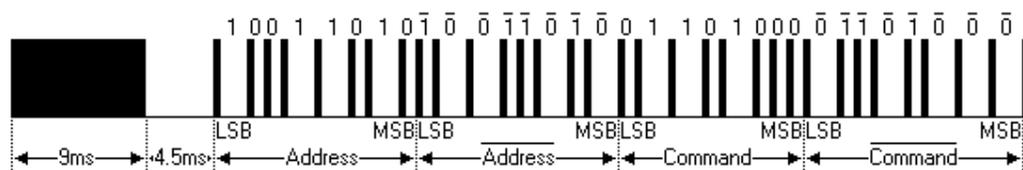
```

Общий алгоритм работы аппаратуры, настроенной на прием сигналов определенного пульта ИКДУ таков:

1. Ожидание начала посылки
2. Прием преамбулы (измерение длительности первого импульса и паузы за ним)
3. Определение “наша” преамбула или чужая. Если чужая – переход к началу.
4. Прием информационных бит согласно “нашему” протоколу.
5. Валидация принятых бит и преобразование принятых бит в более короткую форму, выделение адреса/команды, служебных бит (типа “toggle” для RC5), получение окончательного кода посылки.
6. Сравнение окончательного кода посылки с кодами команд, при соответствии – установка кода команды.

Рассмотрим п.5 – преобразование в более короткую форму. Как описано выше – разные протоколы имеют разное количество информационных бит. Одни используют 12, другие 32, а Панасоник вообще 48. Некоторая информация передается дважды. Но нам интересно знать только адрес и команду. Если устройство рассчитано на работу с одним конкретным протоколом, то длину окончательного кода можно выбрать совпадающим с длиной адреса и команды. Если же протокол заранее неизвестен – нужно выбирать максимально возможную. Некоторые протоколы имеют длину не более чем word (два байта), другие укладываются в long (два word или четыре байта), а Panasonic (JAP) передает аж три слова (шесть байтов). Но многобитные протоколы как правило, содержат избыточную информацию, которую можно использовать для валидации. Т.е. проверки правильности приема посылки.

Например в протоколе NEC:



Как мы видим, адрес и команды передаются дважды, в прямом и инверсном виде. Значит проверив прямое и инверсное значение адреса и аналогично команды, можно судить о

правильности приема. Но для дальнейшей обработки нам не нужны все 4 байта – достаточно двух.

Предположим, мы имеем две переменные - iRI и iResult, в первую записывается “сырой код”, т.е. принятые биты, во вторую помещается результат

```
#define unsigned int word
#define unsigned char byte

union{
    unsigned long   Input;
    word            In[2];
    byte            a[4];
} iRI;

union{
    word            Output;
    byte            a[2];
} iResult;
```

Подобная запись означает, что переменная iRI имеет длину long, и к ней можно обращаться как iRI.Input, как к 4-х байтовой переменной. Вместе с тем, к ней же можно обращаться по ее составляющим – как по словам, соответственно iRI.In[1] для старшего и iRI.In[0] для младшего. Или по отдельным байтам – от iRI.a[3] для старшего байта, до iRI.a[0] для младшего. Это позволяет оперировать как с переменной как одно целое так и по отдельности с ее частями. Если в переменной iRI находится принятая информация по протоколу NEC, то ее валидацию можно произвести следующим образом:

```
if (( iRI.a[3] == ~iRI.a[2]) && (iRI.a[1]==~iRI.a[0])) valid=1; else valid = 0;
```

После чего можно записать результат в другую переменную:

```
if (valid) {
    iResult.a[1] = iRI.a[3];           // адрес
    iResult.a[0] = iRI.a[1];         // команда
} else
    iResult.Output =0;
```

Надобность в подобной валидации и преобразовании, зависит также от того- в каком виде Вы запоминаете коды команд (с которыми потом сравниваете – см. п. 6). Т.е. сколько байт на каждую команды Вы выделяете. Компромиссным выбором, который пока меня не подводил, является выбор 4-х байт на каждую команду. При этом у 48-битового протокола жертвуем старшим (из 4-х) байтом адреса и можем не делать валидацию при приеме 4-х байтовых посылок, в которых половина или четверть (как у Samsung) являются инверсными – валидация автоматически произойдет при сравнении кодов. С одной стороны это несколько избыточно и расточительно (с т.з. расходования EEPROM, в котором обычно хранятся коды команд и времени на сравнение), с другой стороны – наиболее универсально. Сравнение же кодов делается циклическим перебором:

```
for ( i=0, j=eeCmdAddr, valid=0; i<MaxCmd; i++, j+=4)
    if ( (iResult.a[3]==EE_Read(j )) &&
        (iResult.a[2]==EE_Read(j+1)) &&
        (iResult.a[1]==EE_Read(j+2)) &&
        (iResult.a[0]==EE_Read(j+3)) )
        {valid=1; break;}
```

Здесь `eeCmdAddr` – начало области памяти (ЕЕПРОМ), где хранятся коды команд, `MaxCmd` - число команд. При завершении цикла, переменная `valid` равна 1, а переменная `i` – номер команды.

Некоторые протоколы (RC5) имеют отличное средство “toggle” для определения нажатия новой кнопки на пульте ИКДУ (тут “новой” считается не новая вообще, а любая, нажатая после отпускания предыдущей. Это может быть и та же самая кнопка), или специальный признак повтора. Для всех же остальных – это придется делать самим, программно. Запоминая код предыдущей команды и используя программный таймер автоповтора.

Для универсального (т.е. работающего по любому протоколу), это лучше делать одинаково для всех протоколов. Программная реализация автоповтора позволяет задавать различные периоды для разных кнопок, или отменять автоповтор вообще. Например для кнопок увеличения и уменьшения громкости, скорость автоповтора стоит сделать больше, чем для кнопок перебора каналов, для кнопки приглушения громкости (mute) – скорость лучше понизить или выключить вообще автоповтор. Для кнопки включения (Power, Standby, etc.) также лучше выключить автоповтор совсем или сделать его период побольше.

Сводная таблица длительностей преамбул и битов:

Протокол	Pre-Time		1		0		Примечание.
	Mark time, ms	Space time, ms	Mark time, ms	Space time, ms	Mark time, ms	Space time, ms	
NEC	9	4.5	0.56	1.6	0.56	0.56	Используется и многими другими
NEC-Repeat	9	2.25					При повторе
JVC	8.4	4.2	0.53	1.6	0.53	0.53	При повторе преамбула отсутствует
Samsung	4.5	4.5	0.65	1.5	0.65	0.65	
RCA	4	4	0.5	2	0.5	0.5	
Panasonic (JAP)	3.6	1.5	0.4	1.2	0.4	0.4	
Funai-2	3.2	3.2	0.8	2.4	0.8	0.8	При повторе преамбула отсутствует
RC6	2.67	0.89	0.45	0.45	0.45	0.45	
Sony (SIRC)	2.4	0.6	1.2	0.6	0.6	0.6	
RC5	0.89	0.89	0.9	0.9	0.9	0.9	
Nokia	0.5	2.5	0.5	0.5	0.5	0.5	
RC-MM	0.42	0.28					

Литература и полезные ссылки:

- <http://www.rotgradpsi.de/mc/etc/rc5decoder.html> - RC5 Decoder with μC
- <http://telesys.ru/electronics/projects.php?do=p036> - Леонид Иванович Ридико. Применение кода RC-5
- <http://home1.stofanet.dk/hvaba/fprc5rx/index.html> - field programmable RC5/Sony infrared remote receiver/decoder
- <http://www.swampgas.com/robotics/irremote/> - Easy, Cheap IR Remote Control Decoder for Robotics
- <http://www.sbprojects.com/knowledge/ir/ir.htm> - IR Remote Control Theory
- <http://www.epanorama.net/links/irremote.html#irremote> - Infrared remote control technology