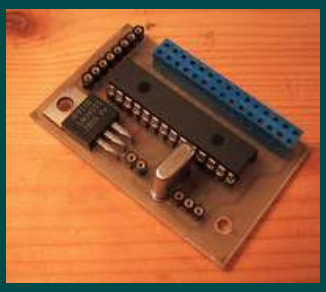


# FPRC5RX V0.9

## - field programmable RC5/Sony infrared remote receiver/decoder.

construction of a 16 bit parallel output learning Phillips RC5/Sony Sircs infrared decoder based on a 16F873 PIC processor in which IR codes are defined at run-time with a remote control.



| | 0 | Introduction | Top |
|---|---|---|---|

There exists a lot of very fine infrared receiver/decoder d.i.y. projects on the internet, so when I needed one for a homebuilt preamp, the obvious solution was to design yet another one. At least this increases the d.i.y. IR projects population by one :-) Since the project has matured it is probably also a candidate for simple home automation tasks and the like, but time will tell what it can actually be used for, if anything...

Initially I believed this project to be fairly innovative, but that only proves that I'm too naive :-) At a stage where this project is pretty stable I did the extremely stupid move to search for "learning ir decoder" on Google. Big mistake ! The number one hit were a pretty much similar project at MP3 IR Decoder/Remote Control made several years ago for a MP3 player control using a PIC16F84 ... So while I don't knowingly have stolen ideas and such (unless I write that I have) then I very likely haven't made - anything - new either... Please check the MP3 page, because they cirtainly were here first. But I still think this is a nice project though :-)

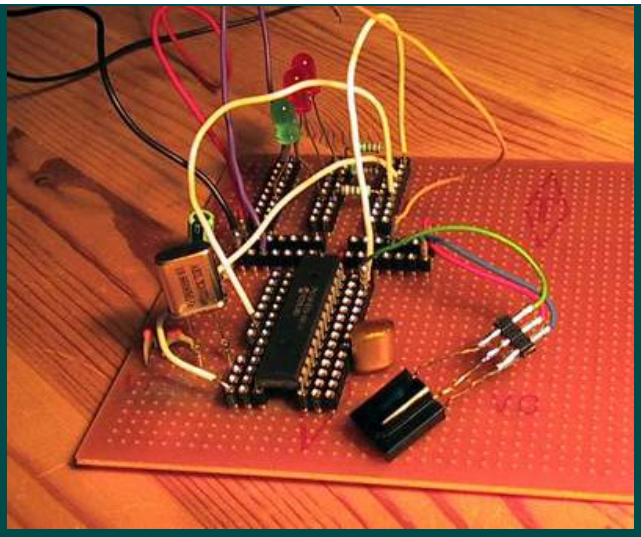


**Figure 0** : Test setup on experimental PCB with two output pins, the signal LED and the IR receiver connected. If you are able to program a 16F873 and have a suitable crystal and a IR receiver at hand you can be up and running in half an hour from - now - :-) Note : The PIC on the picture is an ancient prototype and it uses a another pin layout on the processor than is used below.

# Supported standards

The decoder handles both Phillips RC5 and Sony Sircs formats out-of-the-box and with some limitations also Panasonic REC-80.

| ***IR Standard*** | ***Format*** | ***Supported*** |
|---|---|---|
| *Phillips RC5* | *14 bit standard* | *Yes* |
| | *14 bit extended* | *Yes* |
| *Sony Sircs* | *12 bit* | *Yes* |
| | *15 bit* | *Yes* |
| | *20 bit* | *No (note 1)* |
| *Panasonic REC-80* | *48 bit* | *Limited. (note 2)* |

***Table 1***. *FPRC5RX decoder.*

Note 1 : Newer seen one, thats why :-)

Note 2 : Only the least 16 bits are used (i.e. stored in the PIC). I haven't found any REC80 protocol specs and perhaps the last 16 are not the most interesting bits at all. Obviously there is a risk of incorrect detections when only part of the code is verified.

The results from a web search regarding the Sony and Panasonic IR formats are quite inconsistent regarding timing specs. If people are right, then this construction is bound to come into trouble "somewhere". The timing schemes used are solely based on monitoring the remotes i have at hand with an oscilloscope.

# Features

What makes this decoder perhaps a bit different from others is that it has 16 ready-to-use digital outputs which in total can be assigned to up to more than 16 different IR codes (see downloads for actual capacity). It is therefore possible to have more than one IR code activating a given output pin, for instance when using several remotes. It is also possible to have a individual IR code assigned to multiple output pins if that can be used for anything meaningfull.....

Another feature is the programming of the IR codes, which takes place with the decoder at run-time and shooting at it with the remote controller(s) it is supposed to understand. All that is needed is physical access to the PIC as the programming is initiated by shorting pins on the processor. There are thus no IR remote codes hardcoded in the PIC from scratch, and you don't need to know the actual codes involved when pressing a given button on the remote. This is possible due to the non-volatile eeprom and flash memory in the 16F873 PIC processor.

It is possible to select how the output mode should be for each IR code programmed. (The normal high pulse while a key is pressed, a toggle mode, ...).

Actually the field programming capability is a idea I have stolen from an existing receiver chip. Unfortunately I don't remember who the manufacturer is, only that it was featured in a Elektor magazine. I'll have to check that out. Or perhaps it wasn't really that programmable, and what I saw was a construction using the infamous SAA3049 ?

(Update v0.5 : Its Elektor allright and it is "Learning RC5 Control Decoder 01 - 16" from January 2001. Found at http://www.elektor-electronics.co.uk/ln/ln.htm)

At present there is no possibility to get a readout of the assignment table if (when) you loose track of which channels go where allthough you can erase all asignments made to the individual output pins. Or you can reprogram the PIC to wipe it completely clean....

The serial I2C port available on the PIC processor is used for broadcasting information about pin number and new level whenever a output level changes.

# Vocabulary

**Pin** : Or channel or output pin, one of the 16 output pins on the PIC. Each pin can have multiple IR codes assigned.

**Assignment** : An assignment is a link between a given IR code and a given pin the code corresponds to.

**Mode** : An assignment has a mode, which is how the output pin works. (pulse, toggle etc)

**Programming** : Obviously the PIC processor has to be programmed on its own with a dedicated program. This program is the one that can be found under downloads. Note that when the term programming is used on this page, it most likely refers to the process of getting the IR codes beamed into the PIC (which is done at run-time).

# Future

The project here is currently on hold, replaced by other interesting projects, so right now nothing new is likely to happen. I have got some requests though, and that obviously feels nice, so keep them coming :-)

#1. Request for a audible 'beep' on recognized IR code. This would probably be a new mode to program on a output pin.

#2. Request for a standby mode where one pin would be the master controlling wether the rest of the decoder were in standby or active. This pin would then control the main power supply as well.

#3. Why not use the AUX pin (RA4) to something ??!??

# Recent changes

V0.8 – april 9, 2004. Added a little more info about LEDs....

V0.9 – april 18, 2005. Feedback on a IR reciever and a link corrected..

| | | | |
|---|---|---|---|
| | *1* | *Field programming* | *Top* |

Actually a more appropiate title would have been 'Learning' as this is the term used for intelligent remotes, but for now I'll stick to my initial term field programming.

The programming is quite simple, all you need is a piece of wire hooked up to Vcc and a suitable Phillips, Sony or Panasonic remote. If you really dislike shortcircuiting a PIC output pin like this then try to replace the wire with a 22 ohms resistor. It seems to work. Remember that in order to program you will need a power supply capable of supplying more than a PIC output can sink. 100mA or above should be fine.

A supplying note on this output pin shortening method for user inputs might be in place. What happens (if the PIC is running as it should) is that whenever a output pin is read by the software, then the output pin will change output level to the one currently present regardless of what was actually programmed by the software. This is due to the hardware design of a PIC I/O port. What it means is that the idle loop that runs continuesly in the software will automatically remove the shortcircuit condition inflicted by the user whenever it get around to scan the ports for user interactions. The good news is that this happens 'very often', so the warnings below about beeing quick to remove the shortcircuiting wire are just a precaution in case the PIC for some reason has gone heywire.

All notions of a flashing LED assume that a standard LED is connected to the LEDA output.

# Adding an IR code

## ◊ 1 Turn on power.

Briefly let the +5V wire touch the signal LEDA output (i.e. directly at pin 28) and LEDA will flash wildly in about 2 seconds - the PIC is now in programming mode (as opposed to normal run mode). All 16 outputs are now driven active low. See *Note1*

## ◊ 2 Connect the output pin (one of 16) to be programmed to Vcc, and the signal LED turns on.

This procedure effectively shortcircuits a low driven output pin to the positive supply. Don't panic, but at the other hand, don't leave the short on :-)

### ◊ 3 Select mode.

You are now in a endless loop where you can select the mode you want for the assignment in progress. You change mode in the 'shortly short a pin' fashion. See *Output pin mode tables*. Step to point '4' below once you have selected the mode you want or just to accept the default mode right away (most likely just a monostable output pin).

### ◊ 4 Press the key on the remote that should activate this pin in the current mode.

If the signal LEDA starts to flash very slowly and newer stops again then see error codes section below. The normal behaviour is a quick LEDA flashing with a frequency of about 'now I recieve a code that I recognize' Hz. If no flashes are seen, then the most likely cause is that the remote is a not a RC5 or Sony type after all....

### ◊ 5 Release remote, the signal LED turns off, and the programming is complete.

The above sequence can be executed for all channels and more than one time for a single channel as well. A new assignment is stored in the flash right after point "5" above, meaning that you can just remove/apply power as you please, and still keep on adding new assignments whenever you feel for it. You only need to execute "1" once when programming a bunch of codes in one session. Not surprisingly you leave programming mode by removing the power again....

***Note1*. Extended pins**. Since the pin number associated with a IR code is transmitted in the serial I2C packets (see below) it actually makes some kind of sense to extend the pin range to include extended pins 17-32 as well. If you let the Vcc wire touch the signal LEDA after the point ◊1 (i.e. a second time) then you still can use the pins 1-16 during steps ◊2-◊5 as programmming inputs, but the pin numbers saved internally will now be called 17-32. That is, they will not affect the physical pins 1-16 when running in normal mode, but they will get their state transmitted over the I2C bus to an external listener. When you enter the extended pin range there will be another run of wild flashing LED, only this time it takes approx. 4 seconds. The extended mode were introduced in order to control the stepper motor volume control (see links) via the i2c bus, i.e. without using any of the 16 output pins.

## Deleting a channel

### ◊ 1 Turn on power.

Do this while the signal LEDA is shorted to the PIC Vcc (+5V) line. This is a pretty mean way to power up a PIC since it is trying to drive the same line low, so remove the short after power on. The signal LED will flash in about 5 seconds and the PIC is now in delete mode.

### ◊ 2 Connect a output pin (one of 16) to be deleted to Vcc.

All IR codes assigned to the channel will be purged. Repeat this step for the channels to cleaned up, you get a few signal LED flashes as an confirmation after each delete.

### ◊ 3 Turn off power.

## Delete everything

### ◊ 1 Turn on power.

See description under 'Deleting a channel'.

### ◊ 2 Connect signal LED output to Vcc.

I.e. a second time. All user data in the PIC are deleted and the PIC will act as if it had just been programmed. Since this is point of no return the signal LED will just enter a infinite flashing loop until you remove power again.

### ◊ 3 Turn off power.

## Error codes

Some erroneous situations are signalled by the PIC entering an infinite loop with the following number of flashes of the LEDA spaced by a space. The only recovery is to flip the power off and then on again after which everything is as it was before the error occured. The blinking rate is slow enough to be seen with the eye.

| *Flashes* | *Meaning* |
|---|---|
| *3* | *Recieved IR code allready assigned to this pin* |
| *4* | *No room left, all available assignment slots occupied. (see downloads for actual capacity)* |

*Table 1.1. Error table.*

# Backup

Just upload the entire eeprom and flash from the PIC and save it.


| | 2 | Modes & Codes | Top |
|---|---|---|---|


# Modes

These are the current library of available output modes available for each of the pins on the PIC processor. Note however that it is a build time decision which of these will actually be included in a binary release, see *Output pin mode tables*.

Notice that the output levels during normal use are saved in eeprom on the fly and restored at power up. (relevant to all but normal mode pins).

## Normal :

The output pin is high only while the remote is transmitting the matching code, else its low. (ex : "turn motor up"). Also known as monostable or non-latching mode. In order not to get a series of pulses while a keycode is repeated from a depressed key, a timeout slight larger than the largest expected pulse to pulse time must expire before the output goes low (i.e. > 113.8 millisecs as found in RC5)

## Toggle :

The output changes state every time the remote starts transmitting the matching code. (ex : "lamp on" "lamp off" "lamp ...). Also known as bistable or latching mode.

## Radio :

This mode only gives sense when applied to two or more pins. The name is taken from a radio with pushbuttons in the channel preselector, where only one output at a time will be high. Once a new button (pin) is selected the previously selected pops out again. (Note that there is a small overlap in time from the newly selected pin goes high to the previous high pin goes low in order to get make-before-break when using relays to switch audio paths). There is only one common radiogroup available. Also known as a 1-of-N decoder. During normal run mode the radio group can also be controlled by shorting radio pins to +5V, just as it was done during programming mode. An example of a small radio group wiring can be seen *here*.

## BiToggle :

This is a genuine bastard mode, so if you expect the above modes to be enough then just skip this chapter. This mode was needed in order to have a fprc5rx controlling an audio preamplifier. What is provided is a normal run mode where a toggle output pin can be set both electrically and from a recieved IR code, just as for the radio mode above. The problem is however how to make a push button both drive the pin low and high for release and activate respectively. The solution here is to use two pins, called master and slave. The masters are the toggle output pins which have to be one of output pins 9-12. They toggle as would be expected from matching assigned IR codes just as normal toggle mode pins. The slave pin for a master is the pin 4 counts higher, i.e. pins 13-16. These are configured as inputs in normal run mode and should have a pull up **(v0.6)** resistor connected. A master will now also toggle each time its slave input gets a low **(v0.6)** pulse. To make it more hairy and more phrone to complete chaos, it is still possible to assign modes to the slave inputs. The state of these will only by transmitted over the I2C serial wire for (relatively) obviously reasons. An example of an BiToggle master/slave wiring can be seen *here*.

When programming a bi-toggle pin-pair then use the master pins (9 to 12) for the programming. If you try to assign a bi-toggle mode to another pin than 9 to 12 then please tell me what happens :-)

## BiToggleDelayed :

The BiToggleDelayed mode is a preamplifier requirement for controlling output mute relay(s) if installed. Please notice that the phase of this mode means that it should be interpreted as high level = play and not as high level = mute.

It is equal to the BiToggle above except for a 3 second (or thereabout) delay at power up where the output is forced low regardless of the last active level retrieved from eeprom. So if the PIC were powered off while a pin in this mode were high, then this pin will go high after a 3 second delay at power up. If you interact with a BiToggleDelayed pin in the delay time after power on, then the delayed setting stuff is cancelled.

You can basically have all modes on all pins, but I will leave it up to you to keep the system sane... For example you can have a toggle mode mixed with an radio mode on the same pin. This makes it possible to have two selected pins in the radio group (the correct one, and a subsequent toggle in one of those that were off), both of which will go low when the next radio pin is selected. This at least makes sense to me :-) Said in another way : when a code/pin combination is discovered to run in radio mode, then all other pins with radio functionality are affected (i.e. cleared), where as for normal and toggle mode it is only the individual pin that is modified. Hmm.

# Output pin mode tables

The idea with 'output pin mode tables' is that such a table becomes one of the ingredients in a build. As there are only one table in play at the moment this concept is pretty thin :-)

**'Standard'**

| *Mode* | *LED pulses (off blinks)* |
|---|---|
| *Monostable 'Normal'* | *"1"* |
| *Bistable 'Toggle'* | *"2"* |
| *Radio button* | *"3"* |
| *BiToggle* | *"4"* |
| *BiToggleDelayed* | *"5"* |

**Table 2.1**. Output pin mode table 'Standard'.

# Serial I2C

The serial I2C port from the PIC is used for broadcasting levelchanges on the output pins. The PIC acts as I2C master and transmits 3 bytes (address/command/data) on a output level change, see the following table. The speed is 400 kbit/sec which probably makes the I2C bus as implemented here unreliable for other purposes than local talk in closed environments (i.e. Enclosures).

| *Address* | *Command* | *Data* |
|---|---|---|
| *FPRC5RX broadcast (7bit) : 0x34* | *0 - Level went low* | *Pin number minus one, 0..15 + 16-31* |
| *-* | *1 - Level went high* | *-* |

**Table 2.2**. Serial I2C 3 bytes package.

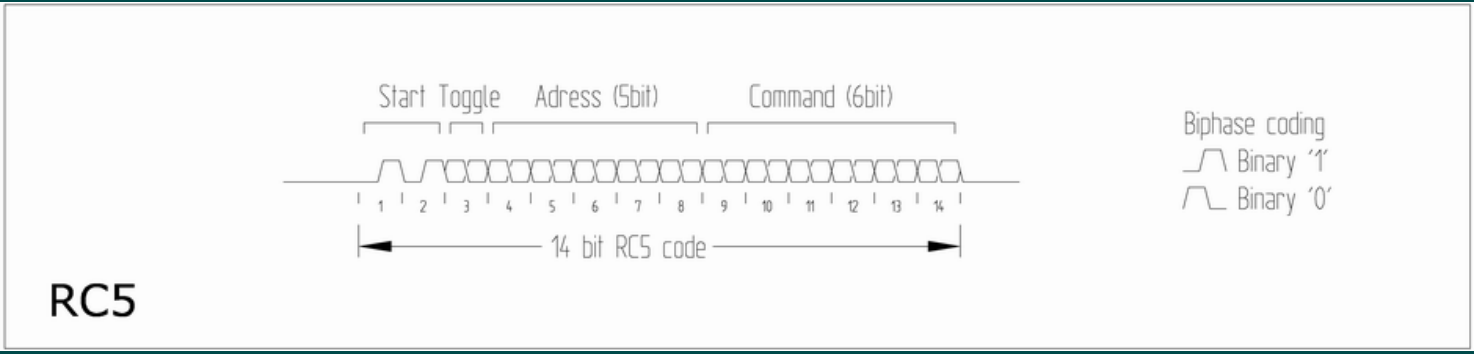


This section is meant as an introduction into the Phillips, Sony and Panasonic IR standards, without even attempting to write everything there is to write. If I tried I would fail anyway. If you want to know more I recommend the internet and/or the link section at buttom.

Please note that the figures, values and everything else here is lifted from other diy pages and/or comes directly from an oscilloscope....

# RC5 - PHILLIPS

The following figure shows the format of the 14 bit biphase RC5 code. The toggle bit is what makes the RC5 really neat. It changes value whenever a button is pressed down, meaning for instance that a TV set won't go to channel 11 if you press 1, and while it is still depressed, let your hand briefly interrupt the IR. The toggle bit will be the same in both instances and a clever decoder will ignore the second sequence. (Currently the decoder described on this page is then - not - clever by this definition :-)



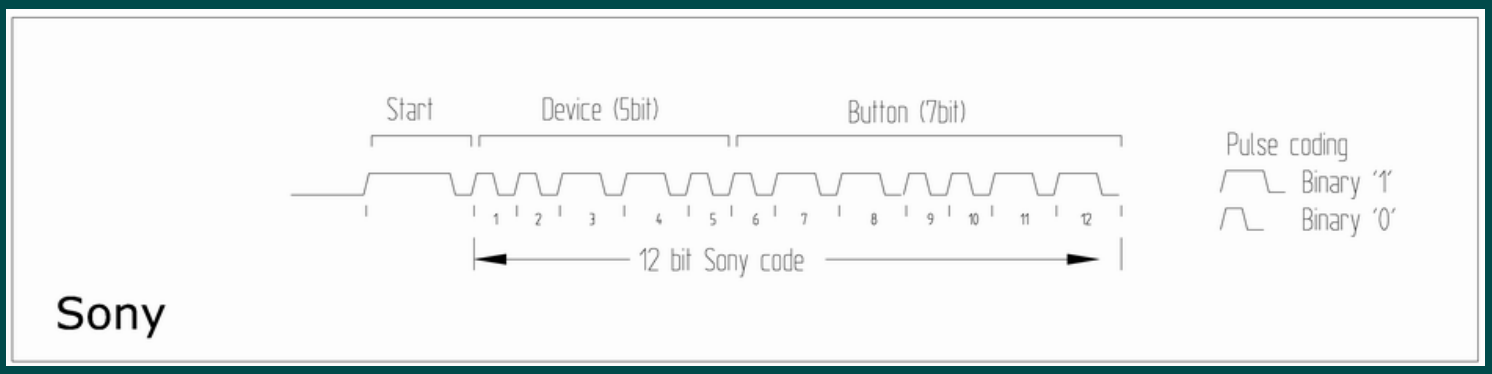

As it can be seen the RC5 code is transmitted with MSB first with a data payload of 11 bits. Actually the start bits can be binary 10 as well as 11 shown in the figure above. It is then called an extended RC5 code.

### Timing values :

The base clock for RC5 is 36kHz and a total of 64 counts is used per bit giving a bit time = 1.78 msec. The duration of the RC5 code is then 14 times this = 24.9 msec and the overall RC5 period is 64 bit periods = 113.8 msec.

## SIRCS - SONY

The Sony IR data is transmitted LSB first, probably since it gives no meaning to transmit the code from top and downwards. (bad joke :-). The format is pulse length modulation. There can be 12 or 15 databits of which the standard code seems to be the one af 12 bit in length, with a 5 bit device id and then a 7 bit button id. I don't know the rationale/meaning of the additional 15 bit code length, but the decoder accepts this as well as the 12 bit code. Contrary to the Phillips code the start sequence before a Sony sequence is not regarded as a part of the bit count.




### Timing values :

The sync in a Sony transmission is a 2.4 msec burst followed by 0.6 msec silence. The base clock for Sony is 40kHz and a total of 24 counts is used per bit element (with a bit consisting of 2 or 3 elements) giving a bit time = 1.2 msec or 1.8 msec respectively. The teoretical duration of the Sony code is minimum 3.0 + 12*1.2 = 17.4 msec and maximum 3.0 + 15*1.8 = 30 msec (*note) and the overall Sony period is allways fixed at approx 45 msec which equals ..... nothing ?

(*note) My personal highscore is approx. 26 msec (on a stop button with remote in TV mode) !

## REC-80 - Panasonic

The Panasonic REC-80 (or RECS-80 ?) uses pulse-space modulation. The code is 48 bits long not counting a single trailing end-of-transmission bit (logic "0") and is based on a 32kHz clock. Please find more elsewhere :-)

## SAMPLING - software

The sampling of the IR signals as done by the PIC processor is made very strict concerning the format. The drawback from this is that codes containing quirks from the devices of the real-world will be ignored. If this seems to be the problem then the consistency checking obviously should be relaxed. So far all bits are sampled twice in the case of RC5 to verify the biphase, and two or three times on the Sony/Panasonic to verify a high-low

or high-high-low sequence. After the code is completed it is verified that there is a few msecs with no signal before the code is accepted. This will obviuosly be a problem if a remote transmits codes with bitlengths different from the bitlengths mentioned above.

Another design issue is whether or not to require that two or more equal receptions of a code has been made before acknowledging it. Not surprisingly this will have effect on the latency of the system. Currently just a single code is needed, as it seems that the 'strict sampling' as described above effectively kills all junk transmissions. This number of matches needed might be subject to change.....


| | 4 | Schematic & PCB | Top |
|---|---|---|---|


# Schematic

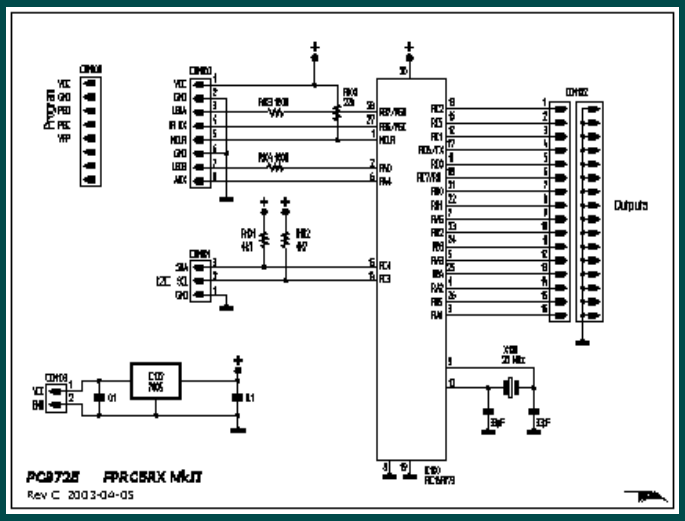
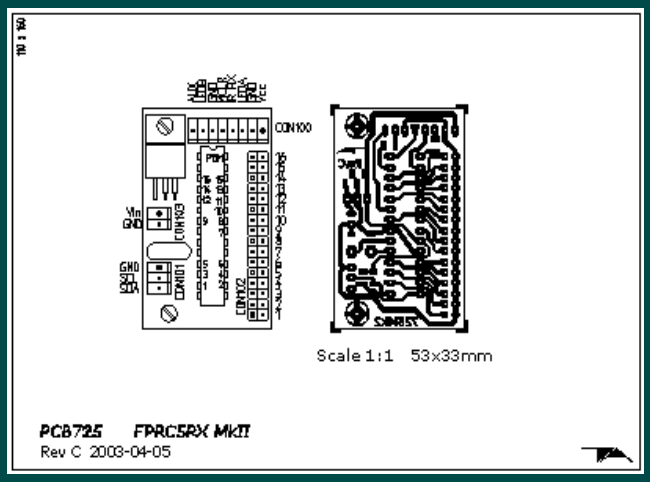



**Figure 4.1**. Schematic.

# PCB

See under downloads to get a readable version. All resistors and capacitors are 1206 size SMD (or 0805). If you think this pcb has a strange mix of component casings, then you're right !



# Tested IR receivers :

Ideally the IR receiver should have a center frequency of 36 kHz for RC5 and 40 kHz for Sony and 32 kHz for Panasonic.

**Vishay TFMM 5380** : Output active low, 38 kHz. Icc,typ 0.5 mA. Very good. (RC5 and Sony). This receiver were taken from a old VCR, and I actually don't expect it is possible to find it as it have been replaced by the TSOPxxx device line.

**Sharp IS1U60** : Output active low, 38 kHz. Icc,typ 2.8 mA. Either unusable or perhaps my sample is just slightly defective ? The device I have (bought from RS-components) is extremely sensitive to "light-bulb-light" and on top it thrashes a RC5 code real badly... (v0.6) Mikkel complained and told me he uses this type and it works fine for him :-). (V0.9) If this continues then I might have to surrender, now also John tells me that the IS1U60 rocks...

**Vishay 1138** : Output active low, 38 kHz. Icc,typ 1,2 mA. Very good. According to Vishay then the TSOP11/21 series actually would be a good compromise for covering all the 3 IR standards used here (noting the different carrier frequencies!).

If you have experiences with any of these or other IR receivers, good as well as bad, then you are most welcome to mail me.

# I/O connectors :

**CON100**. PIC programming, IR receiver and LEDs. The connector has a twofold purpose as it can be used for both normal use (IR receiver and LEDs) and for incircuit programming of the PIC. The intention is that the IR receiver chip and the LEDs are mounted off-pcb, or with a little ingenuity directly into the connector holes, letting the possibility for incircuit programming go.

**CON100 – LED(S).** The LEDA is the 'normally turned off' / 'flashing if active' output and this is the output to use if you want to connect a single standard LED. The LEDB output is just LEDA driven in antiphase in order to make it easy to use a bicolor LED instead. Since it is not obvious to everyone what is meant by this, you can have a look *here*. Consider to at least start with a ground referenced standard LED connected to LEDA while you are programming the PIC as this is easier to monitor.

**CON101**. Serial I2C output.

**CON102**. The 16 output pins. Drive sink/source in the order of 20 mA.

**CON103**. Power. Idle consumption is approx. 6 mA. At least 100 mA should be available when programming.



This table might be of use as a checklist at the first power on.

| ***Action*** | ***Response*** | ***No response !*** |
|---|---|---|
| *Apply power* | *Short led flash. (Normal mode)* | *Check everything (again!)* |
| *Let +5V touch LEDA (PIC pin 28/RB7)* | *LEDA flashes. (Programming mode)* | *Are you sure ?????* |
| *Let +5V touch a output pin X* | *Single LEDA flash (Mode 0)* | |
| *Press a key on remote* | *The signal led flashes.* | *Verify the output from the receiver with a oscilloscope* |
| *Release the key* | *The signal led turns off.* | |
| *Remove and re-apply power again* | *Short led flash. (Normal mode)* | |
| *Measure output pin X* | *0V* | |
| *Press the remote key again* | *5V on pin X and the signal LED flashes rapidly.* | |

***Table 5.1**. First run.*

You can use a DMM instead of an oscilloscope if you don't have one. You should see some voltage fluctuations whenever the receiver is active as opposed to a steady reading when there is no IR present.

# If it doesn't work :

From the email correspondances done so far then the majority of the fails people have had with this project boils down to remotes that turns out not to be transmitting one of the 'supported standards' after all. A few fails have been (unoriginal) remotes that transmitted with an okay standard but were found to have a timing that were slightly off. Then there are the expected ones such as using a 4 Mhz xtal instead of a 20 Mhz xtal and so on. A few problems were so exotic that nobody probably know what it was all about - the only comfort should be that so far I don't know of any that newer

got the decoder to work in the end....

| | 6 | Downloads | Top |
|---|---|---|---|

# Software for 16F873

The downloads are .hex files only, no source files. Please note that since the software can go into a 16F873 it fits nicely into a 16F876 as well, and it will also be happy in the 40 pin versions.

Software with build versions less than one are development. (Everything is in, and at least basic functionality has been verified.)

## Active low output IR receiver (TFM5380 / 1138 style)

20 MHz xtal - 32 assignments - 'Standard' mode table.

Build 0.34 - 28aug2003. Fixed bi-toggle mode on pins 10-12. Fixed radio-mode in general. Reworked channel delete part. *- Download*

| | 7 | Links | Top |
|---|---|---|---|

# Links

Here are some links. In a given timeframe they'l all transmorph to 404's but you might be lucky :-)

## Infrared protocols

Sony protocol : *http://www.ecn.purdue.edu/~laird/electronics/Sony/protocols/sircs.txt*

RC5 protocol : *http://users.pandora.be/nenya/electronics/rc5/index.htm* (5 stars!)

*http://www.hifi-remote.com/infrared/index.shtml*

RC5/RC6/REC-80 : *http://www.innotechsystems.com/primer1.pdf*

DigiPoints Vol3 Issue4 : *http://www.scte.org/professional/digi/Issue%203-04.pdf* (an IR overview)

Dataformat for IR protocols (Vishay) *http://www.vishay.com/docs/fmod_data_formats.pdf* (RC5 and NEC in detail. Table over recommeded TSOP devices for various protocols)

Nice intro to RC5 and Sony : *http://www.ustr.net/infrared/infrared1.shtml*

'SB-projects', basic theory plus various protocols : *http://www.xs4all.nl/~sbp/knowledge/ir/ir.htm*

Extensive list referenced by manufacturer : *http://www.geocities.com/SiliconValley/Lakes/3947/TABLE.HTML* and a postprocessing of this at *http://www.howell1964.freeserve.co.uk/remotes/irrc_systems.htm*

## Hardware

'Vishay.com', TFM 5380 : *www.vishay.com/document/82003/82003.pdf* (Browse the Vishay site yourself, it has a lot of interesting IR reciever info)

The SAA3049 : *http://wwww.ges.cz/sheet/s/saa3049.pdf*

A list of IR receivers called 'Serial port receivers' from the Lirc project : *http://www.lirc.org/*

## d.i.y. projects **(not focusing on the PC IR frontends)**

MP3 IR Decoder/Remote Control : *http://www.pjrc.com/tech/mp3/ircontroller/* (the project I mention at the top of this page)

Ruud van Gessel. *http://people.a2000.nl/rwvgesse/FixFrame.htm?pic_atmel.htm*

Remote standards info, and a nice java remote analyser ! : *http://cgl.bu.edu/GC/shammi/ir/*

Simple IR extender : *http://www.mitedu.freeserve.co.uk/Circuits/Interface/irext.htm*

Graphical IR analyser for a Gameboy color : *http://www.geocities.com/kkaarvik/infrared.html*

PC Remote Control : *http://www.pcremotecontrol.com/info.html* (also a learning decoder, but for a PC running windows)

Holgi's RC5 seite, RC-5 transmitters and receivers (in german) : *http://home.t-online.de/home/holger.klabunde/rc5send.htm*

IR analyser outputting an XML config file and a receiver for Sony to RS-232 (both based on PICs) :
*http://www.users.bigpond.com/pbhandary/index.html*

A diy remote decoder for a standard dc-motorized audio volume control : *http://stiftsbogtrykkeriet.dk/~mcs/Remote/index.html*

TinyIR - A commercial 6$ learning decoder in a 8pin PIC for d.i.y. projects : *http://www.tauntek.com/*

Sony decoder in a PIC12C508A with RS232 serial output. (Also sold ready for 6$) : *http://www.swampgas.com/robotics/irremote/*

Preamplifier with bottles and RC5 control : *http://members.home.nl/baltusg/*

Learning decoder in a 16F84 based lamp controller : *http://www.mnsi.net/~boucher/emporium.htm (My projects / Lamp controller)*

## Assorted Links.

The PIC web ring : *http://members.tripod.com/~mdileo/pmring.html* (a must see)

WWW.Epanorama IR link page : *http://www.epanorama.net/links/irremote.html* (not easy to miss!)

## On this site, somewhat related

Standard audio preamplifier controlled with a fprc5rx : *http://home1.stofanet.dk/hvaba/fprc5rx/preamp.html*

A remote controlled steppermotor for audio volume control (also see 'diy remote decoder' link above):
*http://home1.stofanet.dk/hvaba/steppervolume/stepper.html*

*test*