Configuring any IR remote which can be used to control a Desktop or Laptop using Serial or USB connections

DSP Project

Sonali Dubey	200801208
Yash Soni	200801207

Index

1. Brief Overview of the project

The aim of this project is to control a desktop or a Laptop with a simple remote, like the one we already use for our TV-set. The buttons in the remote can be configured to either start applications, swap between windows, flip pages during presentations, turning off the system and many more. Almost any application that uses shortcut keys and mouse can be controlled using this setup.

❖ **One question that would come up in anyone's mind is -- Why is there a need to make such device?**

Today's PCs are fast becoming home entertainment centres, with DVD players, music jukeboxes and even TV recording facilities.
This is all very well, but it's not much fun if you have to use a number of key presses and mouse clicks to launch the required programs and get them working. Using cabled mice and keyboards is hardly as easy as operating our domestic TVs either.

The answer to this simple question lies in the fact that we will try everything to make our life easier. As rightly said by someone, "I don't think necessity is the mother of invention- invention arises directly from idleness, possibly also from laziness. To save oneself trouble"

❖ **Many other technical advantages are:**

- Can control computer from a distance.
- No need to attach or make any additional changes to the hardware because it connects to the serial port (if provided) or USB port in a PC.
- No external power source needed.
- Works with any type of used/un-used TV remote (Just have to configure the remote)
- Some of the new laptops have same sort of remote mechanism supplied, but its use is limited:
    - Firstly not every laptop/desktop has this facility.
    - You can't configure the remote as per your needs.
    - Can't use that remote with any other device.
    - Should have an infra-red sensor built in the System.
- Most important, constructing this whole mechanism is cheap (costs around 300/-Rs.)

Things Needed

1. An IR remote – almost any old remote will do.
2. An infrared receiver – have to build this
3. WinLIRC –
    - Windows equivalent of LIRC, the Linux Infrared Remote Control program.
      LIRC is an open source package that allows you to receive and send infrared signals with your Linux computer system. Specifically, it is a kernel module that must be compiled by hand in order for it to work with an existing Linux kernel.
4. IR Assistant –
    - It is a shareware that allows you to emulate mouse actions, launch applications, execute macros, and more.

Brief Working of the whole setup:

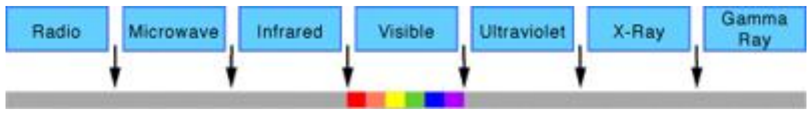
1. We press a button on the remote control.
2. The IR receiver receives the signal sent by the remote.
3. WinLIRC is pre-configured (by the user) to recognize this RAW signal.
4. The name of the signal is sent to IR assistant by WinLIRC.
5. IR Assistant associates the signal with a command we choose (for example, launching a file)
6. Finally IR Assistant sends the command to the application you chose on your PC.

Future perspectives:

- Making the IR receiver assembly such that it can recognize signals from 2 different remotes and perform the respective actions.
- It can be used to play multiplayer educational games, 2 persons can work on their stuff individually etc.

2. Working of an Infrared Remote

The dominant remote-control technology in home-theatre applications is infrared (IR). Infrared light is also known as plain-old "heat." The basic premise at work in an IR remote control is the use of light to carry signals between a remote control and the device it's directing. Infrared light is in the invisible portion of the electromagnetic spectrum.



An IR remote control (the **transmitter**) sends out pulses of infrared light that represent specific binary codes. These binary codes correspond to commands, such as Power On/Off and Volume Up. The IR **receiver** in the TV, stereo or other device decodes the pulses of light into the binary data (ones and zeroes) that the device's microprocessor can understand. The microprocessor then carries out the corresponding command.

To avoid interference caused by other sources of infrared light, the infrared receiver on a TV only responds to a particular wavelength of infrared light, usually 980 nanometers.

2.1 Process

Pushing a button on a remote control sets in motion a series of events that causes the controlled device to carry out a command. The process works something like this:

1. When we push the "volume up" **button** on the remote control, it causes it to touch the **contact** beneath it and complete the "volume up" circuit on the circuit board. The integrated circuit detects this.
2. The **integrated circuit** sends the binary "volume up" command to the LED at the front of the remote.
3. The **LED** sends out a series of light pulses that corresponds to the binary "volume up" command.

3. RC-5 Protocol (Protocol used for communicating between Tx and Rx)

The **RC-5** protocol was developed by Philips in the late 1980s as a semi-proprietary consumer IR (infrared) remote control communication protocol for consumer electronics.

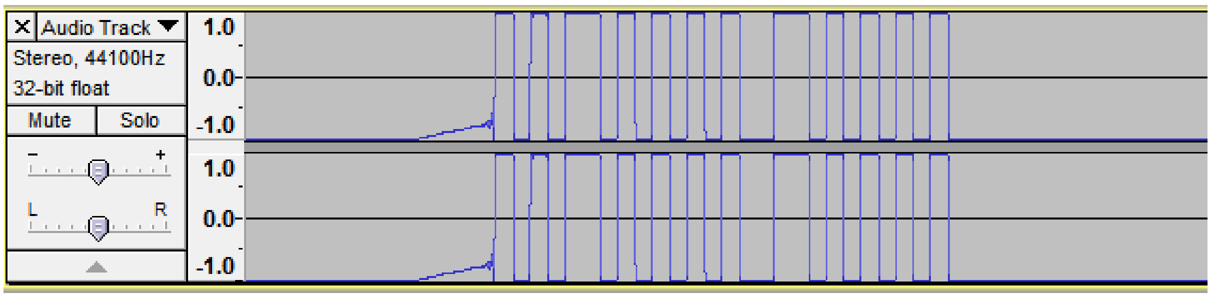
Protocol Details

The basics of the protocol are well known. The handset contains a keypad and a transmitter integrated circuit (IC) driving an IR LED. The command data is a Manchester coded bit-stream modulating a 36 kHz carrier. The IR signal from the transmitter is detected by a specialized IC with an integral photo-diode, and is amplified, filtered, and demodulated so that the receiving device can act upon the received command. RC-5 only provides a one-way link, with information traveling from the handset to the receiving unit.

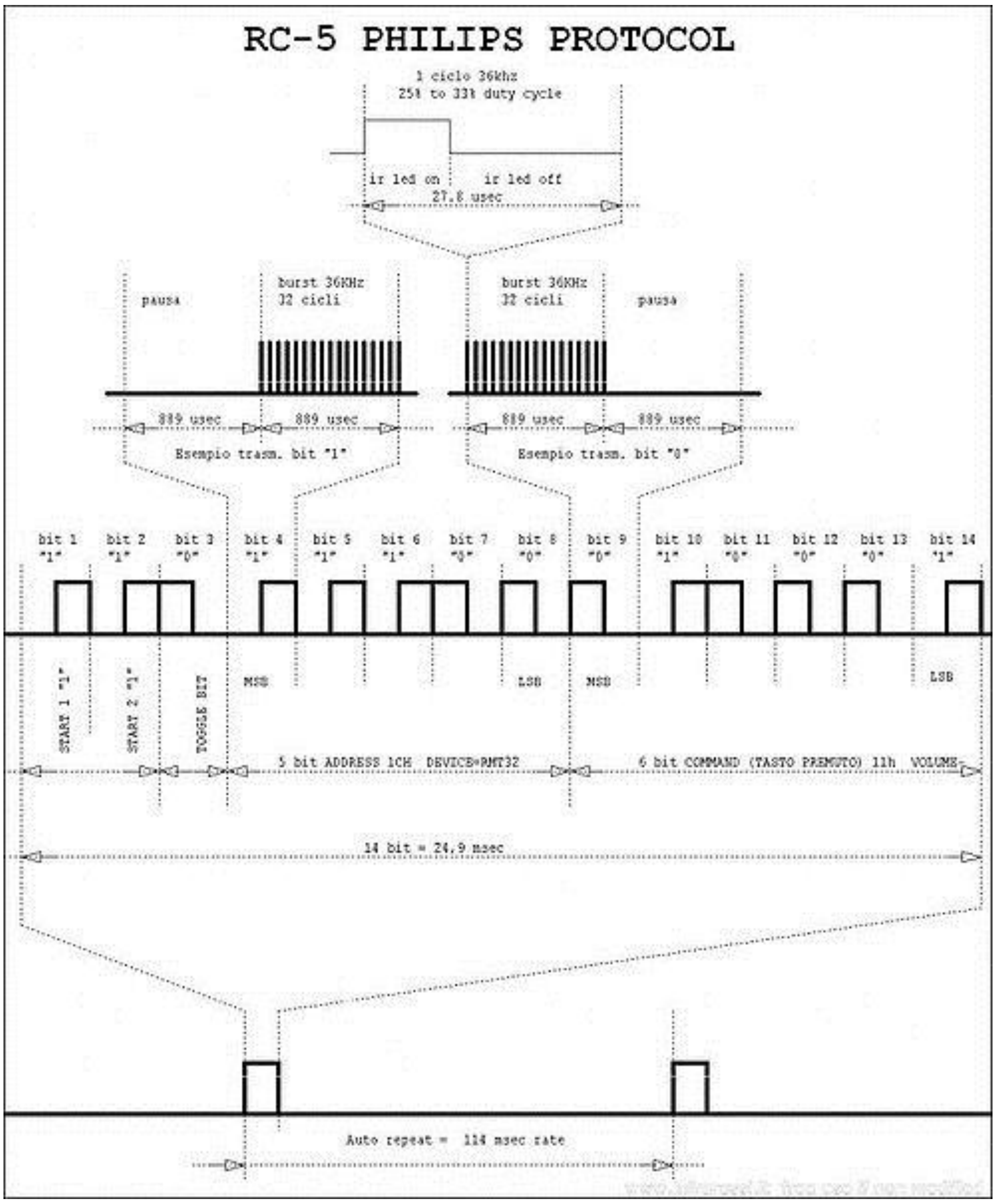
The command comprises 14 bits:

- A start bit, which is always logic 1 and allows the receiving IC to set the proper gain.
- A field bit, which denotes whether the command sent is in the lower field (logic 1 = 0 to 63 decimal) or the upper field (logic 0 = 64 to 127 decimal). The field bit was added later by Philips when it was realized that 64 commands per device were insufficient. Previously, the field bit was combined with the start bit. Many devices still use this original system.
- A control bit, which toggles with each button press. This allows the receiving device to distinguish between two successive button presses (such as "1", "1" for "11") as opposed to the user simply holding down the button and the repeating commands being interrupted by a person walking by, for example.
- A five-bit system address that selects one of 32 possible systems.
- A six-bit command that (in conjunction with the field bit) represents one of the 128 possible RC-5 commands.

The 36 kHz carrier frequency was chosen to render the system immune to interference from TV scan lines. Since the repetition of the 36 kHz carrier is 27.778 µs and the duty factor is 25%, the carrier pulse duration is 6.944 µs. Since the high half of each symbol (bit) of the RC-5 code word contains 32 carrier pulses, the symbol period is 64 x 27.778 µs = 1.778 ms, and the 14 symbols (bits) of a complete RC-5 code word takes 24.889 ms to transmit. The code word is repeated every 113.778 ms (4096 / 36 kHz) as long as a key remains pressed.



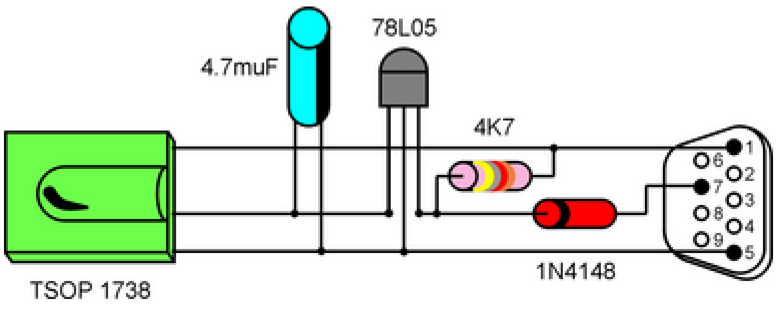
Manchester code for ‘p+’ button as seen in Audacity.
Note: the hex code transmitted by the remote for the same is 0x1020


RC-5 PHILIPS PROTOCOL
1 ciclo 36khz
25% to 33% duty cycle
ir led on
ir led off
27.8 usec
pausa
burst 36KHz
32 cicli
burst 36KHz
32 cicli
pausa
889 usec
889 usec
889 usec
889 usec
Esempio trasm. bit "1"
Esempio trasm. bit "0"
bit 1 "1"
bit 2 "1"
bit 3 "0"
bit 4 "1"
bit 5 "1"
bit 6 "1"
bit 7 "0"
bit 8 "0"
bit 9 "0"
bit 10 "1"
bit 11 "0"
bit 12 "0"
bit 13 "0"
bit 14 "1"
START 1 "1"
START 2 "1"
TOGGLE BIT
MSB
LSB
MSB
LSB
5 bit ADDRESS 1CH DEVICE=RMT32
6 bit COMMAND (TASTO PREMUTO) 11h VOLUME-
14 bit = 24.9 msec
Auto repeat = 114 msec rate

4. Remote Assembly

4.1 Requirements:

1. Female DB9 connector
2. TSOP 1738 IR receiver
3. 78L05 voltage regulator
4. 4700 Ω resistor
5. 1n4148 diode
6. 4.7 µF capacitor
7. T.V remote
8. Softwares
   i. WinLIRC
   ii. IR assistant



Schematics of the IR receiver

4.2 Serial Port

Back and front of Female DB15 connector

It is an asynchronous port on the computer used to connect a serial device to the computer and capable of transmitting one bit at a time. **Serial ports** are typically identified on computers as COM (communications) ports.

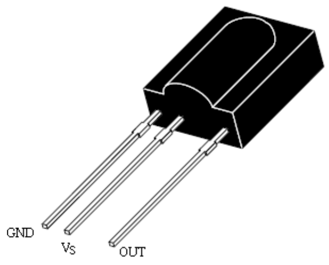
Below is a listing of each of the pins located on the DB9 connector and what each of these pins is for.

PIN	PURPOSE	SIGNAL NAME
Pin 1	Data Carrier Detect	DCD
Pin 2	Received Data	RxData
Pin 3	Transmitted Data	TxData
Pin 4	Data Terminal Ready	DTR
Pin 5	Signal Ground	Gnd
Pin 6	Data Set Ready	DSR
Pin 7	Request To Send	RTS
Pin 8	Clear To Send	CTS
Pin 9	Ring Indicator	RI

For this reciever, we will be using:

Pin 7 RTS- line which gives power to the voltage regulator which fixes it to 5 stable volts.
Pin 5 Gnd- ground.
Pin 1 DCD- The data output of the IR receiver is connected to the DCD line of the serial port together with a pull-up resistor coming from the power line.

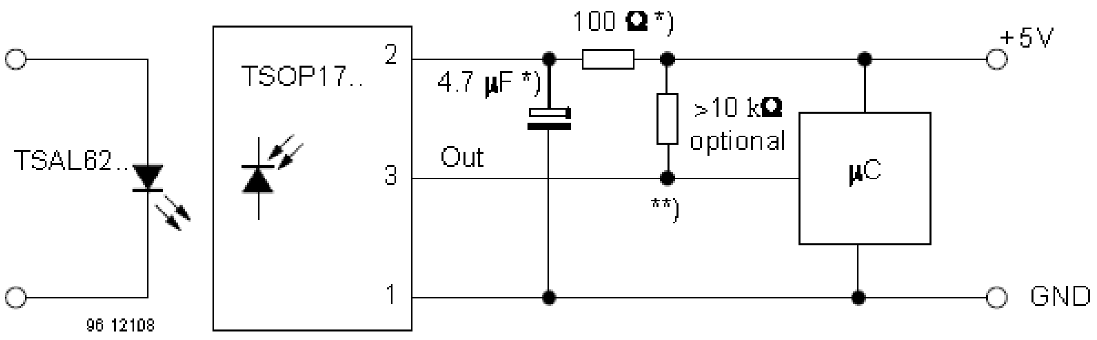
4.3 TSOP 1738



The TSOP17. – series are miniaturized receivers for infrared remote control systems. PIN diode and preamplifier are assembled on lead frame, the epoxy package is designed as IR filter. The demodulated output signal can directly be decoded by a microprocessor. TSOP17xx is the standard IR remote control receiver series, supporting all major transmission codes.

Features
_ Photo detector and preamplifier in one package
_ Internal filter for PCM frequency
_ Improved shielding against electrical field disturbance
_ TTL and CMOS compatibility
_ Output active low
_ Low power consumption
_ High immunity against ambient light
_ Continuous data transmission possible (up to 2400 bps)
_ Suitable burst length .10 cycles/burst

Application circuit



*) recommended to suppress power supply disturbances
**) The output voltage should not be hold continuously at a voltage below 3.3V by the external circuit.

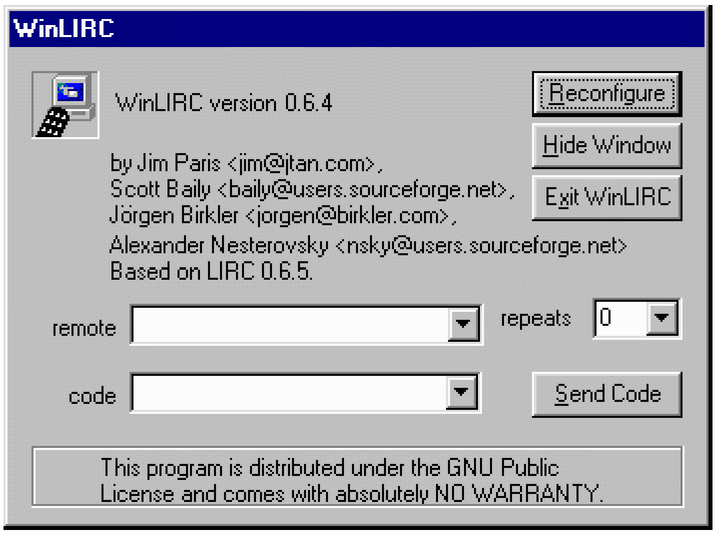
4.4 WinLIRC

It is a Windows equivalent of LIRC, the Linux Infrared Remote Control program.

LIRC is an open source package that allows you to receive and send infrared signals with your Linux computer system. Specifically, it is a kernel module that must be compiled by hand in order for it to work with an existing Linux kernel.

WinLIRC is supplied under the GNU Public License. Source code is included in the installation package.

The Main WinLIRC Window



WinLIRC

WinLIRC version 0.6.4

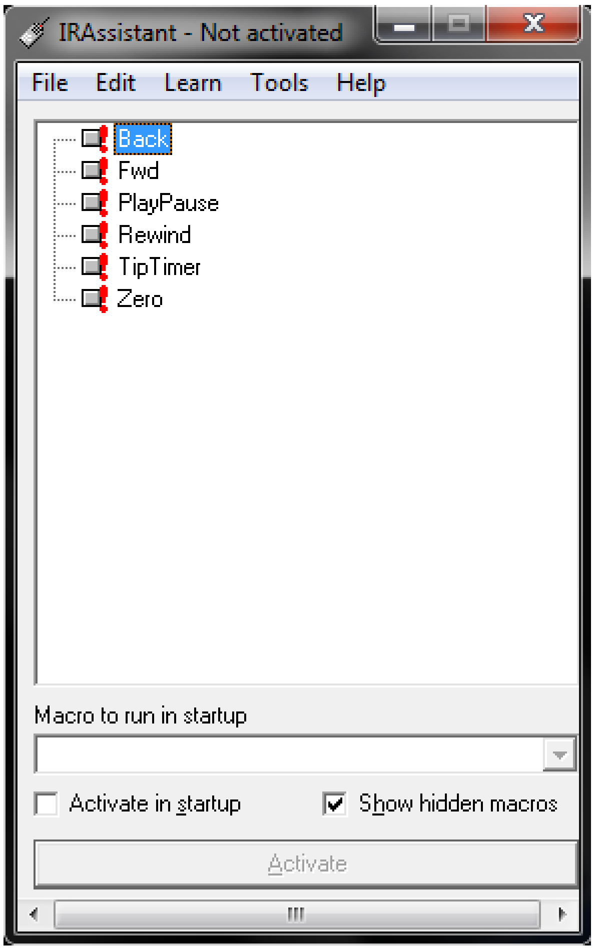
by Jim Paris <jim@jtan.com>,
Scott Baily <baily@users.sourceforge.net>,
Jörgen Birkler <jorgen@birkler.com>,
Alexander Nesterovsky <nsky@users.sourceforge.net>
Based on LIRC 0.6.5.

remote:
repeats: 0
code:

This program is distributed under the GNU Public License and comes with absolutely NO WARRANTY.

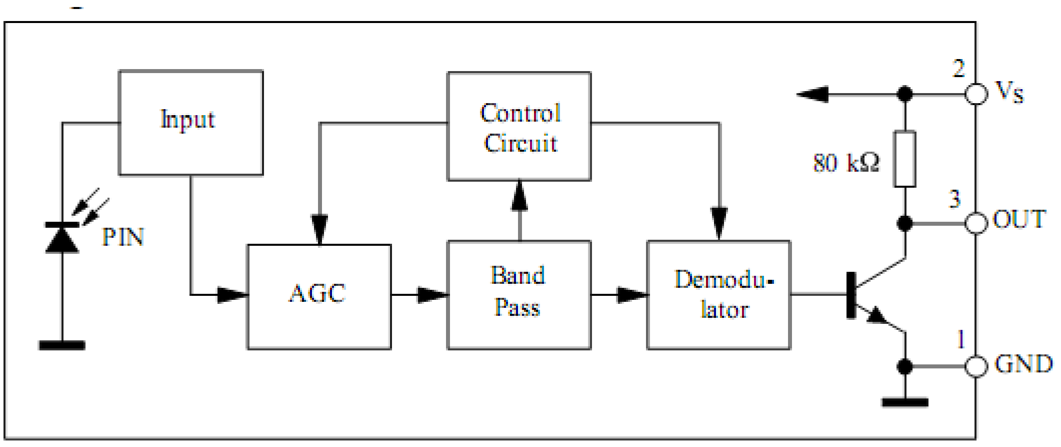
4.5 IR Assistant

It is a shareware that allows you to emulate mouse actions, launch applications, execute macros, and more.
It is free for private and educational use.



5. TSOP simulation on Matlab

- A TSOP is a device which collects (receives) the Modulated wave sent out by the infrared transmitter in the Remote.
- The signal then passes through an **AGC circuit** which amplifies or de-amplifies it according to the signal strength.
- The signal is then passed through a **Bandpass filter** which removes the noise signals and allows only a band of frequency to pass through it.
- The output signal from the Bandpass filter is one which only has frequency components near 36kHz.
- Then the signal is demodulated using a **demodulator** and the final Manchester coded signal is output from the pin 3 of the TSOP



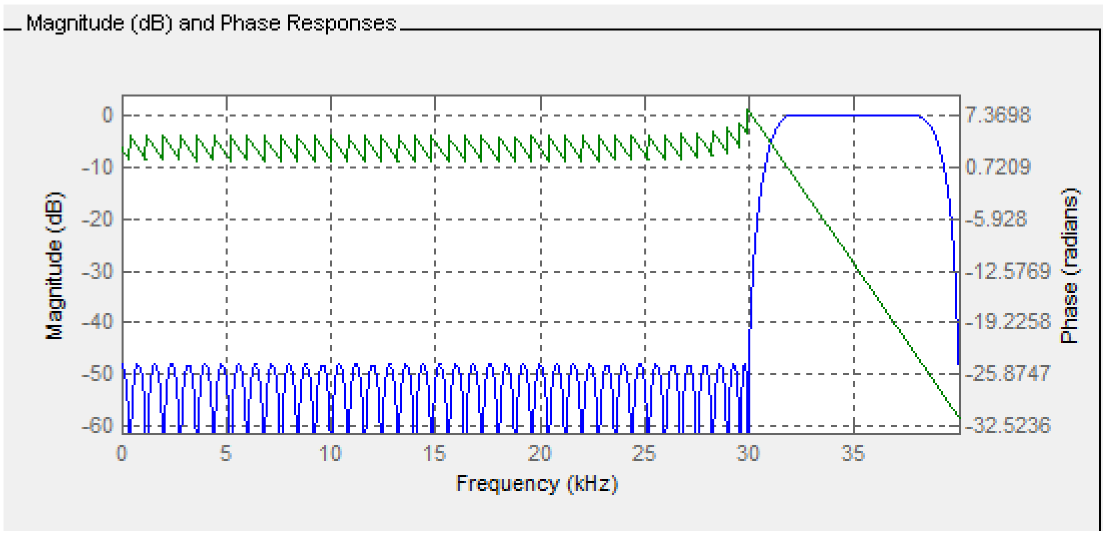
Block Diagram of an TSOP

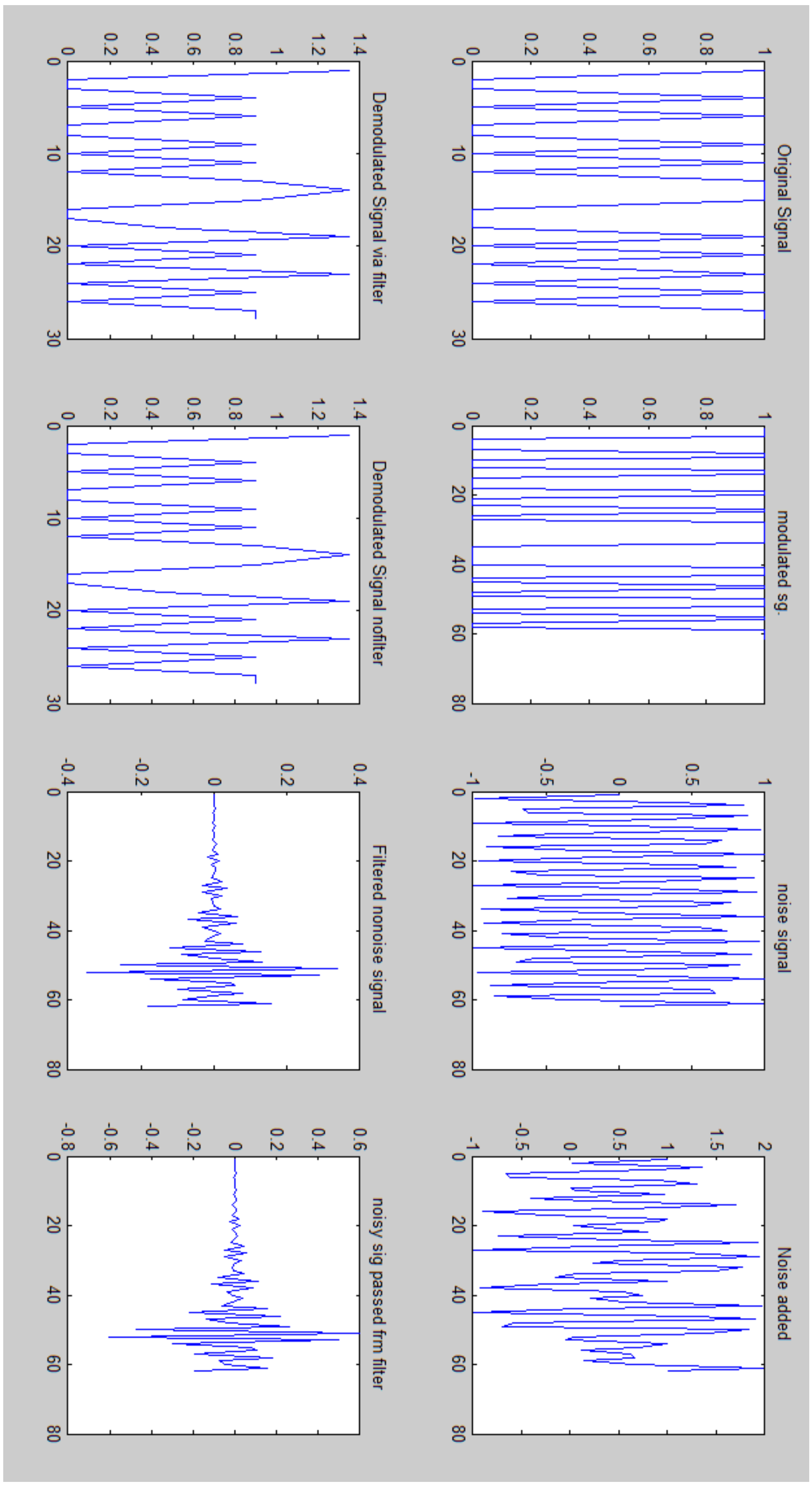
```
x=[1 0 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 1 0 1 1];  %Input Signal

modsig=modulate(x,36000,80000,'pwm');                 % PWM modulation of the input signal

t=(0:1/61:1)
noise=sin(2*pi*12000*t);                              % Noise signal generation

noiseadded=modsig+noise;                              % Adding the noise signal

poles(101)=1;
for j=1:1:101
   poles(j)=1;
end

nonoise=filter(num,poles,noiseadded);                % Filtering the Noisy Signal
check=filter(num,poles,modsig);                      % Filtering the simple modulated signal

z1=demod(modsig,36000,80000,'pwm');                  % Demod. of the Simple modulated Signal
z1=simple output


zz=z1;
inputlength=length(zz);
for i=1:1:inputlength
   if (zz(i)>0)
         zz(i)=1;
   end
end
zzz=nonoise
inputlength1=length(nonoise)
z2=z1

for i=1:1:inputlength1
   if (zzz(i)>0)
         zzz(i)=1;
   elseif (zzz(i)<0)
      zzz(i)=0;
   end
end
z=demod(zzz,36000,80000,'pwm');
dm=z;


for m=1:1:length(z)-1
   dm(m)=z(m+1);
end
```

```
subplot(2,4,1), plot(x); title('Original Signal');
subplot(2,4,2), plot(modsig);title('modulated sg.');
subplot(2,4,3), plot(noise);title('noise signal');
subplot(2,4,4),plot(noiseadded);title('Noise added');
subplot(2,4,5), plot(z2); title('Demodulated Signal via filter');
subplot(2,4,6), plot(z1); title('Demodulated Signal nofilter');
subplot(2,4,7), plot(check); title('Filtered nonoise signal')
subplot(2,4,8), plot(nonoise);title('noisy sig passed frm filter');
```

BandPass filter Filter Design

```
function Hd = bandpass1
Fs = 80;  % Sampling Frequency

Fstop1 = 33;           % First Stopband Frequency
Fpass1 = 34;           % First Passband Frequency
Fpass2 = 38;           % Second Passband Frequency
Fstop2 = 39;           % Second Stopband Frequency
Dstop1 = 0.001;          % First Stopband Attenuation
Dpass  = 0.057501127785;  % Passband Ripple
Dstop2 = 0.0001;          % Second Stopband Attenuation
dens   = 20;             % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fstop1 Fpass1 Fpass2 Fstop2]/(Fs/2), [0 1 ...
                    0], [Dstop1 Dpass Dstop2]);

% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);

% [EOF]
```



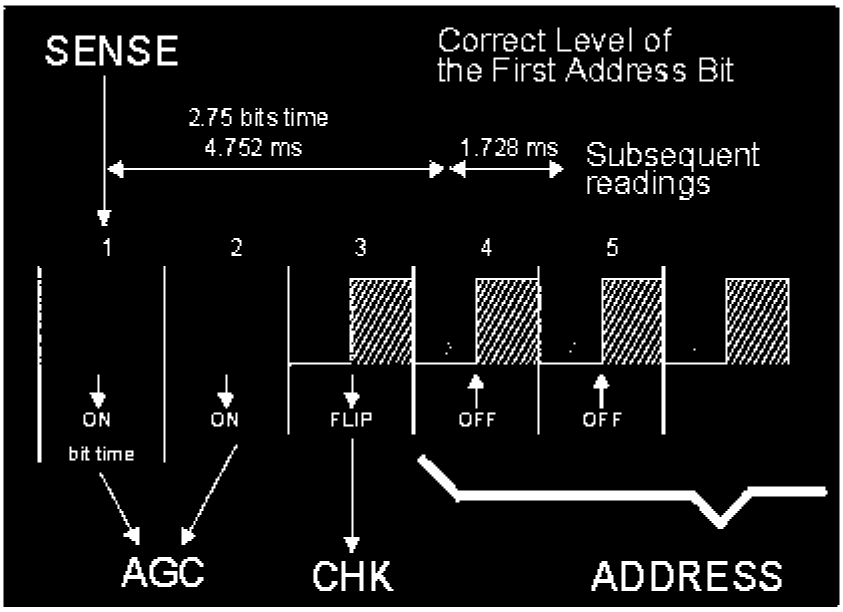

Original Signal
modulated sg.
noise signal
Noise added
Demodulated Signal via filter
Demodulated Signal nofilter
Filtered nonoise signal
noisy sig passed frm filter

6. Decoding RC5 protocol using Matlab

```
a=wavread('menunew.wav');                    %reading the saved sample data
% a=wavrecord(3*44100,44100);                %reading 'real time' input
wavsize=size(a);

i=1;                                         %detecting the first '1'
while((a(i,1)<.999)&&(i<wavsize(1)))
   i=i+1;
end

delimit=i;
j=i+16;                                      %taking the middle samples
vj=j;
k=1;

for k=1:1:28
   b(k)=0;
end

 k=2;
 b(1)=-1;
for j=j:38:wavsize
   b(k)=a(j,1);
   k=k+1;
end

c=b;
for i=1:1:28
if(c(i)>0)
   c(i)=1;

elseif(c(i)<0)
   c(i)=0;
end
end

for l=1:1:12
     d(l)=c(l+14);
end
d                                            %Final binary output
```

%Binary to manchester coding

```
inputData=d;
nBits = length(inputData);                          % length of inputData
decodedData = ones(1,nBits/2);

if mod(nBits,2)~=0                                  % check if array is even
   error('Length of array must be even')
end

for i = nBits:-2:2                     % count from max. size downwards with steps of 2
   if inputData(i) ~= inputData(i-1) % if bits are unequal
      decodedData(i/2) = inputData(i); % the first bit is the binary value
   else
      decodedData = [];                %if bits are equal, it's not manchester code
      break %exit for loop
   end
end
decodedData                                         %final Manchester coded data
```

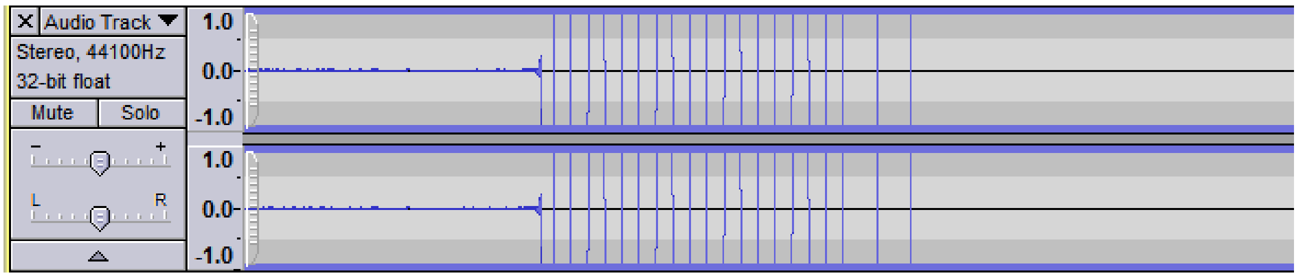


7. Appendix

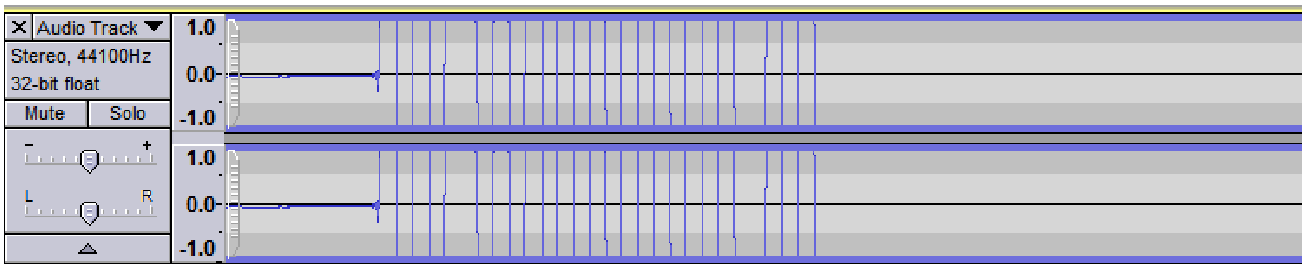
5.1 Configuration File for the remote

```
begin remote

 name  ..\config.cfg
 bits        13
 flags RC5|CONST_LENGTH        // RC-5 protocol is used
 eps         30                //Margin of error (use high values if errors are more (default=25))
 aeps        100

 gap       107492              //Gap between two consecutive signals sent (in µs)
 toggle_bit_mask 0x800

   begin codes
     menu           0x103B
     p+             0x1020
     p-             0x1021
     v+             0x1010
     v-             0x1011
     power          0x100C
     mute           0x100D
     1              0x1001
     2              0x1002
     3              0x1003
     4              0x1004
     5              0x1005
     6              0x1006
     7              0x1007
     8              0x1008
     9              0x1009
     -              0x100A
     0              0x1000
     alt            0x1022
     pp             0x100E
     av             0x1038
     timer          0x1026
     ft-            0x102C
     ft+            0x102B
   end codes
end remote
```

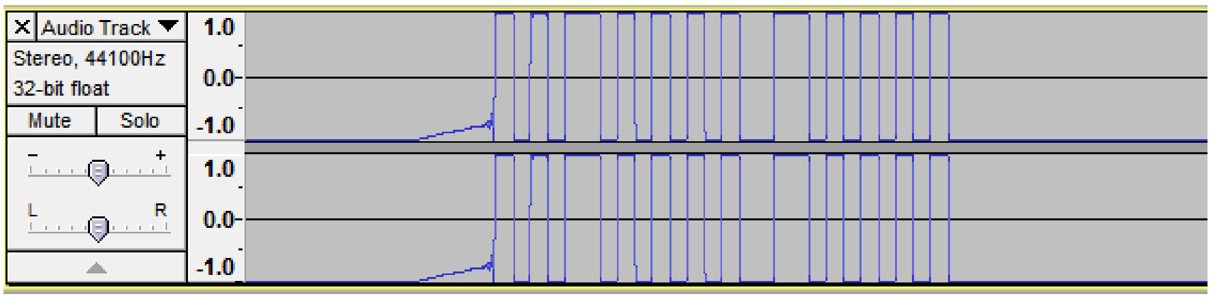
5.2 Sample Manchester coded signals sent by Transmitter:



Code sent for ‘2’ button

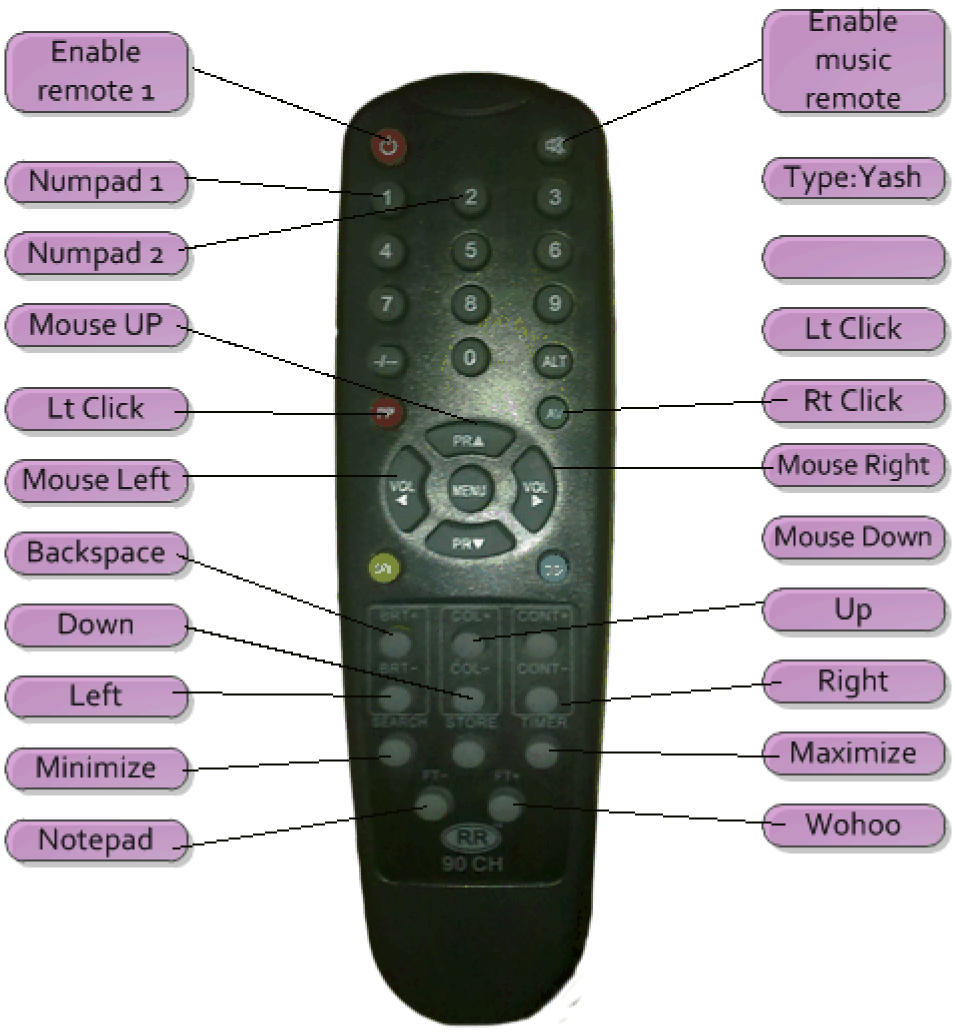


Code sent for ‘3’ button

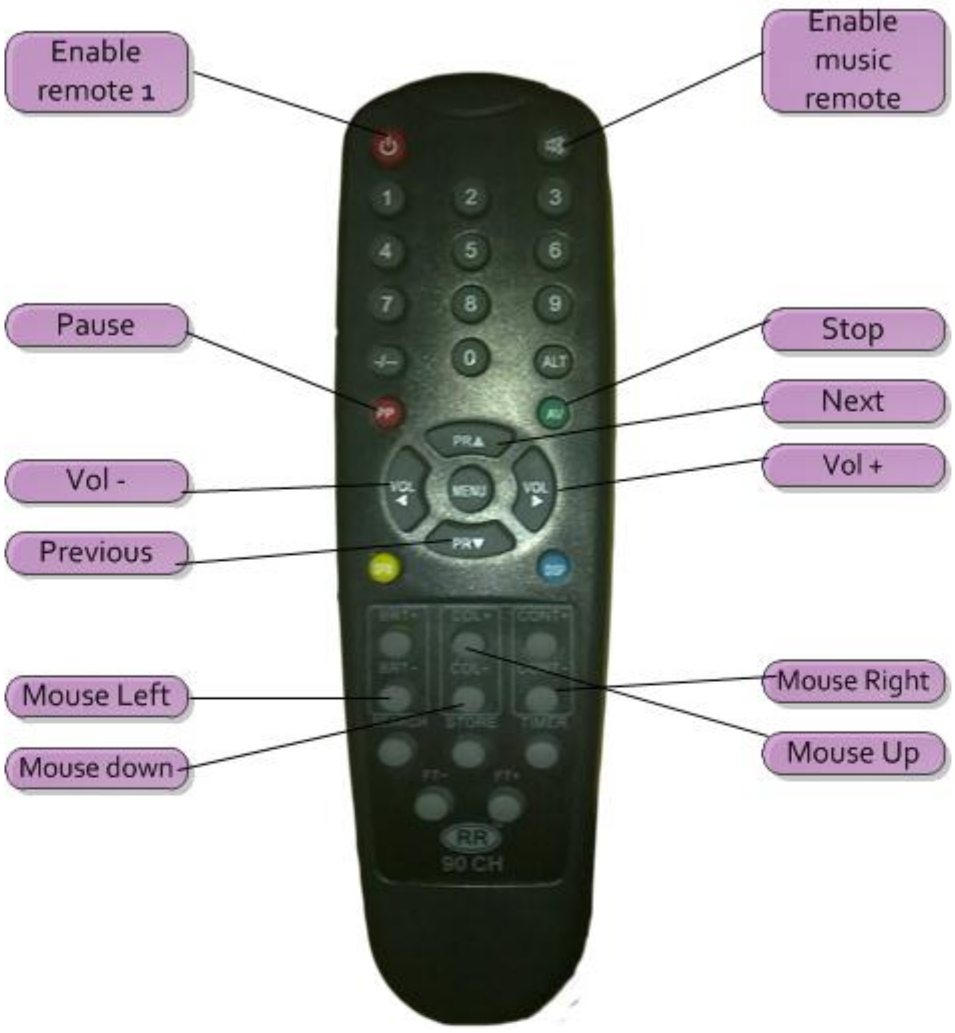


Code sent for ‘menu’ button

Basic Remote



Music Remote


Enable remote 1
Enable music remote
Pause
Stop
Next
Vol -
Vol +
Previous
Mouse Left
Mouse Right
Mouse down
Mouse Up