# Replacement For IR Remote

Paul Byrne 22nd Feb 2024

## Table of Contents

# 1 Background

I have a sound bar with an infra-red remote control. However the remote is very small and often gets lost or misplaced. I have set myself a project of building a remote to use as a back-up to the one supplied with the soundbar.

# 2 Requirements

The initial requirement is to use as much as possible components I have at hand, and only purchase what is absolutely necessary. Also to use the minimum number of components.

I only intend to use the most commonly used commands and have a range of about 3 metres (10ft). The commands are:

- ON/OFF
- Mute
- Volume Up
- Volume Down
- Sound Flat
- Sound Movie
- Sound Music

As there are only seven commands to be implemented a keyboard matrix is not needed, so direct connection to I/O ports will be used.

There are 9 I/O ports required, 2 output and 7 input. The output ports are required for driving an IR diode and LED indicator. The 7 inputs are needed for detecting the button presses for the commands.

# 3 Analysis to Determine Commands and Protocol Used

## Hardware and Software used

The commands and protocol used by the original were determined using an Arduino UNO connected to an IR receiver module(Figure1).

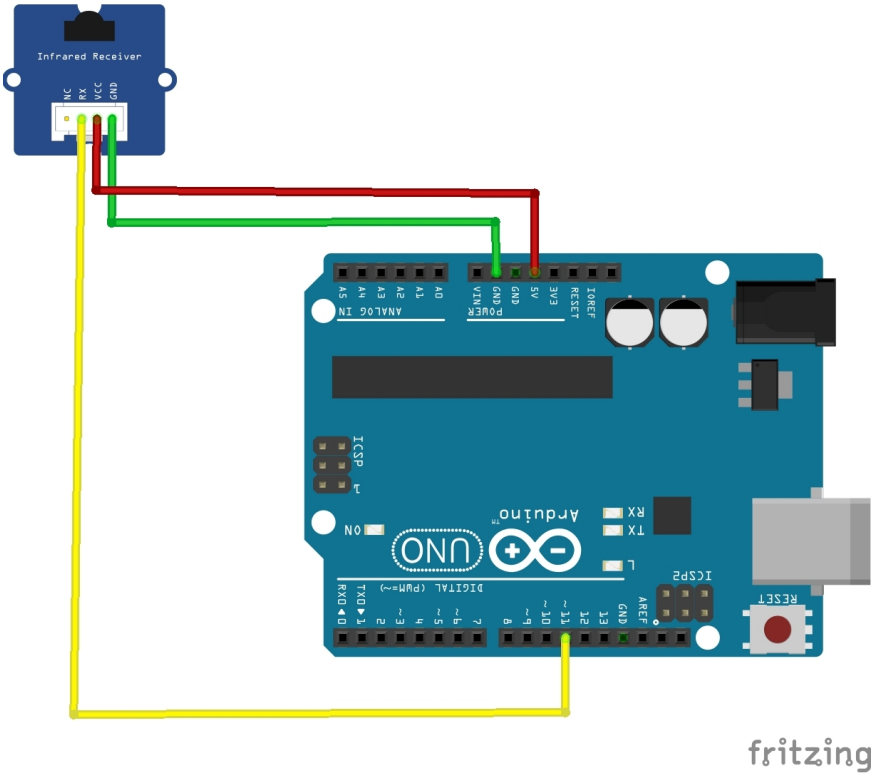



*Figure 1: IR Receiver*

The receiver module used was a TSOP1738, an alternative S-1500 could also be used.

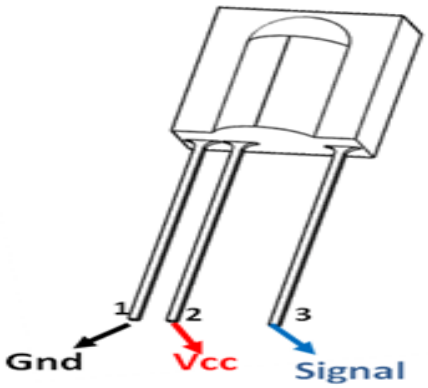
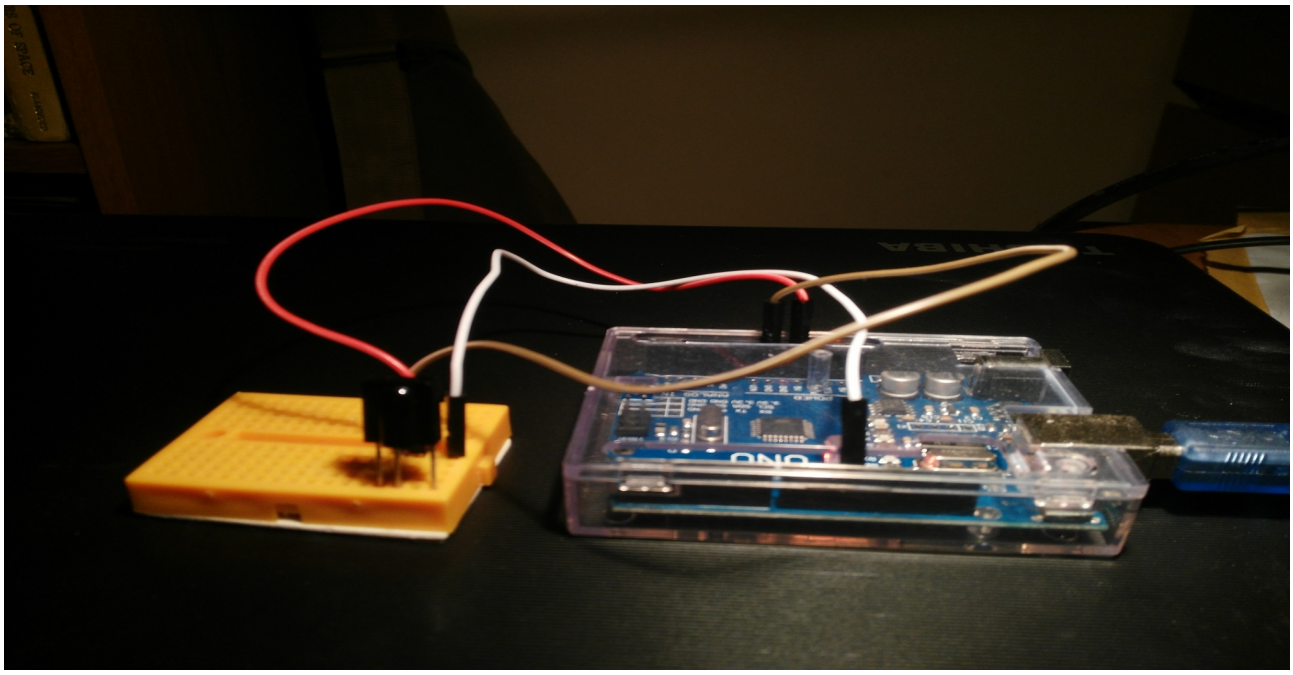



*Figure 2: TSOP1738 pinout*



The Arduino code used the IRremote library from Armin Joachimsmeyer. This can determine the protocol used, and the command sent. The Arduino sketch used was a modified version from one of the examples provided with the IRremote library ( in sketch folder ). The results are displayed the the Arduino serial monitor.

## Results

All the required commands were sent from the remote and detected using the Arduino circuit figure 1. The results from the serial monitor are listed in figure 3.

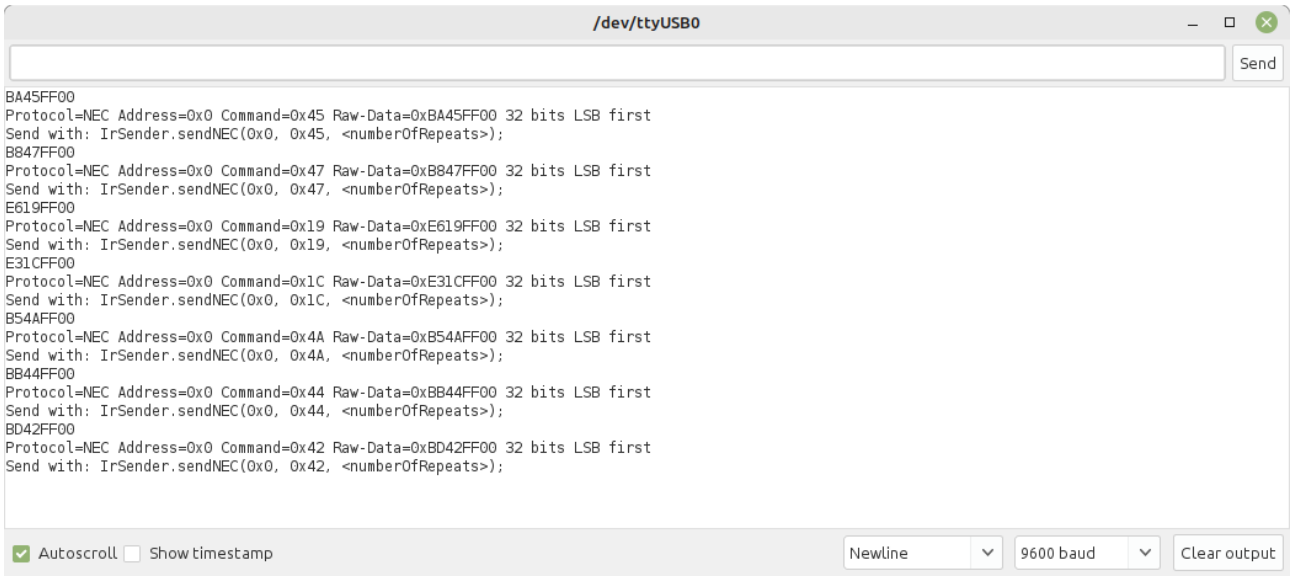
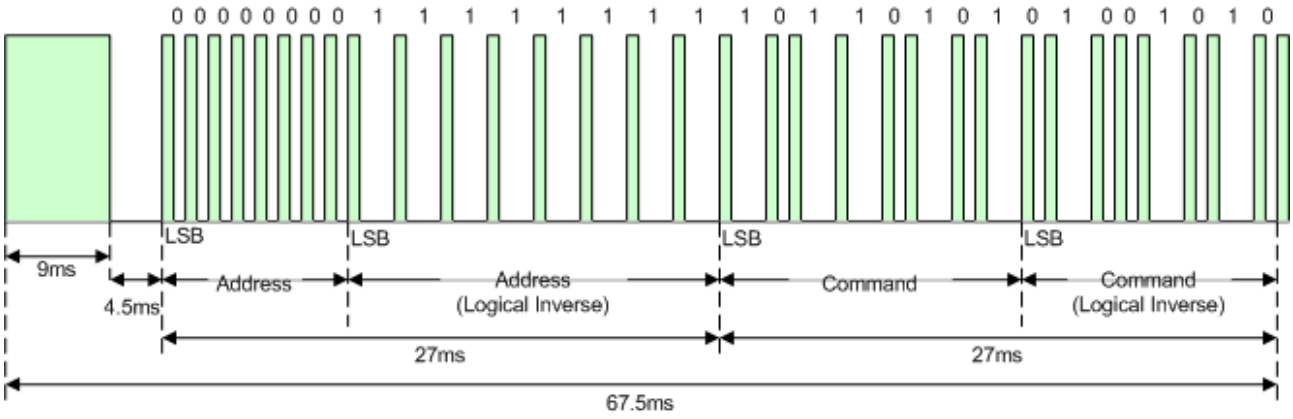



*Figure 3: IR Codes*

The results show that the protocol used was the NEC, and that the address used was 0x0.

# 4 NEC Infrared Transmission Protocol

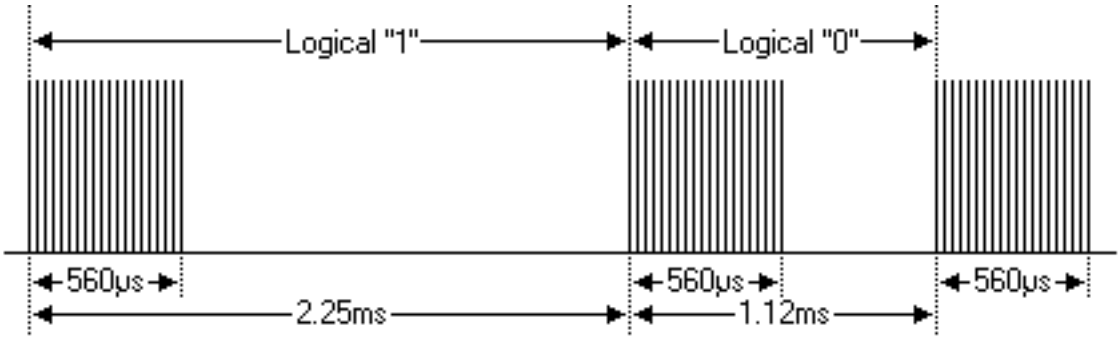
The NEC protocol is based on a pulsed carrier signal with a frequency of 38.222kHz ( nominally 38kHz), carrier **ON** defines a mark, carrier **OFF** space.



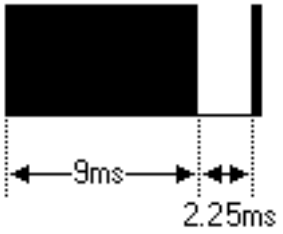

When a remote command is sent the message transmitted is as follows;

- a 9ms pulse burst of carrier signal
- a 4.ms space
- the 8-bit address of control device
- the 8-bit complement of address
- the 8-bit command
- the 8-bit complement of command
- a final 562.5µS pulse burst to signal end of message

The 8-bit data is sent using the following modulation of the carrier;



Normally if the key of the remote control remains pressed the message itself if not repeated instead a repeat code is sent;




This is comprised of a 9mS burst followed by a 2.25mS space and is terminated by a 562.5µS burst.

There is a second NEC format used to increase the number of potential addresses available. This is implemented by replacing the address complement with a 8 bit value representing the hi-byte of a 16bit address, however in this instance only the original format describe above is required.

# 5 Design of IR Remote

## Hardware used

From the microcontrollers if have available I have selected the PIC16F690 which has sufficient I/O ports. The input ports can be set to use internal pull-up resistors and interrupt-on-change. The PIC16F690 can be set to a low power sleep mode which will wake up on an interrupt. This microcontroller also has the advantage of an internal oscillator.

I have purchased a standard 940nm infrared diode as the transmitter.

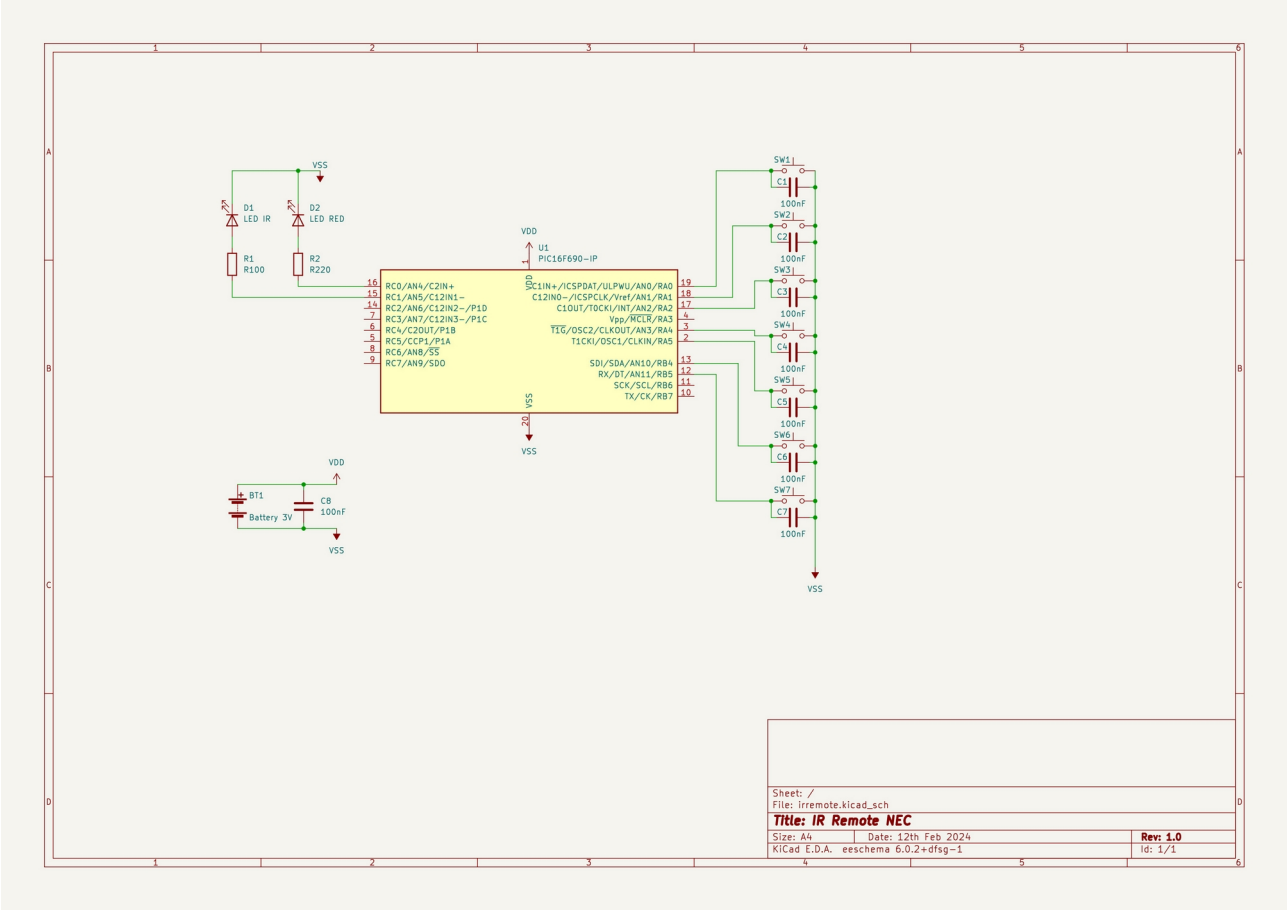



*Figure 4: Circuit of IR Remote*

# Firmware

## Tools

I have opted for efficiency to write the code in assembler using MPASM using MPLAB IDE 8.92 running in windows 10. The latest MPLAB X assembler is not as easy to use for 8-bit micros, MPASM is more straightforward. A PICKIT2 was used in conjunction with a low pin count demo board for development and programming (figure 5).

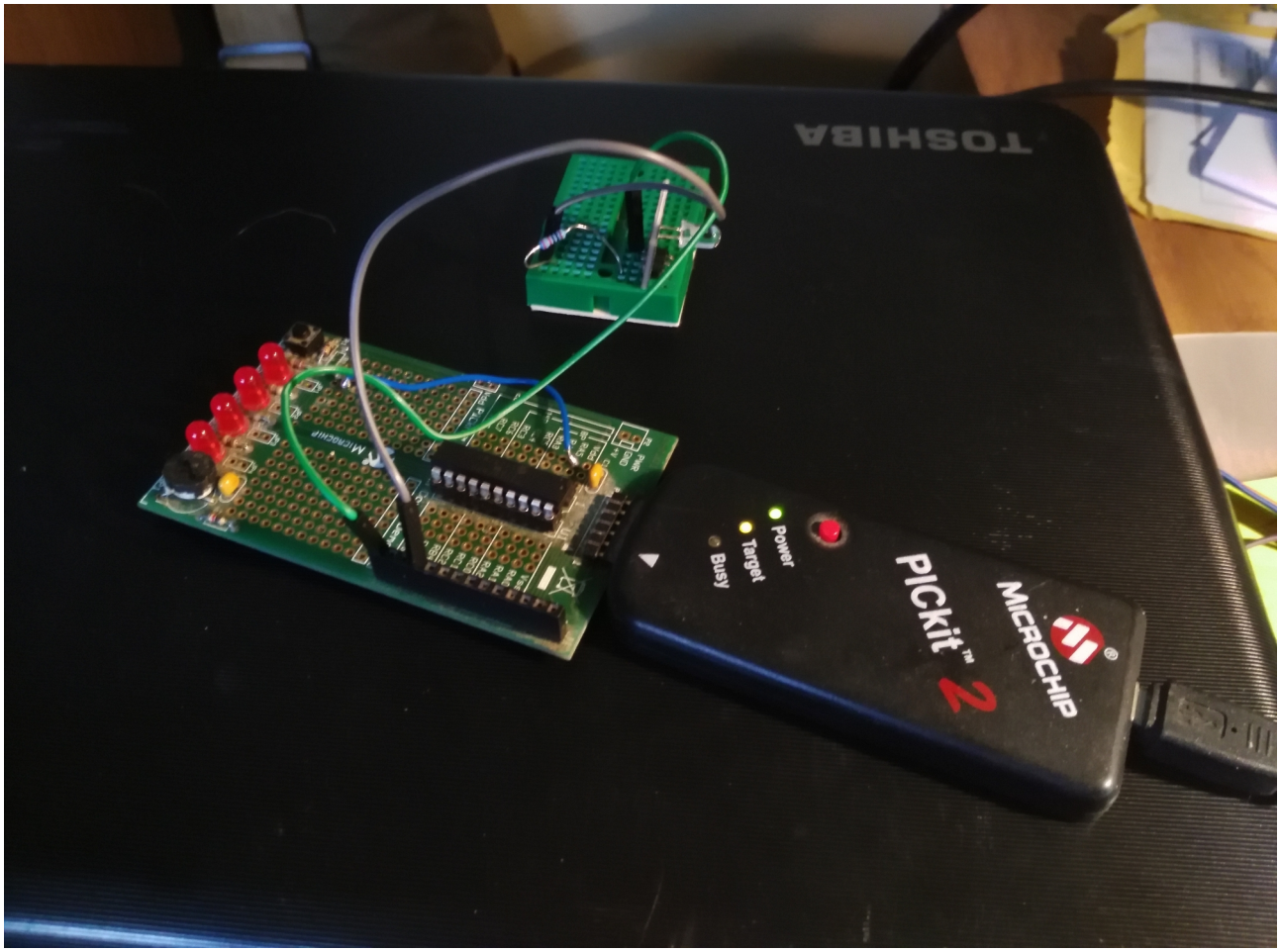
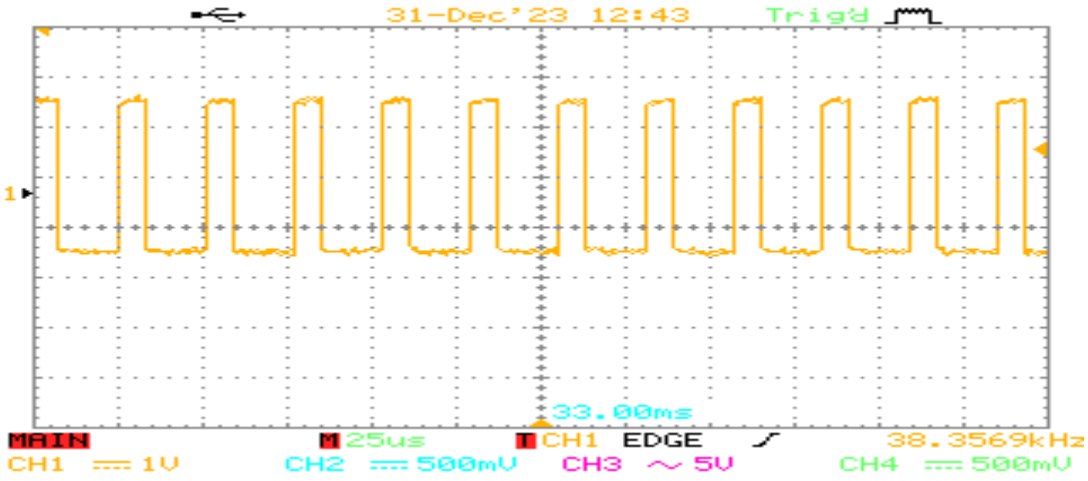



*Figure 5: Development Kit*

## Carrier Signal Generation

The carrier signal is fundamental to the generation of the control codes. A simple delay loop was found to provide an adequate  source.




This technique provided a 38.36kHz signal with the recommended 1/3 mark/space ratio.

## Code

The assembler code is listed in MPLAB Project ‘necir05’ and can be found in https://github.com/Starlight1856.

In the main program section from org 0x40, PORT A and B are configured as inputs with additional pull-ups. All seven input pins used are configured to interrupt on change. PORT C pins 0 and 1 are set as outputs driving the indicator LED and the IR Transmitter respectively.  On completion of setup the microcontroller is set to sleep mode. All unused pins are set as inputs.

When an interrupt occurs the interrupt service routine (ISR) at org 0x04 is called. This detects the button pressed which generated the interrupt and loads the command associated with the button and stores the value in the  IRMessage ram location. The SendMsg subroutine is then called to transmit the message utilising other subroutines in the process.

On completion of the message being transmitted a delay is called and the code exits the ISR and re-enters sleep mode. A repeat message was not implemented.

# 6 Construction

The PIC16F690 was mounted on a 20 pin socket on a small piece of veroboard and assembled as in the circuit diagram figure 4. The unit was powered by 2 AAA cells mounted in a holder producing nominally 3 volts (the PIC16F690 can operate from 2.5 to 5.5 volts).

This was all assembled in a standard remote case Figures 6,7 and 8.

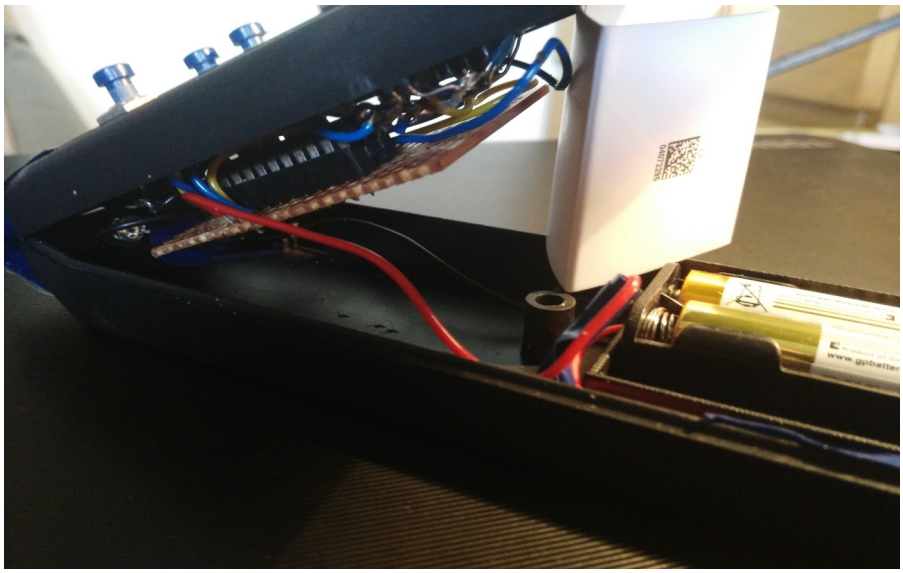


*Figure 6: Inside Case*

*Figure 7: Assembled Case*

*Figure 8: Assembled Case*

# 7 Testing

## Current Consumption

The current consumption was measured after the unit had settled into sleep mode as this is the state it will spend most of the time. The results are shown in figure 9 below.


*Figure 9: Current Consumption*

The measured current consumption was 42nA, which means a the 750mA/Hour batteries should last as long as their normal shelf life.

## Generating IR Messages

The transmitted codes were verified using the Arduino test setup as in figure 1. A video of the unit in use can be found in the folder ‘Testing’ under the IR Remote project in https://github.com/Starlight1856.

# 8 Appendix

## Parts

7x Push-Buttons

1 Two AAA Cell Battery Holder

1 PIC16F690 DIL

1 20pin IC DIL Socket

1 Remote Case

1 LED Colour Yellow

1 Infra Red LED

1 Small Piece VeroBoard 37mm X 56mm

8 x 100nF Capacitors

## Infrared LED Specification

5mm Infrared LED
Voltage (VF) : 1.1 to 1.4 V
Power : 0.15 W
Angle : 40 degrees
Wave length : 940 nm
Maximum forward current : 30 mA
Maximum reverse voltage : 5 v
Maximum pulse current peak : 75 mA

## Tools Used

MPLAB IDE v8.92 Running under Linux Mint 21, using PICKIT2

Arduino 1.8.19 running under Windows 10

Fritzing

KiCad

Keithley 197 DMM

GDS-2104 Oscilloscope