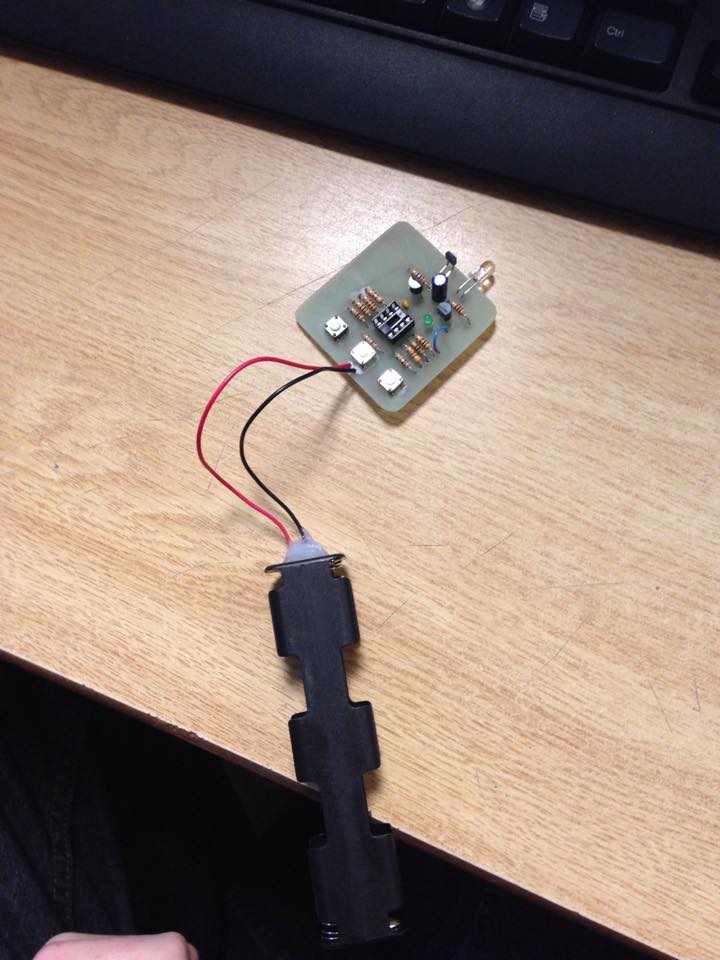
**Universal Remote Control Using Various Infra-Red Protocols**

****

Jan. 16, 2015

Michael Peet

**Abstract:**

Our objective in this project was to create a universal television remote using the Sony, RC5, and NEC infra-red protocols. I wanted the remote to be able to influence televisions from both far away and close up. I decided to use three buttons, one to control Power, and two to control the volume on the TVs. My aim was for the remote to be able to affect multiple televisions at once. My final goal was for us to go into somewhere like Futureshop or Walmart, press a button, and have all of the TVs in the store turn off.

**Table of Contents:**

1. Introduction
2. Description of Infra-Red Protocol
   1. Sony
   2. RC5
   3. NEC
3. Circuit Design
4. Program
   1. Ping Test
   2. Code Test
   3. Final Program
5. Frustrations
6. Summary
7. Appendices

**Introduction:**

Our aim in this project was to create a universal TV remote. It was to work on TVs operated by one of 3 Infra-Red Protocols. The Sony Infra-Red Code, the RC5 Protocol, and the NEC Protocol are standard protocols used for a variety of televisions. They all differ in transmission frequency, transmission format, and time period between code transmissions.

Television remotes are based on the transmission of infra-red light for wireless communications. The technology is often used in remotes and was invented to allow for simpler communication between two or more points.

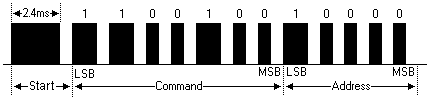
In this report I will be analyzing the three infra-red protocols used in the remote, the design of my circuit, the program I created to transmit the codes, and some frustrations I encountered.

**Description of Infra-Red Protocols:**

Sony Infra-Red Protocol:

Sony Infra-Red Protocol (or SIRC protocol), is an infra-red encoding method that uses a 40 kHz carrier wave. The command code is 12 bits long and is composed of a 7 bit command sent before a 5 bit address. 15 and 20 bit versions of the code are also used. Both the command and address are sent least significant bit (LSB) first. Commands are repeated every 45ms.

The actual code begins with a 2.4ms pulse burst followed by a 0.6ms space. A logical 1 is a 1.2ms pulse burst followed by a 0.6ms space, and a logical 0 is a 0.6ms pulse burst followed by a 0.6ms space. The recommended carrier duty cycle (time on vs time off per pulse) is 1/4 to 1/3. This is to conserve power.

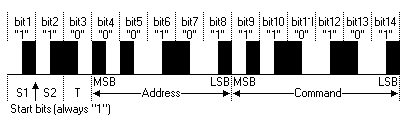


This is a sample pulse train for a Sony command code.

RC5 Protocol:

RC5 Infra-Red Protocol is an infra-red encoding method developed by Phillips that uses a 36 kHz carrier wave. The command code is 11 bits long, composed of a 5 bit address, sent before a 6 bit command. Both the command and address are sent most significant bit (MSB) first. Commands are repeated every 114ms.

The RC5 code starts its transmission with 2 logical 1s, and a toggle bit, which alternates between 0 and 1 each time the key is pressed. A logical 1 is an 889us pulse burst followed by an 889us space, while a logical 0 is an 889us space followed by an 889us pulse burst. The recommended carrier duty cycle is 1/4 to 1/3. Again, this is to conserve power.

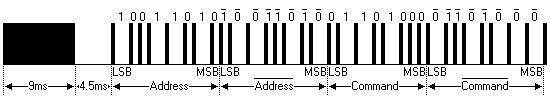


This is a sample pulse train for an RC5 command code.

NEC Protocol:

NEC Infra-Red Protocol is an infra-red encoding method that uses a 38 kHz carrier wave. The address code is 8 bits long, followed by its 8 bit logical inverse. They are sent and followed by the command code, which is 8 bits long, and followed by its 8 bit logical inverse. Both the command and address are sent LSB first. Commands are repeated every 40ms. Often a repeat code is sent after the first time the code is transmitted. It is repeated every 118ms.

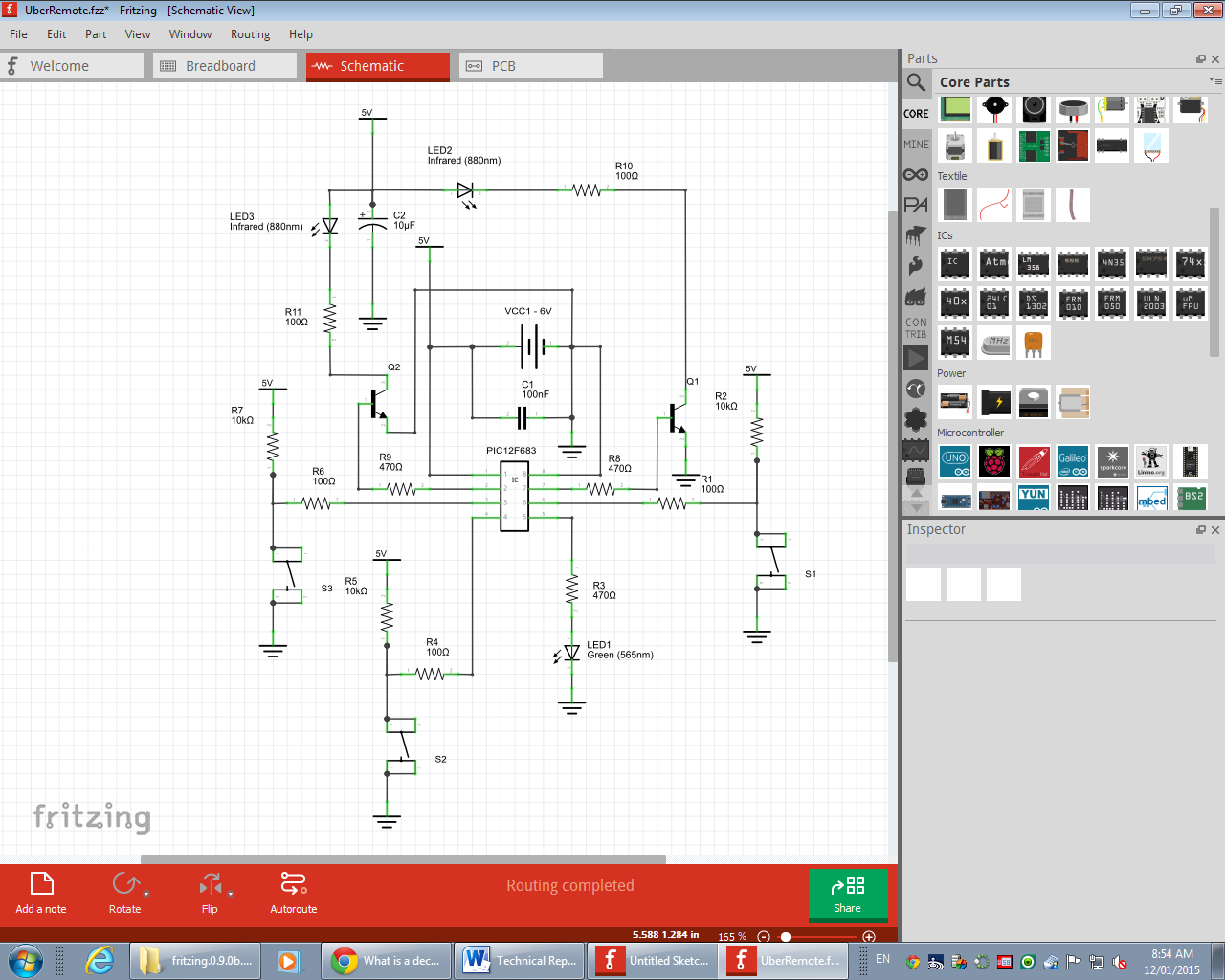
The NEC code starts transmission with a 9ms pulse burst, followed by a 4.5ms space. The 8 bit address is sent first, with a logical 1 being a 562.5us pulse burst followed by a 1.6875ms space, and a logical 0 being a 562.5us pulse burst followed by a 562.5us space. The inverse of the address is then sent, followed by the command, and then its inverse. Finally, a logical 0 is sent to indicate the end of the transmission.



This is a sample pulse train for an NEC command code.

**Circuit Design:**

We chose to use the PIC12F683 microcontroller as the central processing unit for my remote.



Each switch is connected to a pull-up resistor to protect the chip from surges of current if the inputs are configured incorrectly in the software. Stress on the chip is also reduced by a decoupling capacitor put in parallel with the power source to keep the voltage more constant. I chose to use 2 IR LEDs. One was to have a wide angle of effect for close TVs, and one was to have a narrower beam for ones that were further away. I also chose to hook LEDs up to transistors instead of directly to the chip. I hoped this would allow us to send more current through the LEDs as the chip could only supply 20-25mA of current. Because there would be a small time lag between when I wanted to turn the LED on, and when the full current from the battery would reach the LED, I added in a filter capacitor to supply current in the meantime. I also added a green status LED for debugging.

Program:

Our code went through many stages before the final program. I had ping tests, individual program testing, and then my final program, incorporating all programs into one. Each stage will have its own paragraph and explanation.

Ping Tests:

This section of my code was composed of 3 simple programs, each to test the buttons, and the frequency of the light pulses I transmitted. While each button was pressed, the program would call a ping function, which used a combination of commands and “nop”s to pulse the LED at a specific frequency:

void pingSony(unsigned char cPulses) {

for (counter = cPulses; counter != 0; counter--) {

Q1 = 1;

Q2 = 1;

asm("nop");

asm("nop");

asm("nop");

asm("nop");

asm("nop");

Q1 = 0;

Q2 = 0;

asm("nop");

asm("nop");

asm("nop");

asm("nop");

asm("nop");

asm("nop");

asm("nop");

}

}

The function takes in a number, which is the number of pulses the light is supposed to do and then the function changes the state of the transistors to allow power to flow from the battery to the LEDs, turning the light on. The light is then left on for a time using the asm(“nop”) command, which has the processor do nothing for one clock cycle. It is important to note that in this function, I also had to incorporate the number of clock cycles required for the counter in the loop to decrement (about 12). The frequency of the ping test programs were tested by hooking up the board to an oscilloscope, connecting ground to the ground pin on the LED, and power to the correct pin on the chip. Code for pinging at 36, 38, and 40khz were used, one for the frequency of each of the IR protocols.

Individual Code Tests:

This section of my code was composed of 3 slightly more complex programs, each building off of its respective ping function. For these programs, when the button is pressed, the correct television code is sent to an intermediate function, which calls the ping function the correct number of times for a 1 and a 0. It also takes into account the different transfer format for each code (ex. Philips code incorporates a toggle bit as the third bit in the program, which is flipped each time a key is pressed to determine if a key is being pressed multiple times or is just being held down).

void send(unsigned char cAddress, unsigned char cCommand, unsigned char c, unsigned char flip) {

if (c == 1) {

pingSony(91);

\_\_delay\_us(700);

i = 0;

j = 0;

while (i < 7) {

cTemp = cCommand & 0b00000001;

if (cTemp == 1) {

pingSony(44);

} else {

pingSony(21);

}

cCommand = cCommand >> 1;

\_\_delay\_us(680);

i++;

}

while (j < 5) {

cTemp = cAddress & 0b00000001;

if (cTemp == 1) {

pingSony(44);

} else {

pingSony(21);

}

cAddress = cAddress >> 1;

\_\_delay\_us(680);

j++;

}

}

Because this function is supposed to receive information for any of the codes, and transmit correctly, it has to receive the address, the command, a character, c, to indicate which code is being used, and a flip bit for the RC5 code. This will be used if necessary. One issue I encountered was with memory. In the beginning, I used arrays of unsigned characters which I looped through for each code. Once I got to the NEC function however, I realized that I did not have enough memory to hold all of the codes on the device. To solve this problem, I used binary numbers for each part of each code instead, using the bit shift function to loop through the bits as needed:

unsigned char arcSonyDown[] = {1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0};

pingSony(91);

\_\_delay\_us(700);

for (unsigned char i = 0; i < size; i++) {

====================== CHANGED TO ======================

unsigned char SonyDown = 0b00010011;

unsigned char SonyTV = 0b00000001;

unsigned char SonyPower = 0b00010101;

unsigned char SonyUp = 0b00010010;

pingSony(91);

\_\_delay\_us(700);

i = 0;

j = 0;

while (i < 7) {

cTemp = cCommand & 0b00000001;

cCommand = cCommand >> 1;

Another issue I encountered was that the internal timer in the chip is not perfectly accurate when making time delays. To have a 600us space in practice, I needed a 680us delay called in code.

Final Program:

This was the final stage of the process. My goal was to incorporate all of the programs together, and be able to affect Sony, RC5, and NEC televisions at the same time. I decided to copy the code in to make one large program. Because Sony codes have to be sent at least 3 times to be registered, I needed to make another function, charmingly named derp, to send the codes multiple times each:

void derp(unsigned char cAddress, unsigned char cCommand, unsigned char c, unsigned char flip, int nDelay) {

for (int i = 0; i < 5; i++) {

send(cAddress, cCommand, c, flip);

if (nDelay == 25) {

\_\_delay\_ms(25);

} else if (nDelay == 114) {

\_\_delay\_ms(114);

} else if (nDelay == 140) {

\_\_delay\_ms(140);

}

}

}

While Sony only needs to be sent 3 times, I have sent all codes at least 5 times to ensure they get registered by the television. My first version of the program did not work at all. The light was not even turning on. By making a new project however, and copying in code line by line, I were able to have a working program with no issues whatsoever. One possibility is that one of the ports was incorrectly configured (while I did not see any when looking).

Frustrations:

While my project was eventually successful, I encountered many setbacks. One of them was with my soldering. Three parts were soldered in incorrectly, causing the board to not work. Originally I thought that the issue was a software problem, meaning I spent many hours attempting to find a nonexistent error in my code. The fact that some parts were soldered in wrong and that I had been looking in the wrong place was a frustrating discovery.

Another frustration was with my switches. Sometimes when I ran the program, one switch constantly registered that it was pressed. Which switch it was changed quite frequently, making the problem difficult to track down. At one point, the middle switch would not work no matter what I did. I decided to desolder it and eventually got to a point where if you wiggled it, it would work for a while, and then you would need to wiggle it again. Eventually this method stopped working entirely, and I needed to resolder the switch back in. When another switch refused to register that it was pressed, I realized that it was because the MCLR pin was turned on. As a result, pushing the button would reset the code.

Summary:

Despite multiple setbacks, my project was mainly successful. It is able to send out codes for multiple televisions. It has been tested on Sony TVs, and will be tested on NEC and RC5 TVs in the near future. Because the general code logic works and multiple codes can be sent out in sequence, the road to perfecting the other codes is just tweaking the bit ordering. Unfortunately, while the power works quite well, the volume control is quite slow, and not very convenient to use. In future releases, I would like to make the volume controls work faster. I also want to change the LED configuration to allow for longer range on the narrow angle LED. Later this year, I hope to make a board that will respond to one of the sent codes, and will turn on a bank of lights.

References:

SBProjects: Sony: <http://www.sbprojects.com/knowledge/ir/sirc.php>

RC5: <http://www.sbprojects.com/knowledge/ir/rc5.php>

NEC: <http://www.sbprojects.com/knowledge/ir/nec.php>

Altium Techdocs: RC5: <http://techdocs.altium.com/display/FPGA/Philips+RC5+Infrared+Transmission+Protocol>

NEC: <http://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol>

Appendices:

Please see appendix folder for supporting documents.