**The Universal Remote - Team 29**

by

John Capper \_\_\_\_\_\_\_\_\_\_\_

Nicholas McClellan \_\_\_\_\_\_\_\_\_\_\_

A Technical Report Submitted to the Faculty of

Electrical Engineering

Colorado School of Mines

Submitted in partial fulfillment for the requirements of

EENG 383 – Microcomputer Architecture and Interfacing

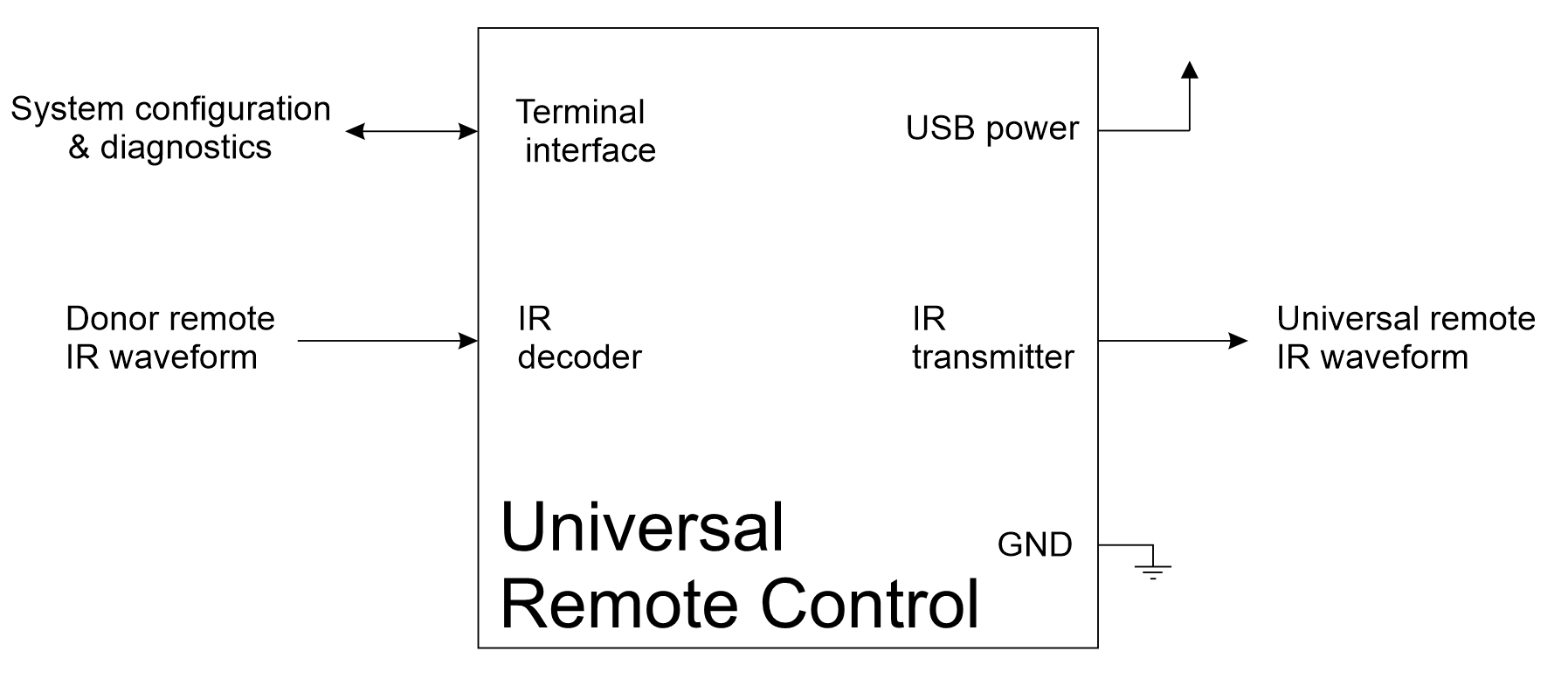
May 5, 2020

## Chapter 1: Design Goals

### Problem Statement

In a world fueled by digital technology, IR remote controls are commonly used across many different applications. Thus, cloning remote signals can be beneficial for many reasons, such as learning how IR data packets behave, or creating a new universal remote. For this project, the Dev ‘20 board will be turned into a Universal Remote control. The goal of this project will be to successfully receive, clone, and reproduce IR signals sent from donor remotes.

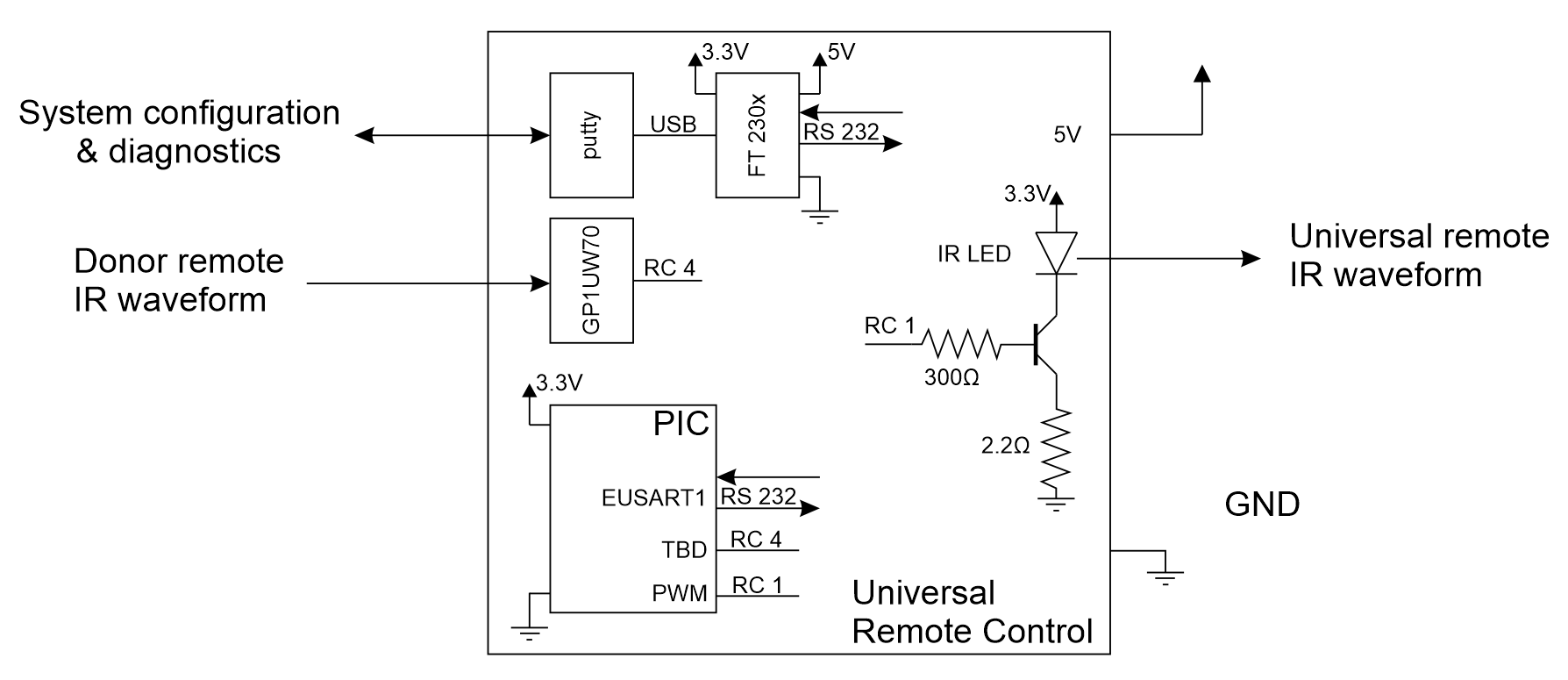
### Level-0 Description



**Figure 1. Level 0 Diagram**

The system shown in Figure 1 will meet the needs provided in the problem statement by allowing users to decode and transmit IR signals . Figure 1 shows the general block diagram of the Universal Remote Control system. To begin with, the system will require some sort of ground connection. Further, for power, a 3V USB power source from a computer will be used. When the user presses a button to send an IR signal, it will be received by the IR decoder. Next, the signal will be decoded by the PIC into a 32-bit hex value. From there, the signal can be transmitted by the IR transmitter. The system configuration will be input via terminal and allow the user to choose whether they would like to receive and decode a signal or transmit a cloned signal.

## Level 1

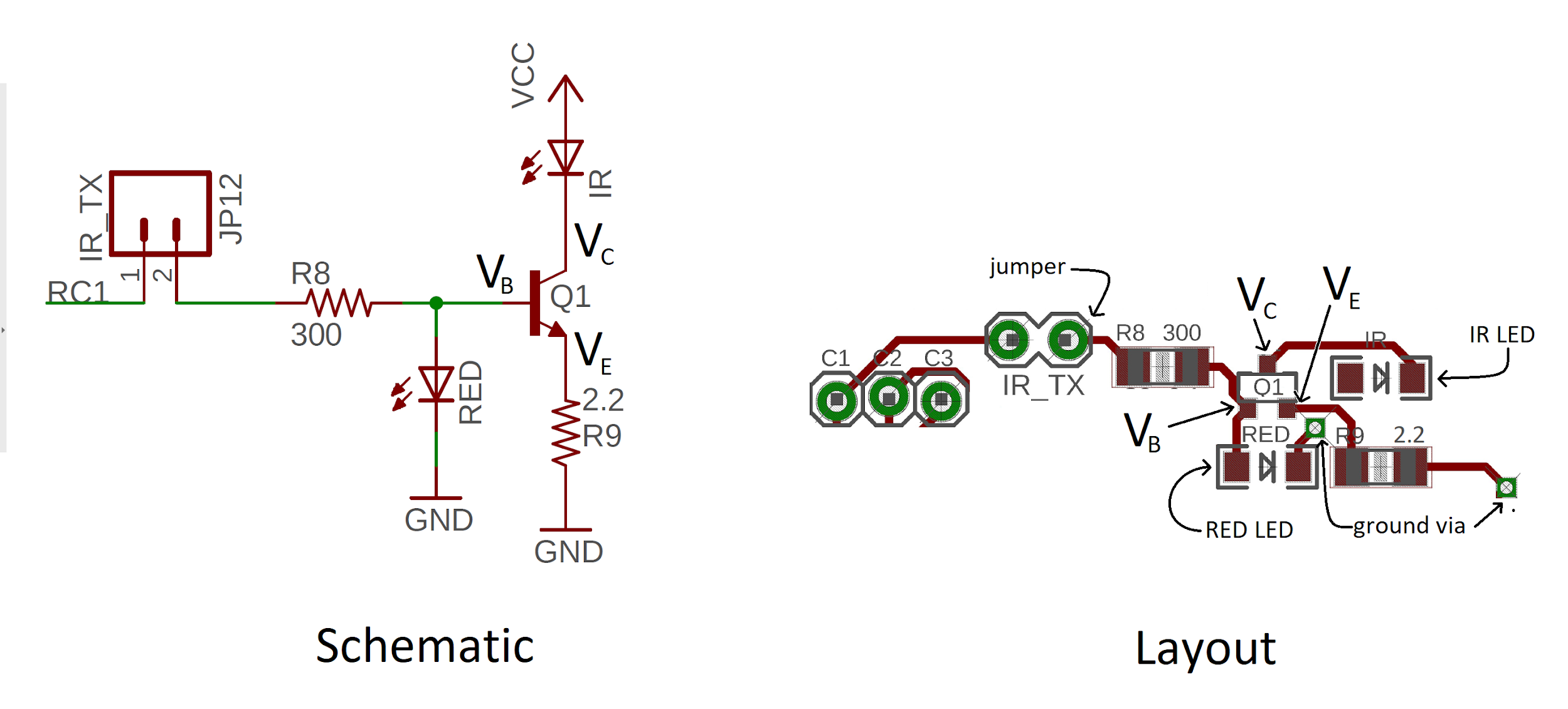


**Figure 2. Level 1 Diagram**

The Level-1 diagram of the Universal Remote Signal is shown above in Figure 2. To power the system, the board will be connected to a 3 V USB source. If a command is received by Putty to receive, when the user presses a button to send an IR signal, it will be captured by the IR decoder. From there the data will go to the ECCP1 module (configured to capture) where the durations between edges will be measured. Next, a PIC function will decode the half-bit durations into their corresponding 32-bit code value. When a command is received by Putty to transmit, the 32-bit code is converted back into half-bit durations and then used by the compare subsystem of the ECCP2 module. When TMR1 matches the CCP2R register, the RC1 pin will toggle to produce a PWM waveform which is outputted by the IR LED. If everything works appropriately, the output of the IR LED should replicate the button pressed on the donor remote.

## Transmissions Subsystem

### TX Hardware



**Figure 3. Hardware of IR LED**

Figure 3 above shows an annotated hardware diagram of components associated with illuminating the IR LED. Since the voltage at is 1.6V, when the voltage of RC1 is set to 3.3V (logic 1), and the current flowing through the IR transmitter becomes and the transmitter is turned on. The role of the RED LED is to act as a visual aid when there is an excess current between R8 and , or the BJT is closed. Finally, the 2.2 ohm resistor, R9, is

## ECCP2 and TMR1 Subsystem

For this project, we used the compare mode of the ECCP2 module using TMR1 as the timer. This was done by setting the following configuration registers:



With this setup, when the value of TMR1 matches the value stored in CCPR2, the RC1 pin is toggled, producing a PWM waveform. The duration of a pulse can be set by advancing the value of the CCPR2 register by the associated number of TMR1 counts. For example, if we wanted a pulse to be 800 μs long with a TMR1 prescaler of 1:8, you would do:



When all seven reference durations are known, a signal can be replicated from the cloned message data by transmitting the corresponding durations for a 1 or 0 as well for the start and stop bits. However, since the message is stored as a 32-bit value, it must first be read in as a binary sequence. This was done by first shifting the entire message over so that the most significant bit was now in the least significant bit column. Then, the message was shifted back to the right, one by one, each time checking if the result was divisible by 2. This was implemented with the following:



## Receive Subsystem

For this project, TMR1 and ECCP1 were used to receive IR signals with capture mode. This method relies mainly on the TMR1 and CCPR1 registers which were configured in MCC.

TMR1 was enabled and set to use a prescaler of 1:8 with the following code:



CCPR1 was enabled, set to capture mode, associated with TMR1, and set to toggle pin with the following code:



When an edge occurs on RC2, the TMR1 register value is copied over to the CCPR1 register. In order to capture both edges of the waveform, a function was written to toggle the trigger of ECCP1 after each capture as shown below:



This allowed the timer counts at each edge to be recorded. The durations between each edge were then calculated by taking the count difference between adjacent edges. From there, the durations were converted to microseconds by using dimensional analysis. For example, with a 1:8 prescaler of TMR1 and a duration of 1450 timer counts:

With the durations of dataZeroLow, dataZeroHigh, dataOneHigh, and dataOneLow, the signal could be converted into a 1 or 0. If a low-high pair had a low pulse duration of within 10% of dataZeroLow and a high pulse duration within 10% of dataZeroHigh, the corresponding value was interpreted as a 0. In similar fashion, if a low-high pair had a low pulse duration of within 10% of dataOneLow and a high pulse duration within 10% of dataOneHigh, the corresponding value was interpreted as a 1.

## 

## Milestones

### Milestone I

| Test name: Donor remote half bit periods | |
| --- | --- |
| Modules: | IR decoder and PIC |
| Setup: | PIC function to measure and report all the half bit durations of an IR waveform. The durations will be reported in timer counts. We will then post process the timer counts using an Excel spreadsheet. |
| Input: | Donor remote button press |
| Expected output: | Durations for all 7 half bits of the donor remote measured in timer counts and duration. The average of the four half bits of the data 0 and data 1 bits will be averaged in an excel spreadsheet and rounded to 1 or 2 significant digits. Different half period bit durations that are withing about 10% of one another will be assumed to be equal. This information will then be put into a table.  We will expect startLow to be roughly double the duration of startHigh. In addition, we will expect dataZeroLow to have approximately the same duration as dataOneLow. |
| Observed Output | | Bit | Timer Counts | Duration (us) | | --- | --- | --- | | startLow | 1580 | 890 | | startHigh | 750 | 385 | | dataOneLow | 1120 | 600 | | dataOneHigh | 1140 | 570 | | dataZeroLow | 1120 | 600 | | dataZeroHigh | 3300 | 1650 | | stopLow | 2000 | 1000 |   As shown in the table above, startLow is roughly twice as long as startLow. In addition, dataZeroLow has approximately the same duration as dataOneLow. All this was to be expected. |

| Test name: Count the number of edges in donor IR waveform. | |
| --- | --- |
| Modules: | IR decoder and PIC |
| Setup: | PIC function will prompt the user to press a button on the donor remote and then report the number of positive and negative edges in the IR waveform. This function is needed because different IR remote controls may have a different number of bits. |
| Input: | Donor remote button press |
| Expected output: | The count of the number of positive and negative edges should be the same. If there are 32 data bits, then there should be 66 positive and negative edges. |
| Observed Output | There were 66 positive and negative edges as expected |

| Test name: Donor remote button codes | |
| --- | --- |
| Modules: | IR decoder and PIC |
| Setup: | PIC function to decode the logic high half bits of the data bits into 0 or 1 depending on their duration. The durations will be hard-coded into the program based on the results of the earlier test. |
| Input: | Donor remote button press |
| Expected output: | Repeatable code outputs for the different donor remote control buttons.   | Donor button | 32-bit code | | --- | --- | | Power | 0x30D07807 | | Up | 0x30D0215E | | Down | 0x30D0611E | | Left | 0x30D0344B | | Right | 0x30D0542B |   We will closely examine the codes, looking for patterns. For example, the RC5 standard breaks the 32 data bits into different fields. Some remotes break the data packet into two halves and make all the bits in the second half the negation of the first 16-bits. This enables a level of error detection on the receiver's end. |
| Observed Output | | Donor button | 32-bit code | | --- | --- | | Power | 0x30D07807 | | Up | 0x30D0215E | | Down | 0x30D0611E | | Left | 0x30D0344B | | Right | 0x30D0542B |   The table above shows the 32-bit codes of different donor buttons. |

### Milestone II

| Test name: Converting 32-bit remote codes back into binary sequences for transmitting | |
| --- | --- |
| Modules: | PIC |
| Setup: | PIC function will prompt the user to press buttons 1-4 on the donor remote and then store the decoded 32-bit value for each. PIC function will convert 32-bit code back into binary sequences and display. This function is to ensure that the signal will be reproduced correctly. |
| Input: | Donor remote button press |
| Expected output: | | **Button** | **32-bit hex** | **Binary Sequence** | | --- | --- | --- | | 1 | 0x30D0007F | 110000110100000000000001111111 | | 2 | 0x30D0403F | 110000110100000100000000111111 | | 3 | 0x30D0205F | 110000110100000010000001011111 | | 4 | 0x30D0601F | 110000110100000110000000011111 | |
| Observed output: | | **Button** | **32-bit hex** | **Binary Sequence** | | --- | --- | --- | | 1 | 0x30D0007F | 110000110100000000000001111111 | | 2 | 0x30D0403F | 110000110100000100000000111111 | | 3 | 0x30D0205F | 110000110100000010000001011111 | | 4 | 0x30D0601F | 110000110100000110000000011111 |   The binary sequences are equal to what was expected for each button |

### 

| Test name: IR Waveform Production | |
| --- | --- |
| Modules: | IR LED Transmitter and PIC |
| Setup: | Enable ECCP2 in Compare Mode. Set to toggle RC1 PIN on TMR1 match. Write PIC function to transmit pulse durations corresponding to 1 or 0 based on cloned remote code. |
| Input: | Cloned button presses (1-4 in PuTTY) |
| Expected output: | The cloned buttons produce IR waveforms in an attempt to imitate the donor remote control. |
| Observed Output | When the various buttons were pressed, a digital camera was used to detect the presence of the IR waveforms, as the light was then visible. All of the cloned donor buttons did output IR waveforms, but the difference between each code was not something that could be distinguished by the human eye. |

### 

| Test name: Single Button Reproduction | |
| --- | --- |
| Modules: | IR LED Transmitter and PIC |
| Setup: | Enable ECCP2 in Compare Mode. Set to toggle RC1 PIN on TMR1 match. Write PIC function to transmit pulse durations corresponding to 1 or 0 based on cloned remote code. Then, use the cloned buttons to produce said waveforms from the IR LED. |
| Input: | The cloned donor button and furthermore the proceeding press of the corresponding key in PuTTY (1-4). |
| Expected output: | | **Button** | **TV Response** | | --- | --- | | Up | TV goes up one channel | | Power | The power state of the TV is toggled | | 1/2/3/4 | Any of the cloned number keys input # | |
| Observed output: | | **Button** | **TV Response** | | --- | --- | | Up | None | | Power | None | | 1/2/3/4 | None | |

### Final Implementation

The reliability of the transmission in terms of operating the TV is non-existent. However, the other portions of the code have been fairly consistent. The reliability of our transmission ties in a lot with the lessons that we learned. The lesson being that eventually there is going to be a problem that you cannot solve by yourself. At least not in a timeframe that would have been beneficial to the completion of the task at hand. Moreover, there may come a time where reaching out for guidance or other help is in the best interest of both time and the completion of the assignment. If we were to do things a bit differently it would be to reach out for assistance earlier. Ultimately, we failed to get aid when it was available and in turn were unable to complete the last bit of the project.

## 

## 

## References

[1] Microchip, “28/40/44-Pin, Low-Power, High-Performance Microcontrollers with XLP Technology,” PIC18(L)F2X/4XK22 datasheet. 2010.

## 

## Appendix A: Running the Project

The Universal Remote control project was written completely in C code and compiled using MPLab’s standard XC8 compiler. For this project to work on the Dev ‘20 board, the ECCP1 module needs to be set to capture and the ECCP2 module needs to be set to compare. In addition, the IR decoder output needs to be connected to the ECCP1 input by placing a jumper wire between RC2 and RC4 and the ECCP2 output needs to be connected to the IR transmitter input by placing a jumper across IR\_TX.

## 

## Appendix B:

<https://drive.google.com/drive/folders/19IQDj8v1Kf8YiNVKWQgLq-rJOArV3w0g?usp=sharing>

The above link contains the video presentation, milestone demos, final report, and source code