Johnny Li
4/26/2020
9699 4228

FINAL PROJECT – Report

Introduction

The goal of my final design is to mimic an infrared (IR) remote receiver for multiple remotes such as those used for TV remotes. I will be using the given microcontroller, PIC18F47K40, to process inputs from different IR remotes to change the “channels” being displayed on the LCD. This is done by using a photodiode to detect incoming IR signals and the capture mode on the microcontroller to read the signal. There will be an I^2C timer button to set the time and alarm with LEDs to indicate a unit change (ex: hours, minutes, seconds). The time and alarm mode will be controlled by a digital input from a switch and a button to enable or disable the alarm. A potentiometer will be used to alter the voltage value to the microcontroller ADC to select the time values. The time, channel, and alarm status will be displayed on the LCD. If the alarm has triggered, a buzzer will go off till the alarm is disabled. An external DAC using an SPI interface will be used to drive a sin, triangle, or sawtooth wave relative to which channel is on.

IR remotes send signals in terms of duration of high and low while the IR receiver detects and decode the durations to interpret the remotes command. Each remote transmission consists of two ordered parts: sync and its message. Long sync is usually sent first and only once per transmission, ex: 000111111111100. It is used as an identifier for the remote and signifies that the message will follow. This can be seen below in image 1:

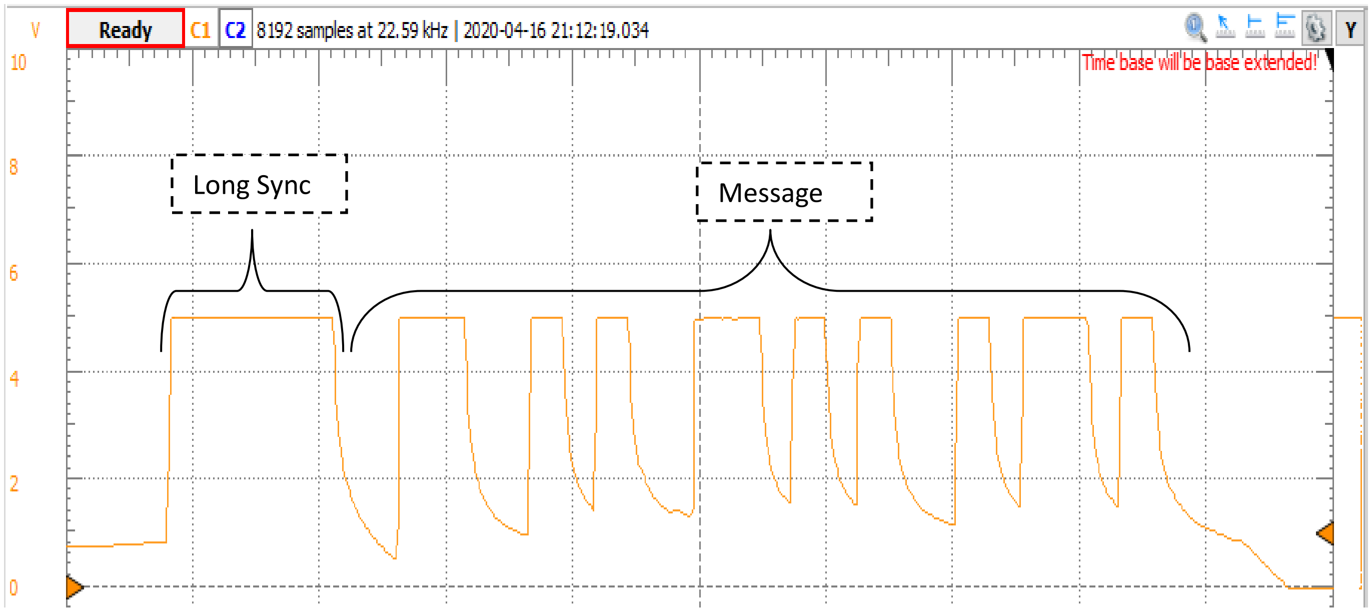

V
Ready
C1
C2
8192 samples at 22.59 kHz | 2020-04-16 21:12:19.034
Time base will be base extended!
10
8
6
4
2
0
Long Sync
Message
Y

Image 1: IR signal

Depending on the IR remote standard: RC5 standard- the row is repeated as long as the button is pressed or RECS80 standard- if a unique message is to be sent more than once in a transmission, each subsequent row with this repeated message will send a short sync instead of a long sync, ex: 0001111100. The standards are seen below in the images:

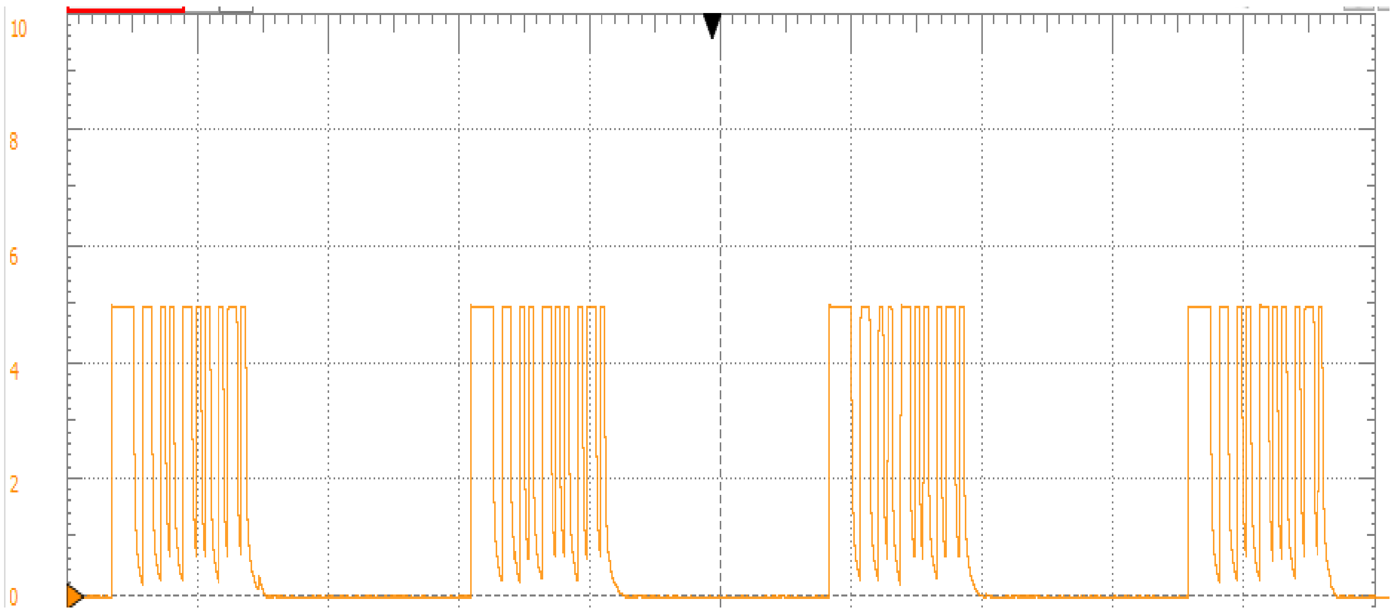


Image 2: RC5 Standard- repeating row transmission.

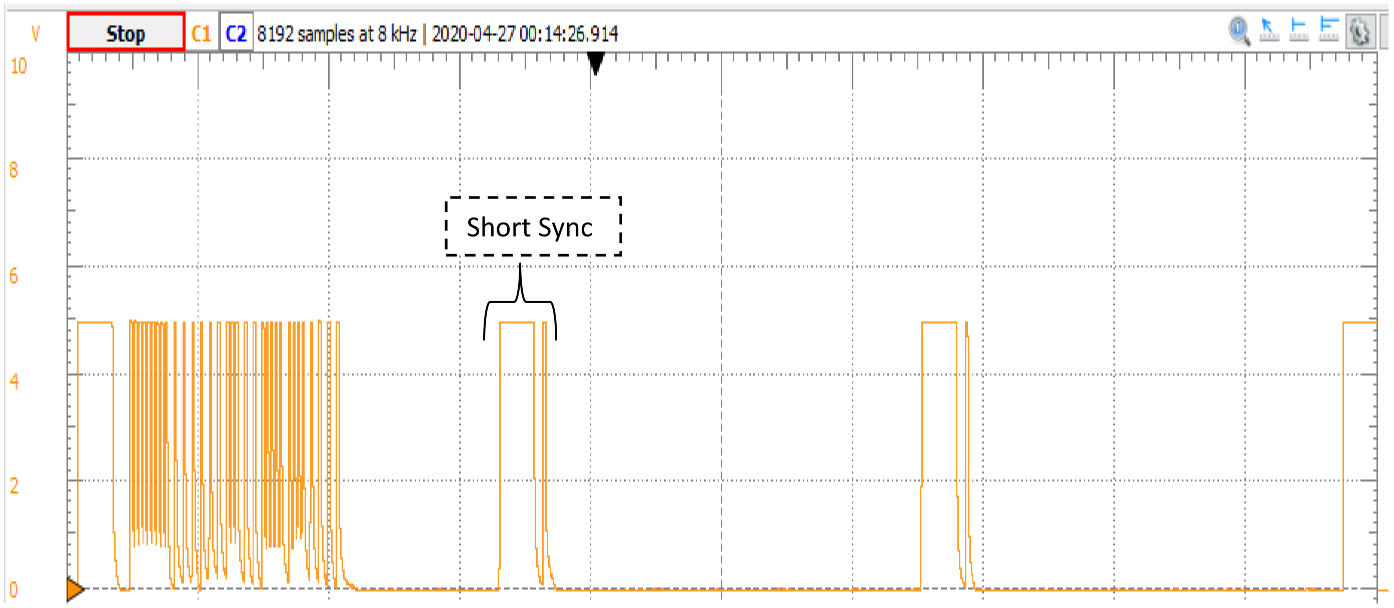


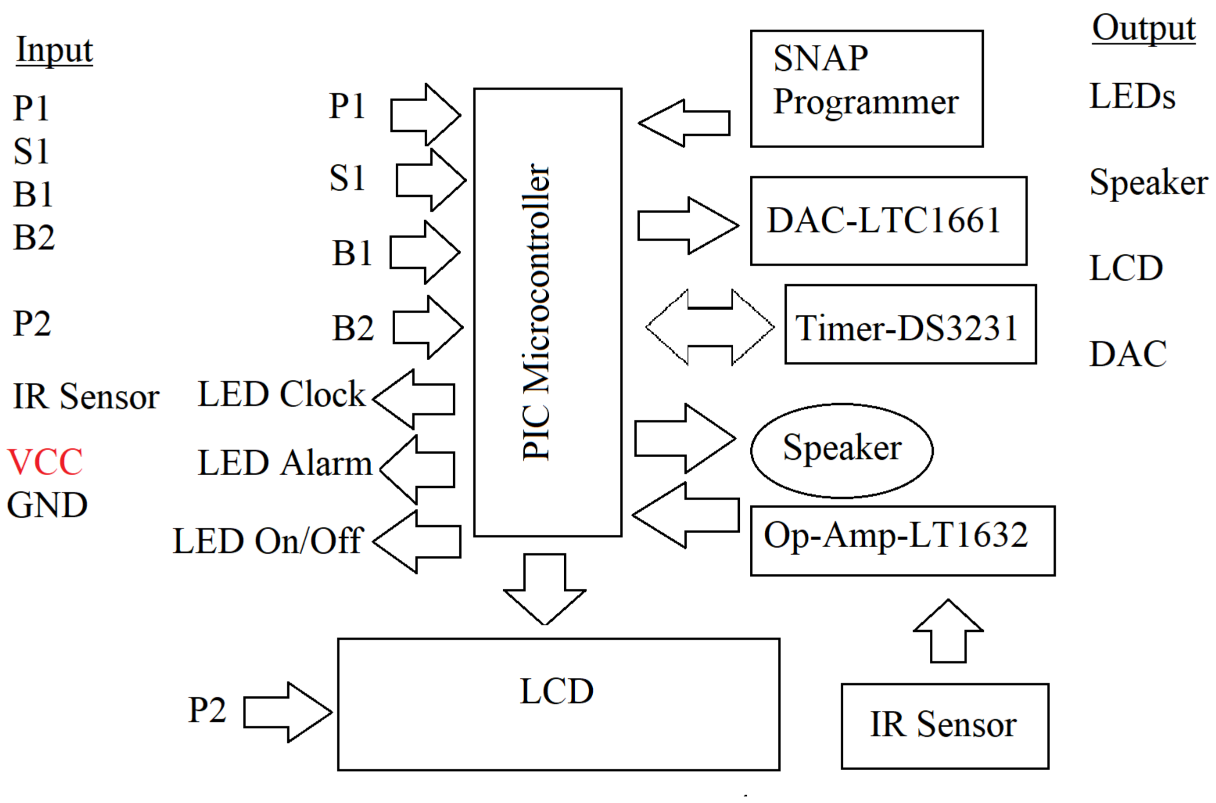
Image 3: RECS80 Standard- repeating short sync after initial transmission.

To decode the IR signals requires using some sort of Differential Pulse Width Modulation (DPWM) method. Between each symbol transition (from 1 to 0 or 0 to 1) count the number of symbols that exist in that state. If there is only one symbol- the data bit is a 0, two symbols- the data bit is a 1, and if ignore anything with 3 or more symbols for this design.

Note: For the final design, several IR remotes are used with each being its own channel. The length of the long syncs was used to identify the remotes.

Design

Image 4: Hardware connection block diagram.

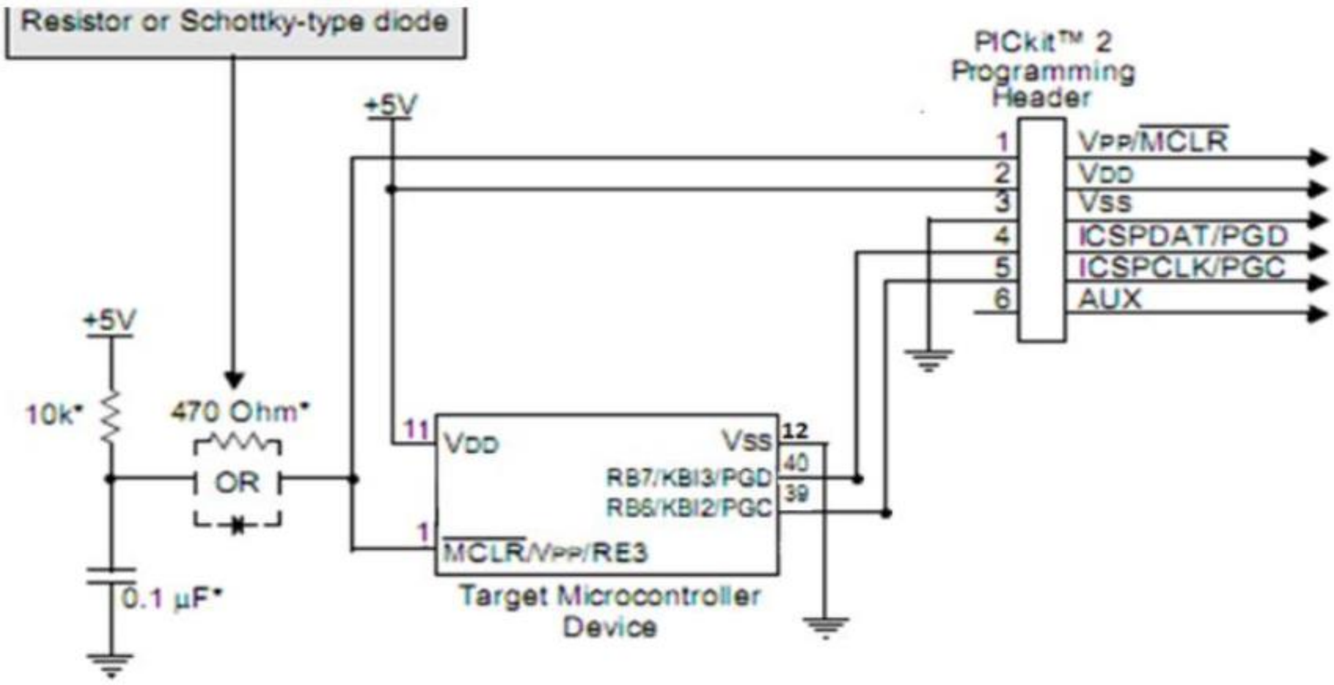


There are 8 main components to be implemented to create the microcontroller design, starting with the setup of microcontroller and programmer, input controls, output LEDs, LCD, DAC wave functions, I^2C timer, alarm buzzer, and IR photodiode sensor. The microcontroller is to be properly wired to the SNAP programmer, having the master reset, clock, and data pins connected, with both having their power and ground sources. The microcontroller also requires a connection with the master reset to power, having a capacitor in between to dampen noise. The time and alarm are configured through the input controls, including a switch, some buttons, and a potentiometer, with output LEDs signaling a change in the unit of time. The time setting is passed to the microcontroller which then communicates to the DS3231 RTC through I^2C to set or retrieve the time and alarm. An ADC configuration in the microcontroller is used to drive the potentiometer to be able to vary the units of time. When the time of RTC matches the set alarm time, then an interrupt occurs and outputting an active high signal on the RTC. The microcontroller then reads the active high signal from the RTC and signals the piezoelectric speaker to go off till the reset button is pressed. The current time or alarm is displayed on the LCD device, a CFAH1602Z, depending on which mode is selected. The microcontroller contains the functions to deliver commands and characters to be shown based on the input controls. The input signals from the IR sensor were at too low of a voltage so an op-amp, LT1632, was used to up its voltage to 5V so that the microcontroller could detect it. From decoding the IR signals received from 3 different remotes, channels ranging from 0 to 2 can be obtained. The current channel value is displayed on the LCD and the microcontroller then outputs a generated wave values via SPI to a DAC device, an LTC1661, relative to the current channel. A sin wave for channel 0, triangle wave for channel 1, and sawtooth wave for channel 2.

Assume that the microcontroller has the functional program installed, using the SNAP to connect the PC and microcontroller. The microcontroller first initializes the LCD for 4-bit mode and RA0 on the microcontroller as ADC. The required I/O ports are enabled and I^{2}C is set up with the RTC. Then SPI is initialized for the DAC wave output. Lastly, the CCP was prepared to handle the receiving and decoding of the IR signals. Using switch 1 to select the clock mode, I initialization the clock time with the potentiometer 1 to dial the value and button 1 to switch between the units of time, an LED blinks to indicate the change. Then switch 1 is flipped to select alarm mode, the same time initialization process is then repeated to set up the alarm. In the microcontroller, some code is started to convert the ADC value to a time value which is then transferred to the RTC. Button 2 is pressed to set the alarm. For the LCD output, the microcontroller reads from RTS and outputs to the LCD. The microcontroller also checks for an active high signal from the RTS's interrupt pin, if so, the speaker goes off till button 2 is press, else no speaker tone is made. The CCP is to trigger for every edge and through an interrupt will attempt to measure the duration of the long sync when a signal is received from the op-amp. Due to the input signals from the IR sensor were at too low of a voltage so an op-amp with a gain of 22 was used to up its voltage to 5V so that the microcontroller could detect it. Depending on the channel decoded, 0-2, the wave value is outputted to the DAC and the generated wave is checked to see if it was correct through the oscilloscope. An image of the hardware connection block diagram is shown above. Note that the potentiometer 2 is used to adjust the contrast on the LCD to make the display visible.

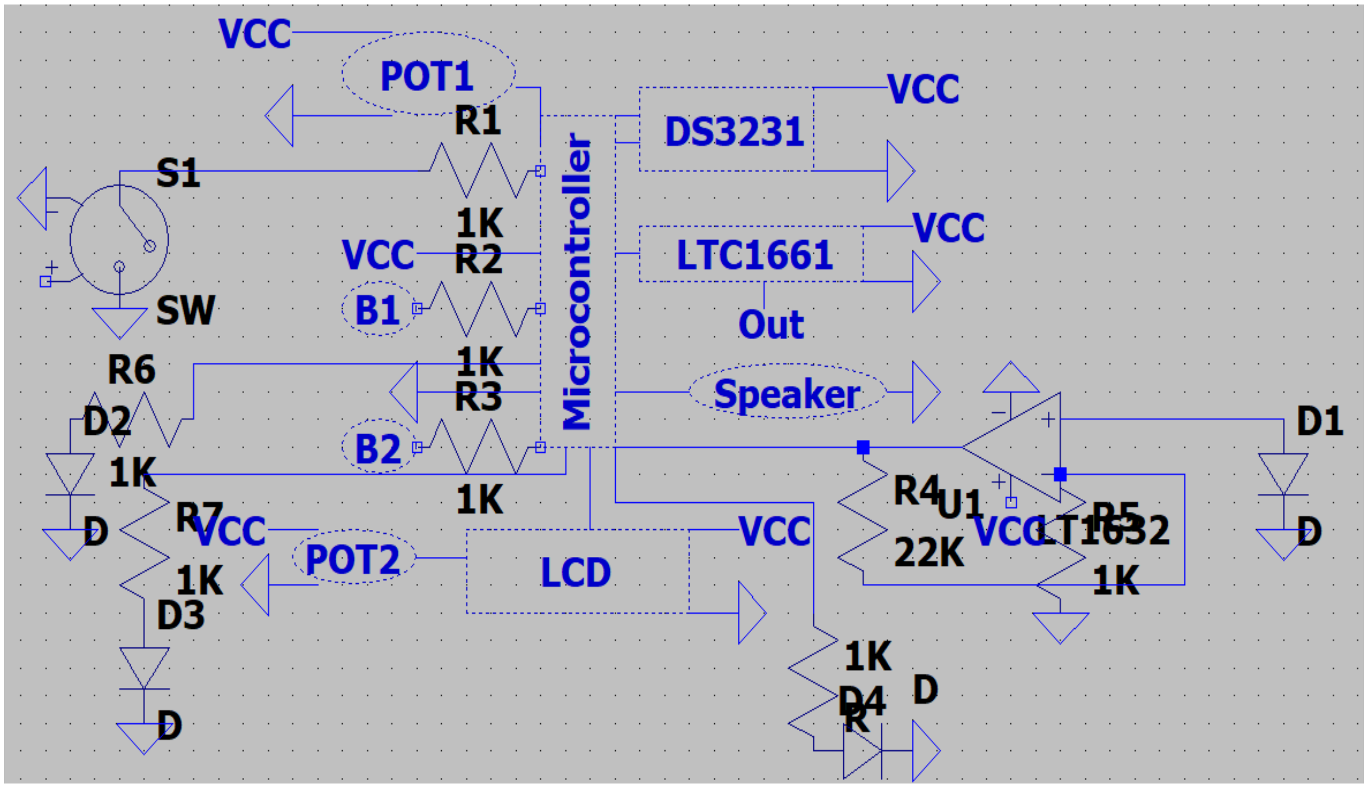
Note general microcontroller pin value would be represented as Mpin# while SNAP pin value would be represented as Spin# in the report. The setup of the microcontroller and SNAP can be seen below in image 2. This included some resistors and a capacitor to handle the master reset on Mpin1, to always keep the pin at high while reducing the noise it receives. The power of 5V was connected to the Mpin11 and Mpin32 while the ground was connected to the Mpin12 and Mpin32. The output data, Spin4, and clock, Spin5, from the SNAP was wired to the respective input data, Mpin40, and clock, Mpin39, on the microcontroller. Also, the output master reset, Spin1, from the SNAP connected to the microcontroller Mpin11. Power was connected to the SNAP through Spin2 while the ground was connected to the Spin3.

Image 5: Programmer and microcontroller setup



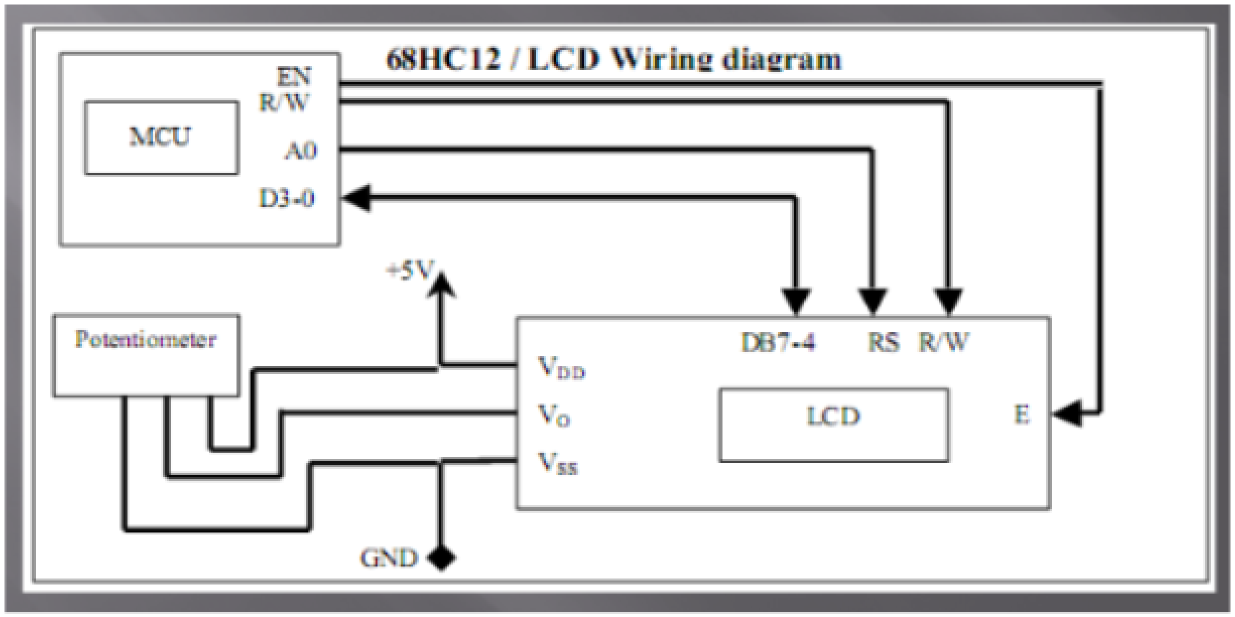
Note that the hardware schematic will omit the SNAP programmer and master reset additions for the convenience.

Schematic 1: Final Design



As shown in schematic 1: switch 1 change the LCD from displaying clock time to alarm time and potentiometer 1 is wired to RA0 input thus an ADC value is obtained potentiometer 1 is used to change alarm values in minutes, seconds, and hours. Button 1 is used to change the unit of time and button 2 is to set or reset the alarm. LEDs are used to indicate a change in the state of the clock and the speaker was triggered through toggling on and off its pin. A resistor is added to the input devices to limit the current through the component so that it doesn't damage the components. An image of the LCD wiring can be seen below. The microcontroller Port D is configured to be the output for the LCD. For the LCD, pin 1 is connected to VCC and pin 2 is connected to GND, this lights up the LCD. The backlight can be turned on by powering pin 15 and grounding pin 16 but this is not needed. A potentiometer was connected to pin 3, variable voltage pin, of the LCD to be able to adjust the contrast on the LCD to make the display visible. The microcontroller RC4 pin is connected to the LCD pin 4 which is the RS pin meaning that when high, data is read or written, and when low, instruction is written to. This is important for the functions lcd_char() that writes a character to be displayed and lcd_command() that writes a command to modify the LCD configuration. LCD pin 5 is for switching between reading and writing were since the program is always writing and not reading it can be grounded. LCD pin 5 is for enabling read or write, it is connected to RD7. LCD pin 4-7 are for data bits that are connected to RD0-3 respectively. The LCD is configured for 4-bit mode so only the upper 4 data bits are used rather than the 8-bit available.

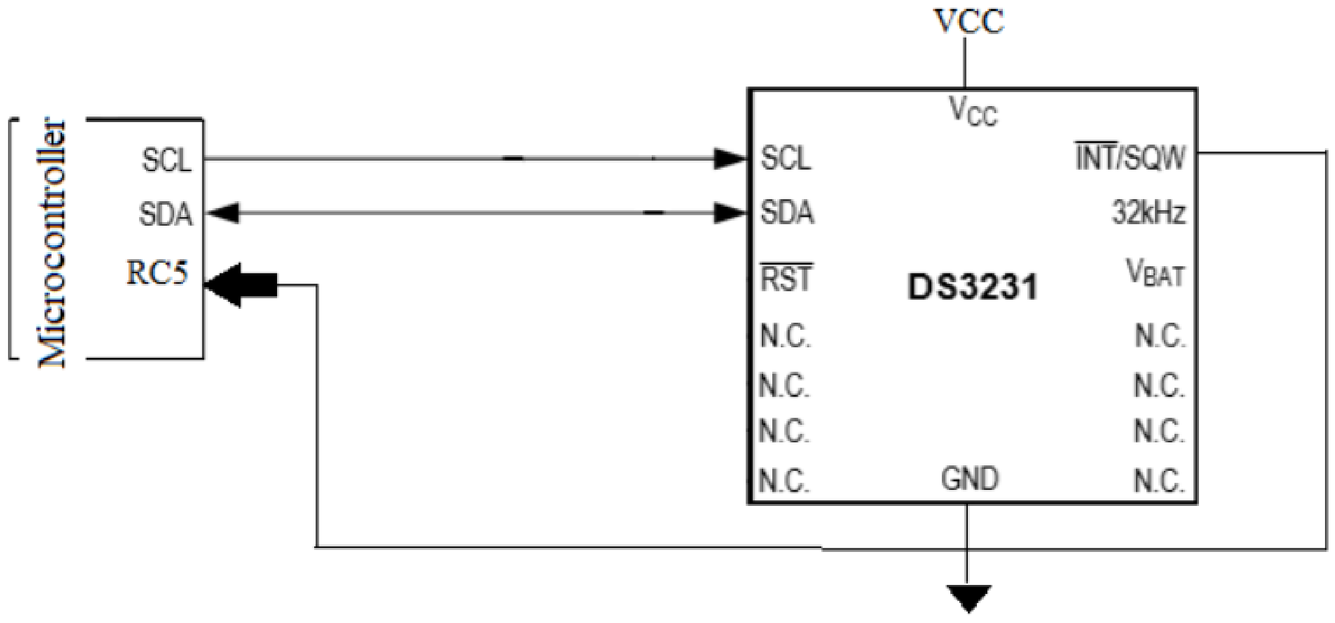
Image 6: LCD and microcontroller setup.



The microcontroller Port C is configured to be the output for the RTS. For the RTS, pin 1 is connected to VCC and pin 5 is connected to GND, to power up the RTS. The microcontroller RC4 pin, PPS SSP data pin, is connected to the RTC pin 2 which is the SDA pin meaning that data is read or written too. The microcontroller RC3 pin, PPS SSP clock pin, is connected to the

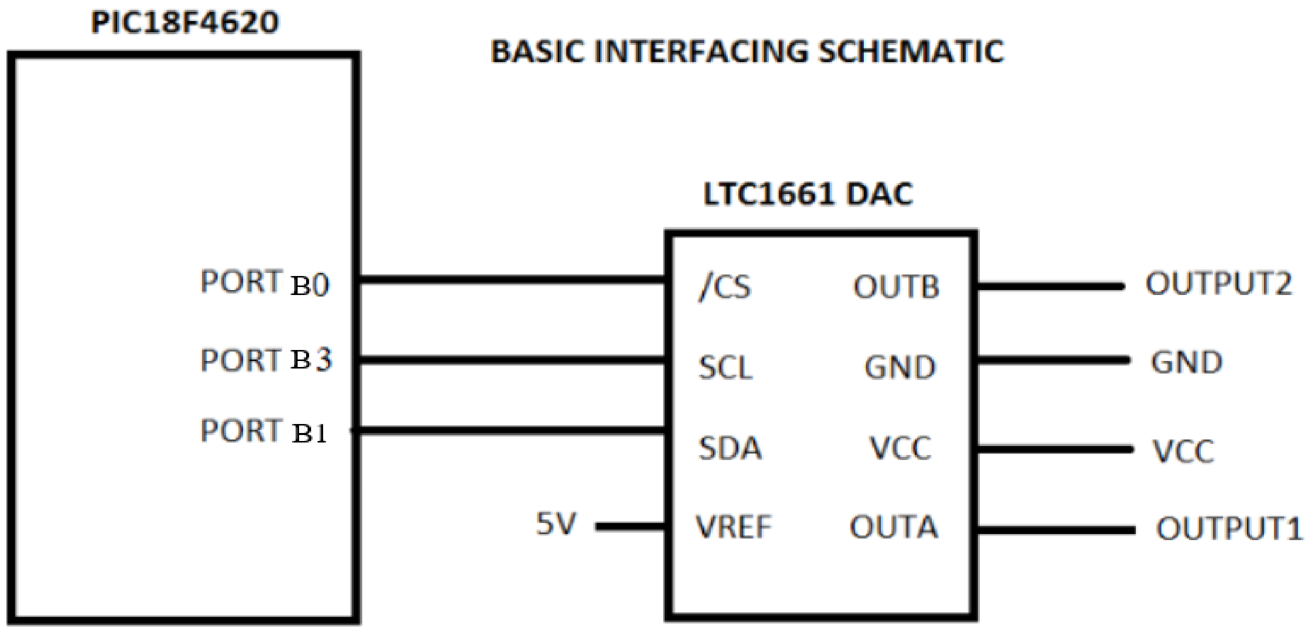
RTC pin 3 which is the SDC pin, clocking the RTC. The microcontroller RC5 pin, an input pin, is connected to the RTC pin 4 which is the modified interrupt pin which when the alarm goes off will output an active high signal. The microcontroller checks if a high signal from the interrupt pin is active if so, toggle the speaker pin to make an audible tone, else do nothing to the speaker. An image of the RTC wiring can be seen below.

Image 7: DS3231 RTC and microcontroller setup.



Depending on the selected channel, the microcontroller will send the data from a lookup table to the DAC to output a wave function. The microcontroller Port B is configured to be the output for the DAC. For the DAC, pin 6 is connected to VCC and pin 7 is connected to GND. VCC is connected to the DAC's reference voltage at pin 4, allowing for the tuning of the max voltage level of the wave function. The SPI chip select from RB0 output is wired to the DAC's chip select at pin 1, SPI clock from RB3 output is wired to the DAC's clock at pin 2, and SPI data from RB1 output is wired to the DAC's data input at pin 3. The master has control of the communication sends Clock and Select signals to the slave while the slave sends data through MISO and Master sends data out through MOSI. Timing synchronization is needed to effectively use SPI. This is important for the function spi_write() that to write a command to modify the DAC configuration and writes a wave function value to be displayed. The output from the DAC can be seen through pin 8 (output A) or pin 5 (output B). An image of the DAC wiring can be seen below.

Image 8: DAC and microcontroller setup.



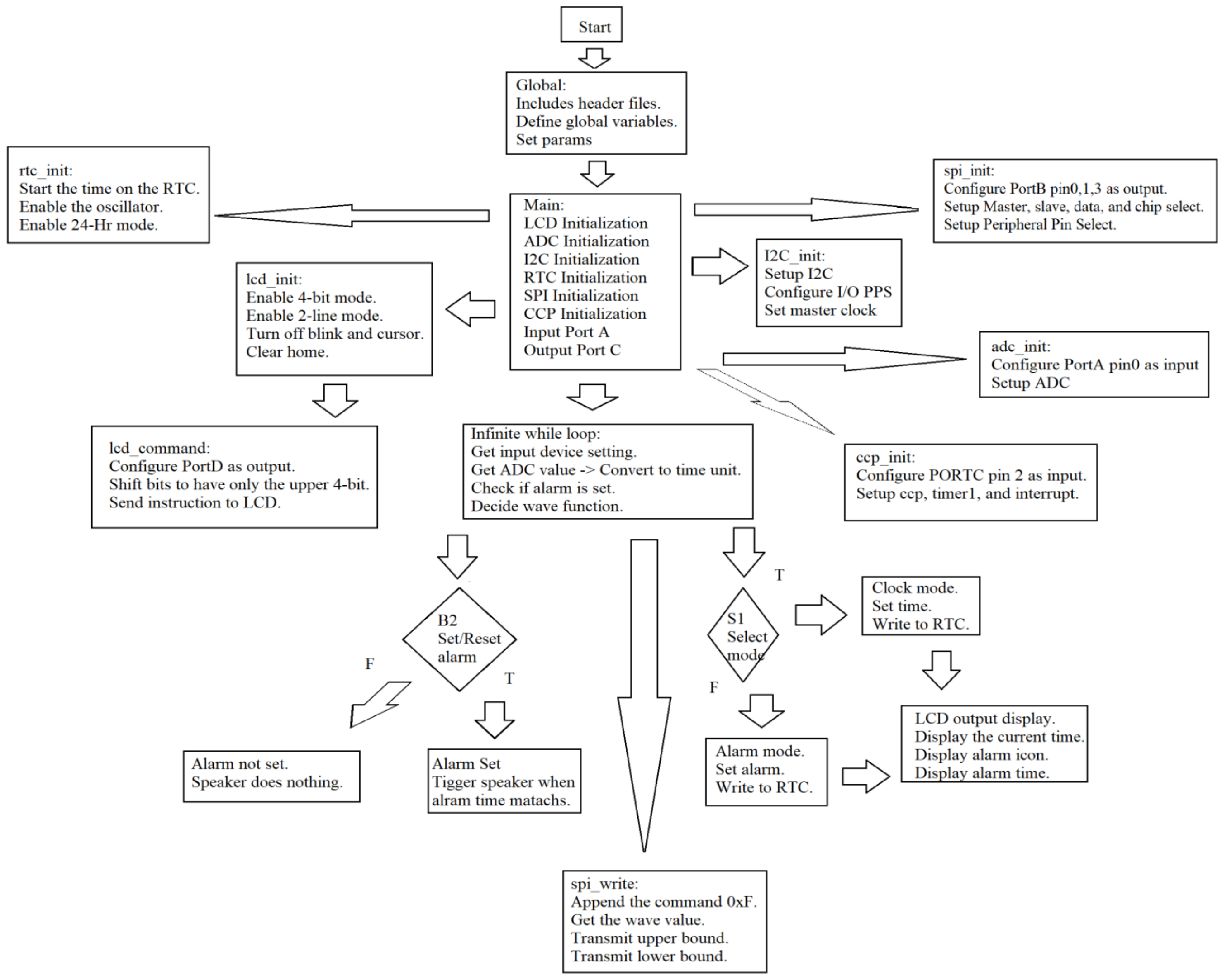
Due to the IR photodiode sensor's voltage being too low to be detected on the microcontroller, the IR signals are first sent to an op-amp that is ups the voltage to 5V through a 22x gain so that it can now be detected by the microcontroller. When an IR signal is received, the CCP interrupt flag is set which leads to a the ISR() to decode the long sync of the signal to a channel.

The flow of the program can be seen below in image 8. The software program began with including the header files: #include <xc.h> (Contain the PIC C commands), #include <stdio.h> (Contain I/O commands), and #include <stdlib.h> (Contain C command). Then came the definition of global variables, declaration of methods, disabling the watchdog timer, enable low voltage programming mode, and set the delay clock to 4MHz. The LCD functions, which include lcd_char(), lcd_command(), and lcd_init(), was already provided in the previous assignment guide and just copied with some minor adjustment to make it compatible to the new microcontroller.

The main method initializes the ADC, LCD, I^{2}C, RTC, SPI and CCP. Port A is enabled as input and port C is enable as output. Then it follows an infinite while loop where it checks the input setting to enter clock or alarm mode and obtain the ADC value which is then converted to a time value and written to the RTC set up the clock or alarm. The current time is then retrieved from the RTC to printed as an output to the LCD. Communication done with the RTC is under I2C protocols, to read or write the time and alarm. The microcontroller checks if a high signal from the interrupt pin is active if so, toggle the speaker pin to make an audible tone, else do nothing to the speaker. A button is used to set or reset the alarm after it has triggered. Based on the selected channel, the lookup table data is sent to the DAC to finally be converted to a wave. The wave function is checked to see if it is the correct output. When an IR signal is received, the

CCP interrupt flag is set which leads to a the ISR() to decode the long sync of the signal by taking the time for the first rising edge to the next falling edge as this is the pattern for the long sync. The time is then translated to a channel value. This process is then repeated forever

Image 9: Software flow chart.



Given functions:

The lcd_init() function will initialize the LCD screen into 4-bit mode, 2 line mode, turn off the blinking cursor, clear the screen, and place the cursor at address 0x00 (first position). The function takes no argument and does not return any data.

The lcd_char() function will write a character to the screen at the current cursor address. The function takes a character (8-bit number) as an argument. If the argument is an integer or a long (16 or 32 bit number), only the lower 8 bits will be used. The function does not return any data.

The lcd_command(char x) handles the writing to the LCD through PORTC. The function will write a command to the command register of the LCD screen. The function takes a character (8-bit number) as an argument. If the argument is an integer or a long (16 or 32 bit number), only the lower 8 bits will be used. The function does not return any data.

Note-RA6 and RA7 need to be configured in a special way because they share pins with external oscillators, so PORTA was abandon and PORTC was used instead.

Written functions:
The adc_init() function will initialize the ADC on PORTA pin 0, an input. The ADC configured to be left adjusted and disable for now.

The adcNum() function will enable ADC and get the ADC value through the ADRES with a right shift by 6 bits to correct bits for the upper 4-bit before returning.

The i2c_init() function will initialize I2Con PORTC pin 3-5. Setup the I2C master clock and the PPSI/O, I2C uses the same values as SPI.

The i2c_read() function will setup the I2C protocol to read data from a given address and register.

The i2c_write() function will setup the I2C protocol to write data to a given address, register, and data.

The spi_init() function will Configures SPI2, using PortB for the DAC LTC1661. The clock rate used for the SPI output is FOSC/64.

The spi_write() function will transmit the data to the DAC. The 0xF command will append to the data value from the calculation to wave made with the frequency. The entire 16-bit output is split between two bye output with the high bound being transmitted first followed by the lower bound. The transmission is data is stored in the SPI buffer until it is read to the DAC

The rtc_init() function will start the time on the RTC.

Numerous getters and setters method to manipulate the clock and alarm time; hours, minutes, seconds, day, etc.

The ccp_init() function will initialize the CCP and timer1 on PORTC pin 2, an input. The CCP configured to trigger on every edge and the timer running at freq/4.

The __interrupt() ISR() function will enable CCP and get the time value through the CCPR1 and convert it to a channel.

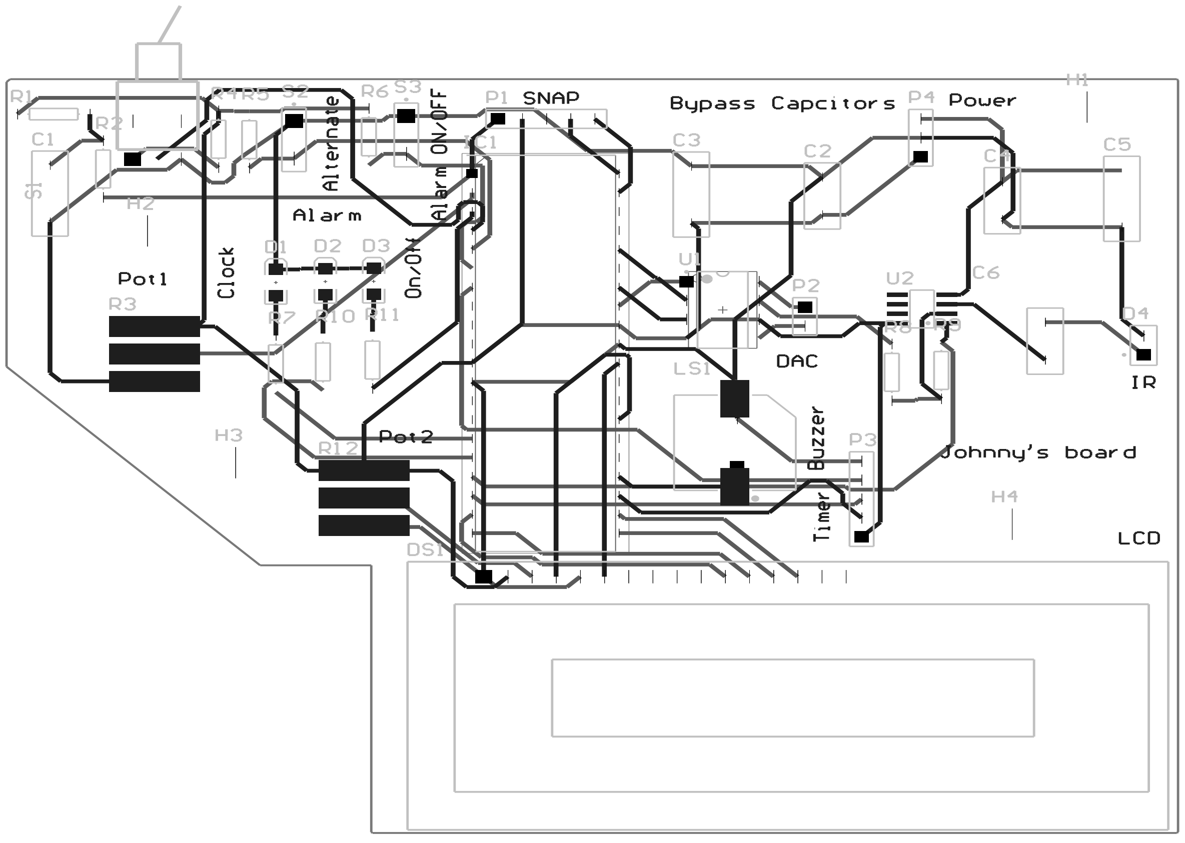
The getr() returns the current channel.

Formula used to calculate the timevalue:
24-Hour unit: u24= ADC value/42
60-Minute/Second unit: u60= ADC value/17

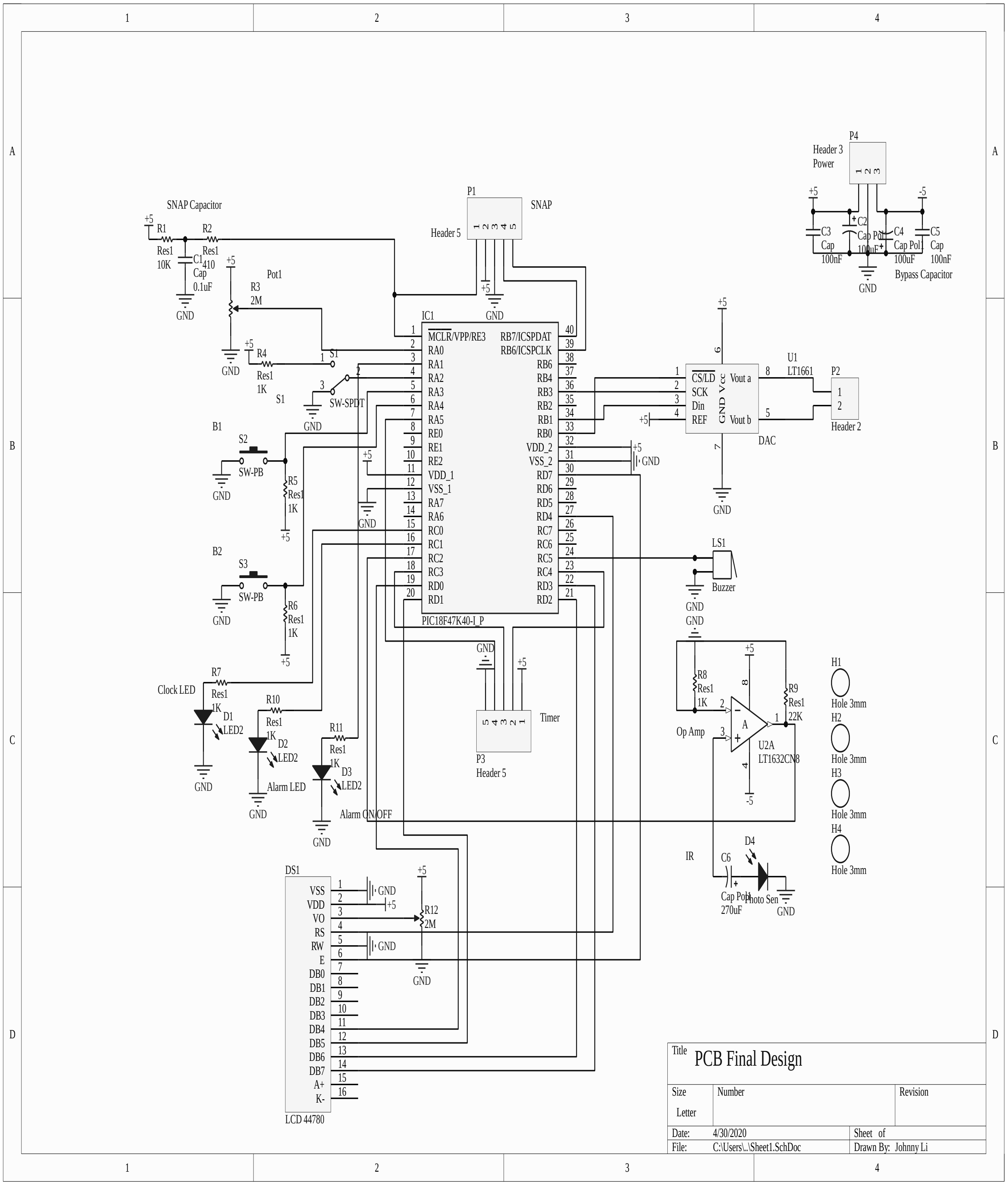
Formula used to calculate the ADC value:
ADC value = (Resolution of the ADC/System Voltage) * (Analog Voltage Measured) = (1023/5V) * Vout

Altium Designer 20 was used to create a schematic and generate a printed circuit board as shown below. The design schematic will be the schematic designed on Altium Designer 20 and converted to a PCB. All I/O components mentioned on the original schematic have their respective components built in the design schematic including the power source, input, and output. To symbolize the power source an extra component of a 3-pin header with +/-5V and ground was attached to the design. Since the IR photodiode of the original schematic had no identification, a generic sensor diode was used from the libraries in Altium. The generic components from the libraries were chosen for the remaining resistors, polarized capacitors, holes and diodes. Different component footprints were used in the instances of polarized capacitors and switches to make the PCD design more realistic. This can be seen in Schematic 2.

Image 10: PCB layout drawing.



Schematic 2: Altium schematic drawing.


SNAP Capacitor
R1 Res1 10K
R2 Res1 410
C1 Cap 0.1uF
Pot1 R3 2M
R4 Res1 1K
S1 SW-SPDT
B1 S2 SW-PB
R5 Res1 1K
B2 S3 SW-PB
R6 Res1 1K
R7 Res1 1K
Clock LED D1 LED2
R10 Res1 1K D2 LED2 Alarm LED
R11 Res1 1K D3 LED2 Alarm ON/OFF
P1 Header 5 SNAP
IC1 PIC18F47K40-I_P
MCLR/VPP/RE3 RA0 RA1 RA2 RA3 RA4 RA5 RE0 RE1 RE2 VDD_1 VSS_1 RA7 RA6 RC0 RC1 RC2 RC3 RD0 RD1
RB7/ICSPDAT RB6/ICSPCLK RB6 RB4 RB3 RB2 RB1 RB0 VDD_2 VSS_2 RD7 RD6 RD5 RD4 RC7 RC6 RC5 RC4 RD3 RD2
P3 Header 5 Timer
P4 Header 3 Power
C3 Cap 100nF
C2 Cap Pol1 100uF
C4 Cap Pol1 100uF
C5 Cap 100nF
Bypass Capacitor
U1 LT1661 DAC
CS/LD SCK Din REF Vout a Vout b GND Vcc
P2 Header 2
LS1 Buzzer
R8 Res1 1K
R9 Res1 22K
Op Amp U2A LT1632CN8
IR C6 Cap Pol1 270uF D4 Photo Sen
H1 Hole 3mm H2 Hole 3mm H3 Hole 3mm H4 Hole 3mm
DS1 LCD 44780
VSS VDD VO RS RW E DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7 A+ K-
R12 2M
GND +5 -5

Title	PCB Final Design	
Size: Letter	Number	Revision
Date: 4/30/2020		Sheet of
File: C:\Users\..\Sheet1.SchDoc		Drawn By: Johnny Li

To test the microcontroller, the RTC's interrupt active high signal for the alarm was checked with the digital multimeter to confirm that the RTC was configured correctly. To test the LCD and RTC, the displayed value was checked to see if it was able to display properly and had the correct time value. Alarm time was set to be triggered to ensure that the speaker went off the property. To test the DAC output was checked with the digital oscilloscope to confirm the proper wave value was obtained. The channel was varied, and the displayed wave was checked to see if it was able to properly change. The IR photodiode was connected to the digital oscilloscope to identify that a signal was received and to check if the decoding process was correct. The results are shown below.

Image 11: Alarm mode

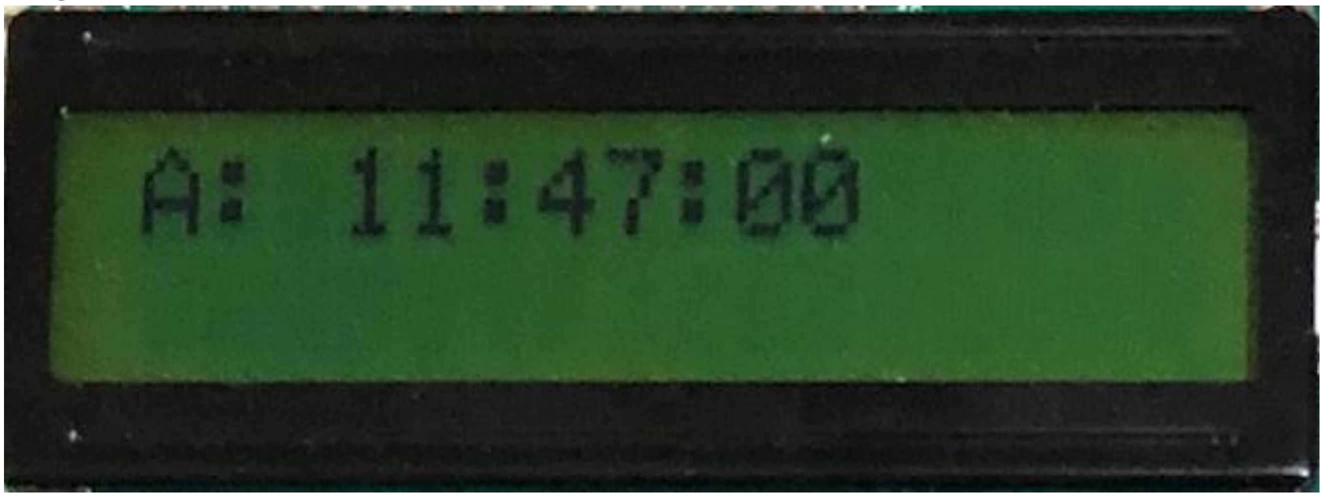
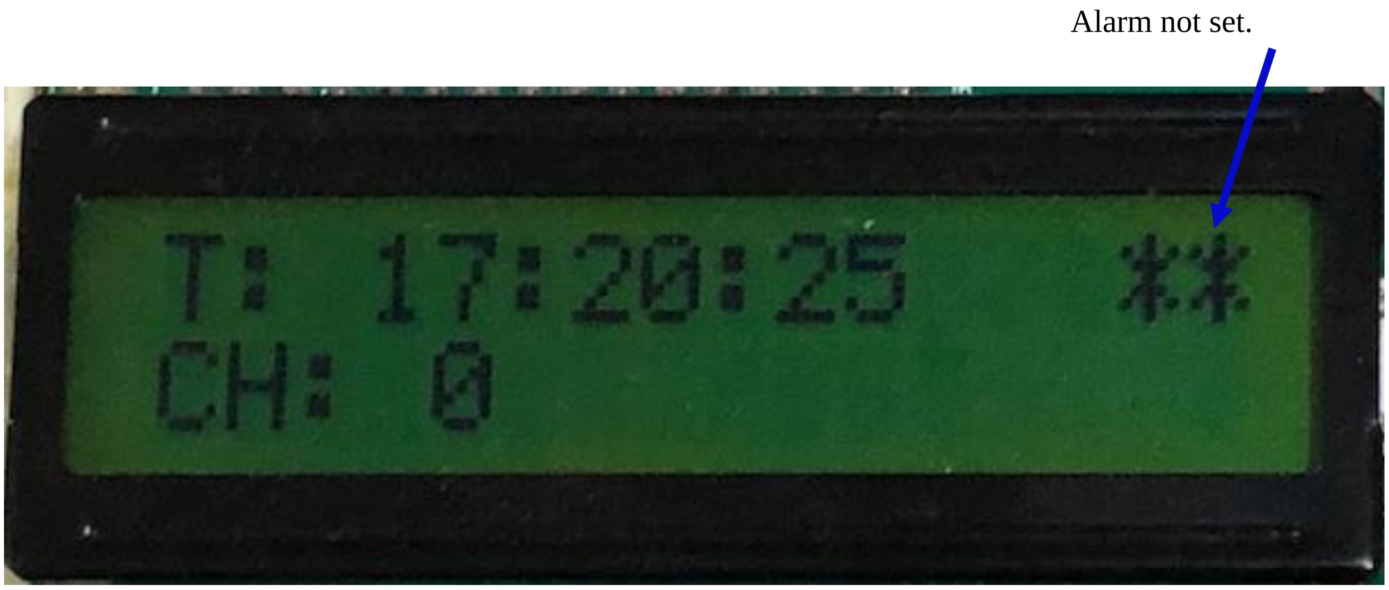


Image 12: Clock mode with channel 0.



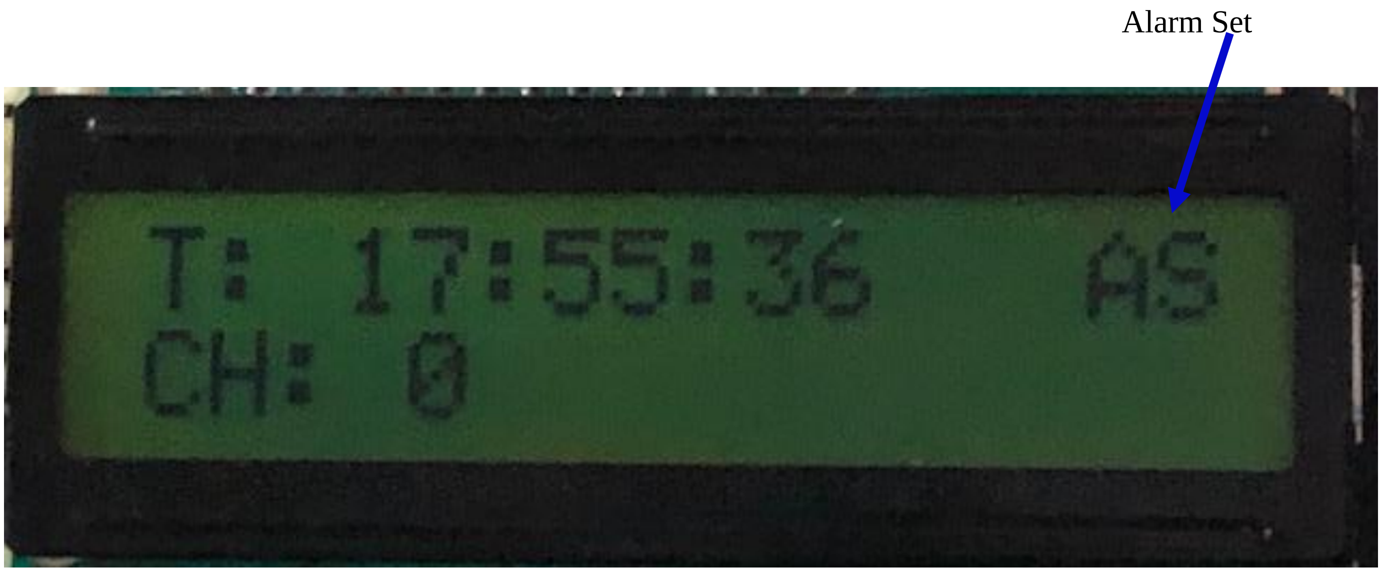


Image 13: DAC channel 0.

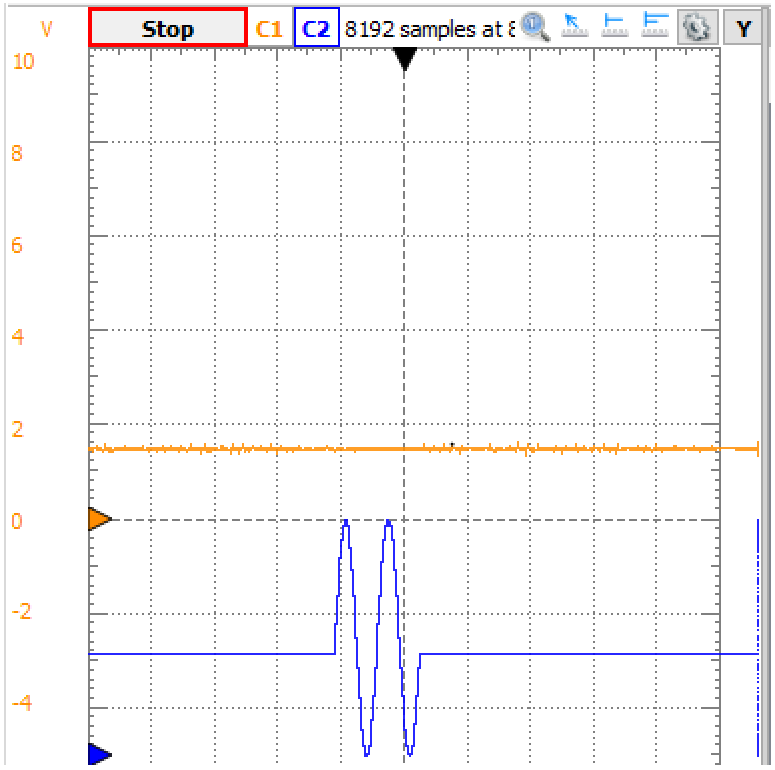


Image 14: Clock mode with channel 1.

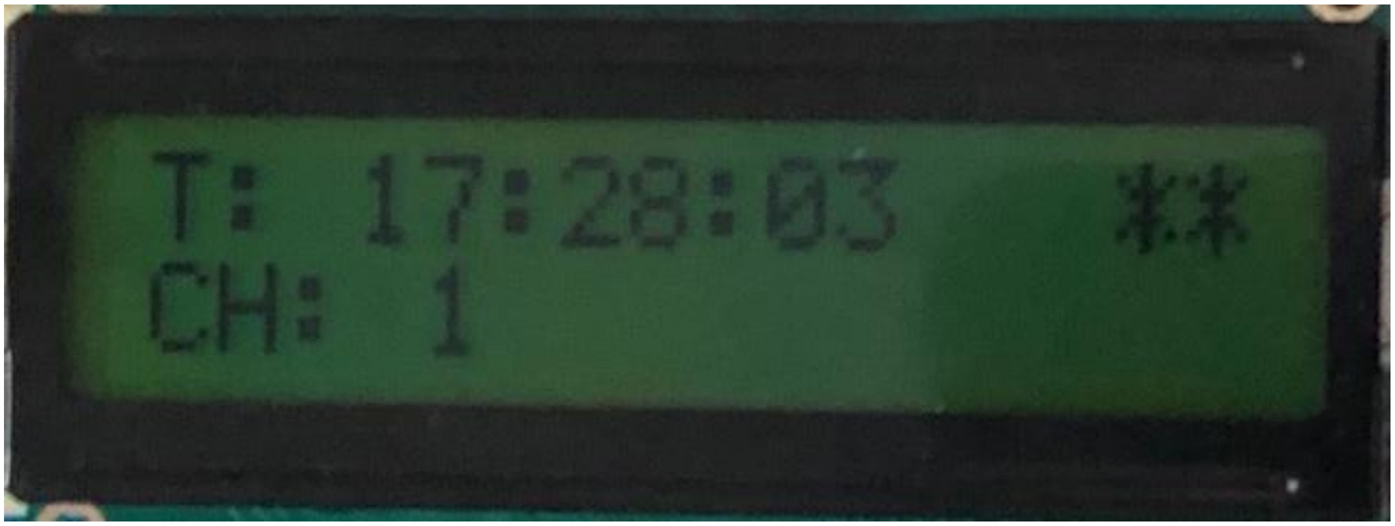


Image 15: DAC channel 1.

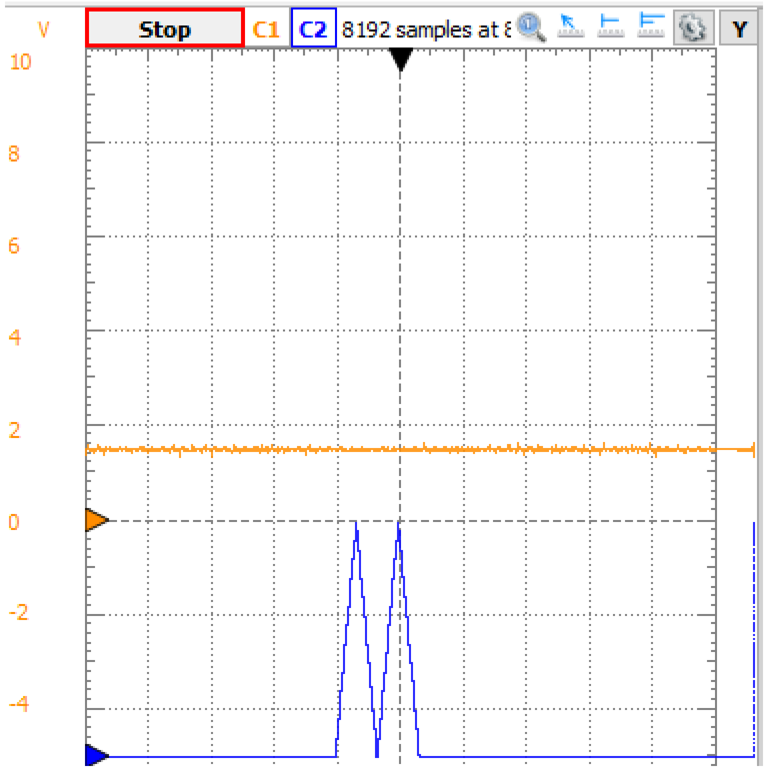


Image 16: Clock mode with channel 2.

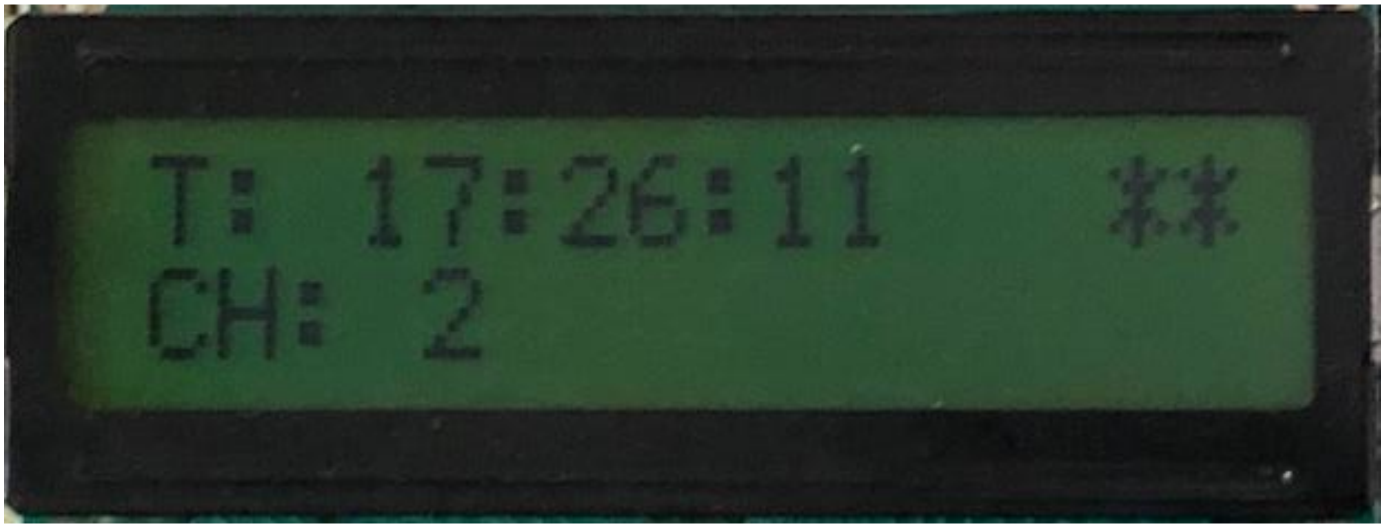
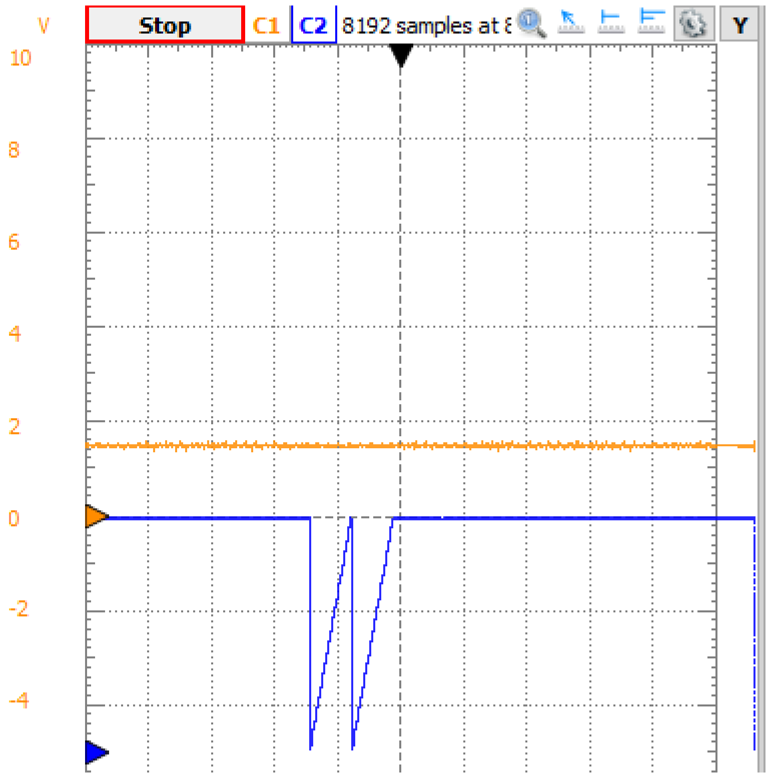


Image 17: DAC channel 2.



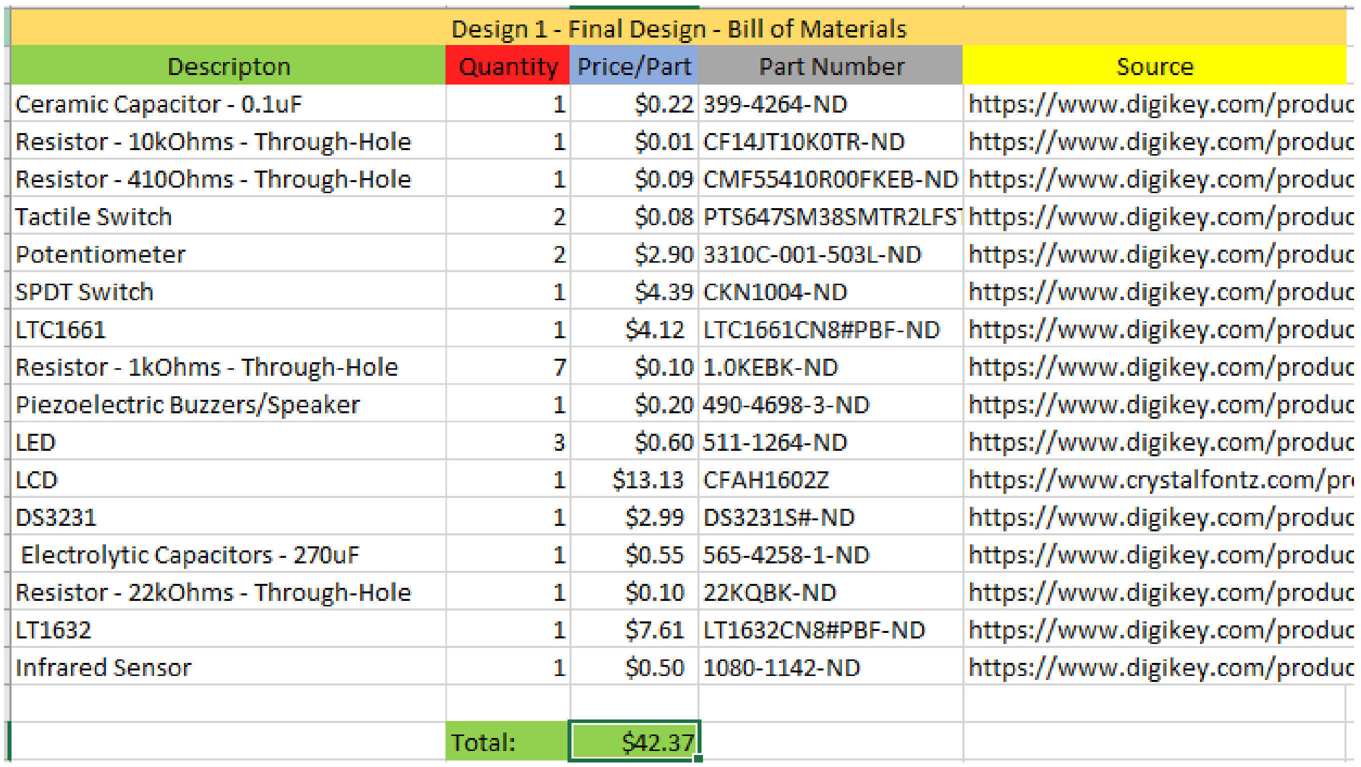
An interesting thing that had to be done for the microcontroller designs to work correctly was to swap the LCD output from Port A to Port D. The new microcontroller's RA6 and RA7 are configured pins with external oscillators so PORTA and would not function the same as with the old microcontroller. Rather than code some more or recalculate the need bit setup the simplest things were to abandon Port A and use Port D instead. Also, the delay function from the given code was not usable in the new microcontroller so it was replaced with __delay_ms(5). A problem found during the design of the microcontroller was that the RTC interrupt pin was not initially physically connected to the output pins available for it, so a wire had to be soldered to connect the interrupt pin to the available output pin to get the RTC working.

Another thing that had to be done was to swap the DAC output from Port D to Port B. The microcontroller’s RB3 and RB1 are configured pins with PPS so PORTD and would not function without remapping the pin of the PPS. Rather than code some more or rethink the pins set up the simplest things was to abandon Port D and use Port B instead. A problem found during the design was that the CCP period value would continuously vary. This took a while to figure out and require testing of several microcontroller signal setting to confirm that the issue existed.

For the PCB to be created correctly, there were some changes in the components’ footprint, for example, the default capacitors were too large, and the switch had an irregular design. Pour designs were laid out on both the top and bottom of the board. A problem found during the design of the PCB was that repeated reconfiguration had to be done to the PCB file’s setting as the connection lines and position errors, green highlights, did not function as intended. No error of positioning would show till attempting to fabricate the output of the Geber files. The solution was mainly spacing the components out more and resetting the PCB file. The mounting holes for the board were placed unevenly since there was a lack of room after the keep-out layer phase had occurred.

The initial idea for this project was to be able to decode the IR signals for each numbered button, 0-9, but this turned out to be much more difficult than anticipated thus the idea was discarded and replaced with identifying the remote through the long sync.

The bill of materials, excluding the microcontroller and SNAP programmer, is shown below:



Design 1 - Final Design - Bill of Materials				
Descripton	Quantity	Price/Part	Part Number	Source
Ceramic Capacitor - 0.1uF	1	$0.22	399-4264-ND	https://www.digikey.com/produc
Resistor - 10kOhms - Through-Hole	1	$0.01	CF14JT10K0TR-ND	https://www.digikey.com/produc
Resistor - 410Ohms - Through-Hole	1	$0.09	CMF55410R00FKEB-ND	https://www.digikey.com/produc
Tactile Switch	2	$0.08	PTS647SM38SMTR2LFS	https://www.digikey.com/produc
Potentiometer	2	$2.90	3310C-001-503L-ND	https://www.digikey.com/produc
SPDT Switch	1	$4.39	CKN1004-ND	https://www.digikey.com/produc
LTC1661	1	$4.12	LTC1661CN8#PBF-ND	https://www.digikey.com/produc
Resistor - 1kOhms - Through-Hole	7	$0.10	1.0KEBK-ND	https://www.digikey.com/produc
Piezoelectric Buzzers/Speaker	1	$0.20	490-4698-3-ND	https://www.digikey.com/produc
LED	3	$0.60	511-1264-ND	https://www.digikey.com/produc
LCD	1	$13.13	CFAH1602Z	https://www.crystalfontz.com/pr
DS3231	1	$2.99	DS3231S#-ND	https://www.digikey.com/produc
Electrolytic Capacitors - 270uF	1	$0.55	565-4258-1-ND	https://www.digikey.com/produc
Resistor - 22kOhms - Through-Hole	1	$0.10	22KQBK-ND	https://www.digikey.com/produc
LT1632	1	$7.61	LT1632CN8#PBF-ND	https://www.digikey.com/produc
Infrared Sensor	1	$0.50	1080-1142-ND	https://www.digikey.com/produc
	Total:	$42.37		

Conclusion

The specifications for the final design was mimic an infrared (IR) remote receiver for multiple remotes. The microcontroller, PIC18F47K40, is used to process inputs from different IR remotes to change the "channels" being displayed on the LCD. Using a photodiode to detect incoming IR signals and the capture mode on the microcontroller to read the signal. There will be an I^{2}C timer button to set the time and alarm with LEDs to indicate a unit change. The time and alarm mode will be controlled by a digital input from a switch and a button to enable or disable the alarm. A potentiometer will be used to alter the voltage value to the microcontroller ADC to select the time values. The time, channel, and alarm status will be displayed on the LCD. If the alarm has triggered, a buzzer will go off till the alarm is disabled. An external DAC using an SPI interface will be used to drive a sin, triangle, or sawtooth wave relative to which channel is on.

In image 12, the timer is in clock mode and is displaying the current time with all the required units of time without having the alarm being set as shown with **, symbolizing that the alarm is not set yet. In image 11a, the timer is in alarm mode and is displaying the set alarm time trigger the speaker when it matches the time. The time value was able to change due to a potentiometer that alter the voltage value to the microcontroller which select the time values. As seen in image 11b the timer is in clock mode and is displaying the current time with all the required units of time with the alarm being set as shown with AS, symbolizing that the alarm is set and thus meeting the LCD and I^{2}C specification. The IR photodiode was able to detect the IR signals and the microcontroller was able to successfully decode the signals to a channel. With this seen in image 13,15, and 17 the result of the DAC output through SPI generated wave functions at a 5 V based on the respected channels, meeting the DAC specification. Digital inputs were meet with switches and buttons while digital outputs were meet through LEDs. The specifications for the design of the PCB in reference to the original schematic were met. All components, or closely related part, in the original schematic were placed in a similar fashion in the design schematic. The design schematic was effectively transformed to a PCB design error free and the fabrication output came out correct. The design was able to meet all the required specifications.

Some problems found during the design include that the CCP period value would continuously vary. This took a while to figure out and require testing of several microcontroller signal setting to confirm that the issue existed. Also, the design of the microcontroller was that the RTC interrupt pin was not initially physically connected to the output pins available for it, so a wire had to be soldered to connect the interrupt pin to the available output pin to get the RTC working. In addition, the design of the PCB was that repeated reconfiguration had to be done to the PCB file's setting as the connection lines and position errors, green highlights, did not function as intended. No error of positioning would show till attempting to fabricate the output of the Geber files. The solution was mainly spacing the components out more and resetting the PCB file. The mounting holes for the board were placed unevenly since there was a lack of room after the keep-out layer phase had occurred.