Laboratory 07 – Analog

By: **Caleb Probst**

Spring 2023 ECE 381 Microcontrollers

Introduction

For this laboratory, the main idea of the lab is to learn about the ADC and DAC in the PSoC4. An ADC (Analog-Digital Converter), the basic operation is it takes a voltage between two references and outputs a digital value to a register. This allows the microcontroller to input analog voltages, in between two reference points. A DAC (Digital-Analog Converter) does the opposite. The DAC takes a digital value from a register and outputs an analog signal (in the case of the PSoC4, it is a current, as the PSoC4 only has an IDAC). This allows the microcontroller to output analog voltages/currents.

Goal

There are two parts to this lab. The goal of the first part is to simply calibrate the DAC and make a graph mapping digital values to voltages. The goal of the second part is to make a display for a temperature sensor and a potentiometer, and control a set of LED's based on these inputs.

Materials

PsoC4 4200M Microcontroller	1
MCP9701 Temperature Sensor	1
Potentiometer	1
Hitachi LCD	1
100µF Capacitor	1
2.2KΩ Resistor	3

Procedure and Results

Part 1 – Setup in PSoC Creator

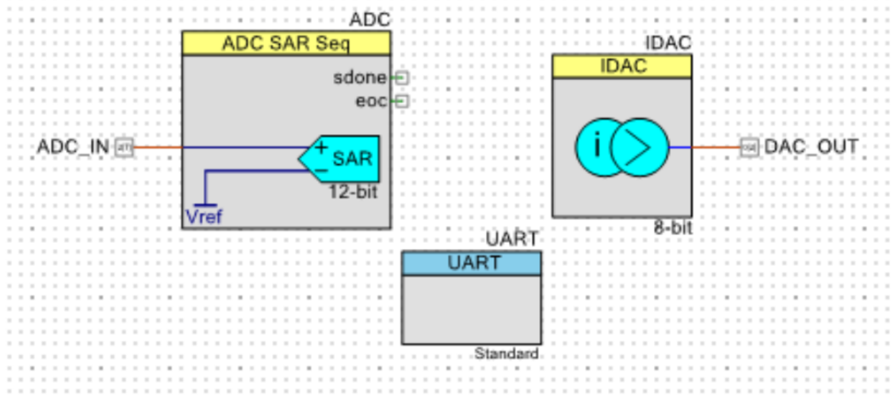


Figure 1: Part 1 Top Design

For part 1, all we are doing is simply calibrating the ADC, so we will have an ADC module and a IDAC module. We will be transmitting this data through UART, so we will set that up as well. For the ADC, we want to configure it in the following way: set Vref to “Internal, 1.024V, bypassed” and “Single ended negative input” to Vref. You should see that “Single ended mode range” should be 0.0 to 2*Vref. Also set the “Single ended result format” to unsigned. In the channels tab, setup 1 channel with 12-bit resolution in “Single” mode. For the IDAC, ensure that the polarity is Positive so it will source current, and the resolution is 8-bit. Then, setup some analog input pins on the ADC and IDAC. Below is the pins for this part of the lab.

ADC:Bypass	P1[7]
UART:rx	P7[0]
UART:tx	P7[1]
ADC_IN	P2[7]
DAC_OUT	P0[2]

Table 1: Part 1 Pin Outs of Hardware on PSoC4

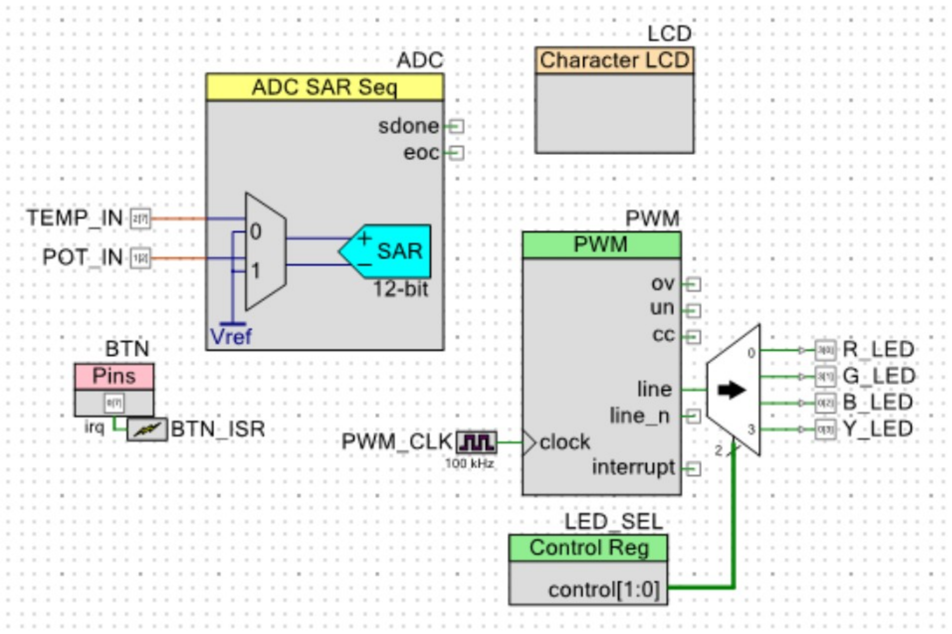


Figure 2: Part 2 Top Design

For part 2, we want a similar setup, but without the DAC. For the ADC setup, it will be setup in the same fashion as part 1, but with 2 channels instead of 1. The rest of the setup is the same as previous labs. For the BTN, set it up for resistive pull up interrupt on a falling edge. For the PWM, set the clock to 100kHz and use the control register to control the multiplexer for the LED's. Below is the pins for this part of the lab.

ADC:Bypass	P1[7]
LCD	P2[6:0]
BTN	P0[7]
TEMP_IN	P2[7]
POT_IN	P1[2]
R_LED	P3[0]
B_LED	P0[2]
G_LED	P3[1]
Y_LED	P0[3]

Table 2: Part 2 Pin Outs of Hardware on PSoC4

Part 2 – Hardware and Wiring

For part 1 of the laboratory, it was a very simple setup with the IDAC and ADC. We simply set the IDAC up to send current through the 3 series resistors, and have the ADC measure the voltage from that current. Shown by *Figure 3* is the circuit we built for this system.

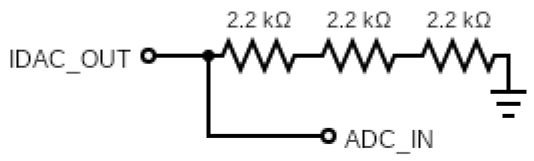


Figure 3: Part 1 Circuit Diagram

For part 2 of the laboratory, we simply set up the potentiometer and temperature sensor as inputs to the PSoC. The wiring diagram for this is shown in *Figure 4*.

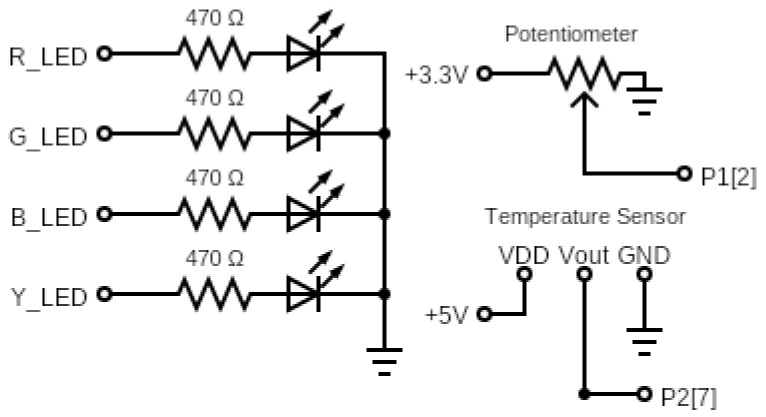


Figure 4: Part 2 Circuit Diagram

Part 3 – Software

For part 1 of the laboratory, we simply started the modules and recorded the results through UART. Then, once we got these values, we calculated the voltage of the DV’s that we were given from the ADC. Here is the graph we calculated that maps digital ADC values to Voltage, shown by *Figure 5*.

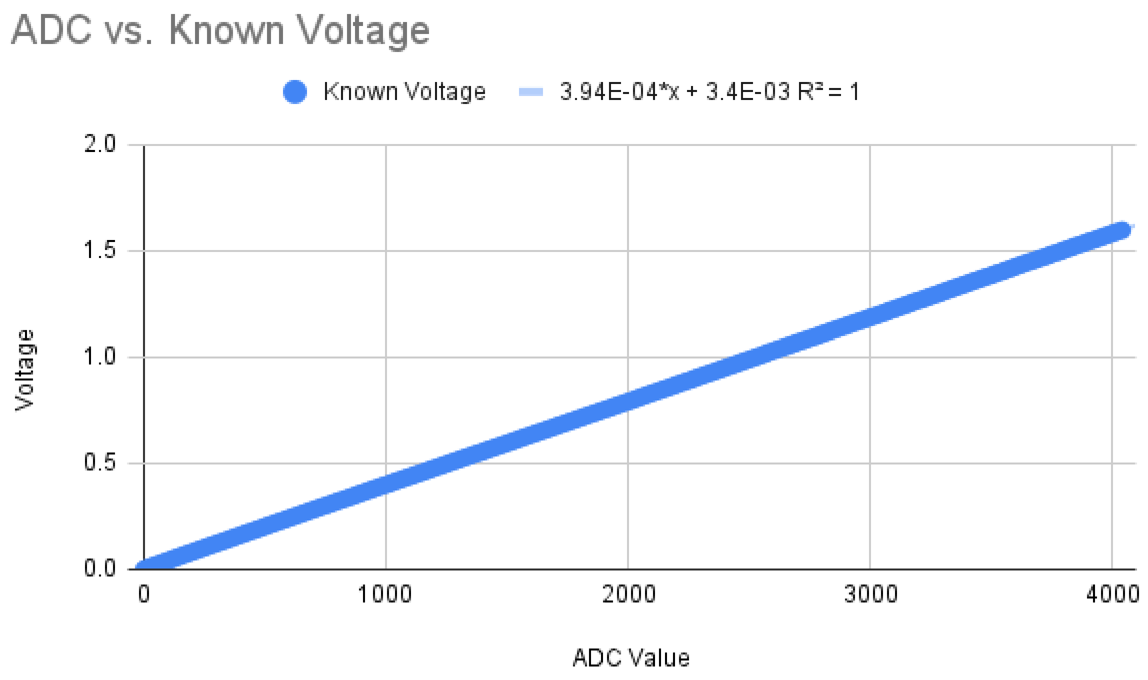


Figure 5: Graph Mapping for ADC Value Vs. Voltage

The way these were calculated was given the digital value of the IDAC (0x00 through 0xFF), each step was 1.2µA, so we multiplied each value by the resistance (5.237KΩ [3 series 2.2KΩ]) to get the voltage. Then we mapped this to the ADC value that was returned.

For part 2, we took the general concept of the ADC and used it with the temperature sensor and the potentiometer, as both of them return an analog voltage. The flowchart for this program is shown by *Figure 6*.

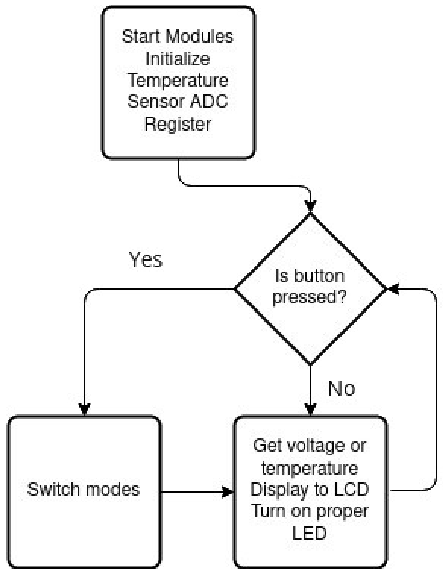

Start Modules Initialize Temperature Sensor ADC Register
Yes
Is button pressed?
No
Switch modes
Get voltage or temperature Display to LCD Turn on proper LED

Figure 6: Part 2 Flow Chart

The software for this portion of the laboratory was extremely simple. First, we started the modules, which included the ADC, LCD, PWM and ISR. Then, we set up the ADC to read the temperature value. We did this by bitmasking the *ADC_SAR_CTRL_REG* to modify bits 7-4. We wanted these bits to be 1100 (0xC), so we first cleared them then set them. When the button is pressed, we simply do the same thing, except set bits 7-4 to 1110 (0xE). This bitmasking allows us to use an absolute voltage reference (for temperature sensor) vs a relative voltage reference (for potentiometer). To get the temperature from the temperature sensor, we simply read the digital value from the ADC and convert it into degrees. We did this in two steps. First, we turned it into a voltage, using *Equation 1*.

$$V_{out} = DV * \frac{2.048}{4095}$$

Equation 1: Voltage Conversion Using Absolute Reference

Then, to convert it to a temperature in Celsius, we used the equation found in the datasheet for the temperature sensor, which is given by *Equation 2*.

$$T_C = \frac{V_{out} - .400}{.0195}$$

Equation 2: Temperature Conversion Using Voltage

Now that we have the temperature in Celsius, we can now convert it into Fahrenheit and display both of them. So, we simply use snprintf() to display them on the LCD. Then, we select the proper LED based on the temperature given. To do this, we simply used a function that took in the temperature in Fahrenheit and selected the LED. So, we set up an extended-if statement to pick which range it was in, then turn on the LED with proper brightness. The brightness can be done by using the following equation, which works for any range (can work in voltage mode too).

$$\text{Duty Cycle} = \text{Period} * \frac{x - x_{min}}{x_{max} - x_{min}}$$

Equation 3: Duty Cycle Calculation

In the case of temperature, *x* is the temperature in Fahrenheit and *x_min* and *x_max* are the ranges where the LED is lit up at. The ranges at which each should be turned on is given by *Table 3*.

LED Color	**T_min**	**T_max**
Blue	0°F	75°F
Green	75°F	80°F
Yellow	80°F	85°F
Red	85°F	212°F

Table 3: LED Color Based on Temperature

In voltage mode, we do a very similar process, but with a relative voltage instead of absolute. To get the voltage, we used *Equation 4*.

$$V_{out} = DV * \frac{3.3}{4095}$$

Equation 4: Voltage Conversion Using Relative Reference

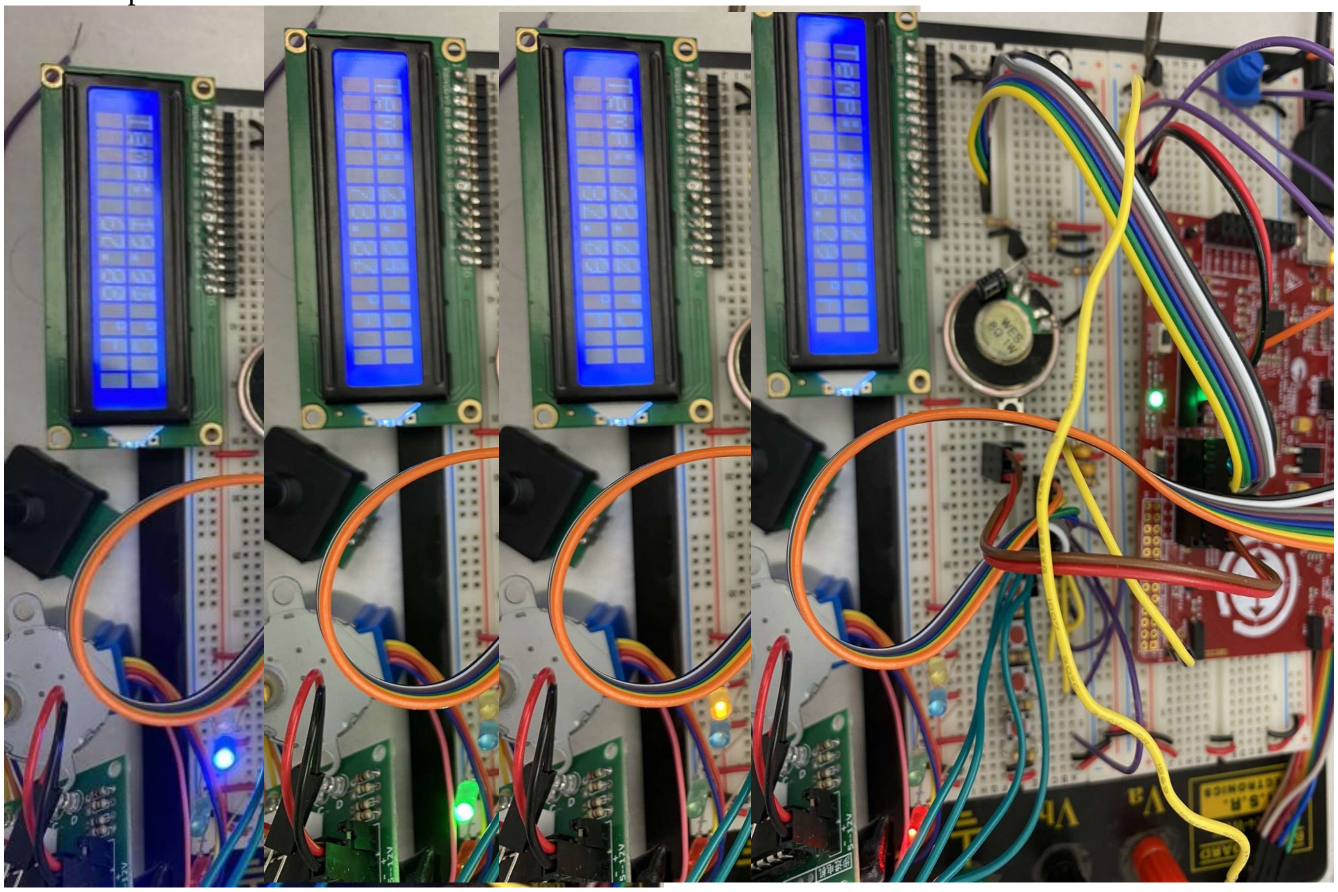
Once we have the voltage, we simply displayed it to the LCD and used a similar function to turn on the proper LED with proper brightness. We can reuse *Equation 3* with *x* being the voltage, *x_min* being the lower bound and *x_max* being the upper bound. The bounds for these are given by *Table 4*.

LED Color	**V_min**	**V_max**
Blue	0V	0.8V
Green	0.8V	1.6V
Yellow	1.6V	2.4V
Red	2.4V	3.3V

Table 4: LED Color Based on Voltage

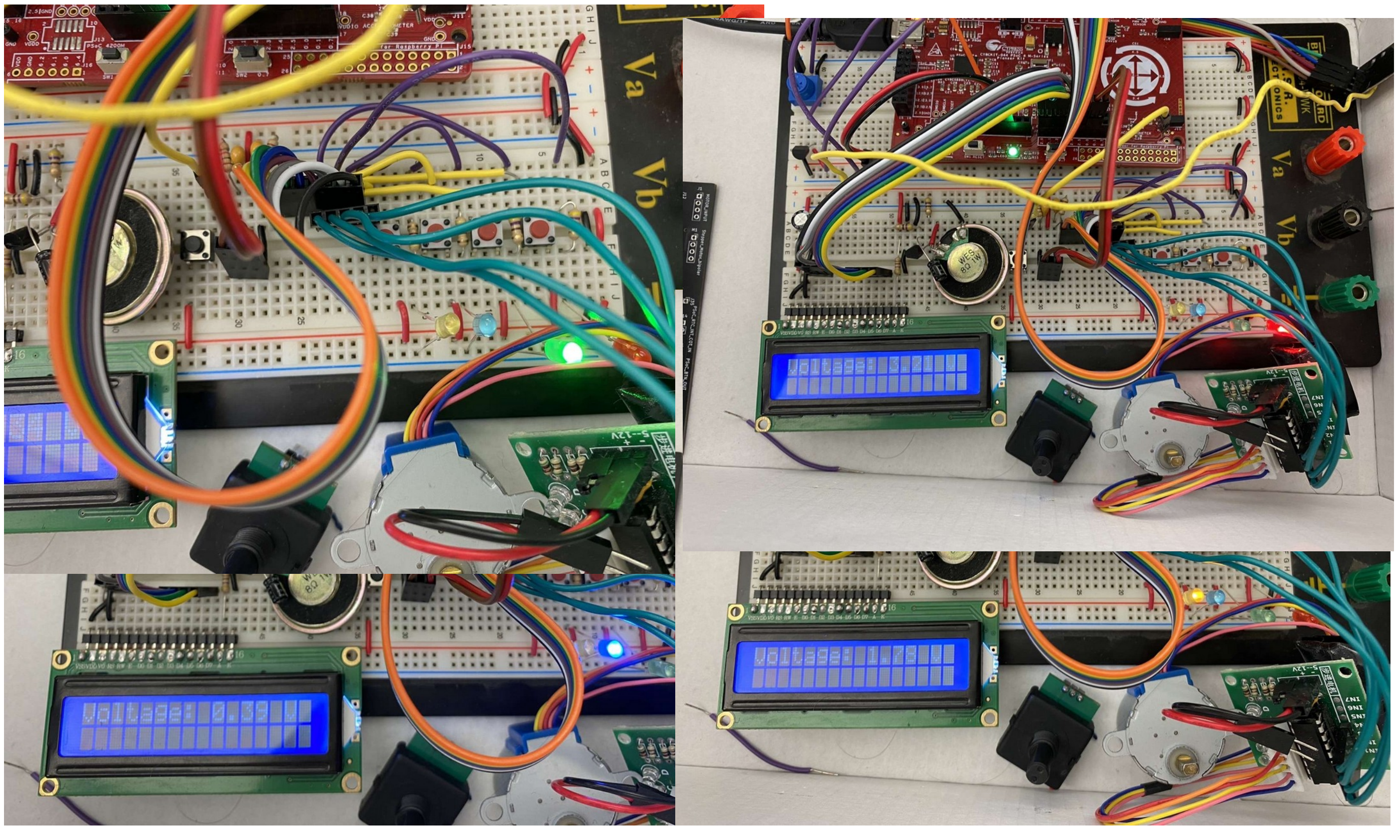
Testing Methodology

For testing this device, we needed a soldering iron. Reason for that, is we simply just need a heat source that can get very hot, very fast, and a soldering iron is the best method that we had in lab for testing. When testing the temperature, simply use the soldering iron on the tip of the temperature sensor and wait. *Figure 7-10* show the stages at which the LED and LCD change with temperature.



Figures 7-10: LCD and LED Temperature Updates

For testing the voltage, simply press the button on the PSoC, and turn the potentiometer. This is shown by *Figures 11-14*.


Figures 11-14: LCD and LED Voltage Updates

Conclusion

This laboratory was relatively simple once we figured out the ADC. From there, it was as simple as outputting a certain output to a LCD and switching modes. One issue that we had ran into was the potentiometer would effect the voltage reading on the temperature sensor if turned. We ended up figuring out that this was because the ADC is *really* bad at sequencing. The best way to fix this is to actually average the values in between the channels instead of getting the channel itself. However, this takes significantly longer and can mess with the refresh rate of the LCD. Since this is more of an issue of faulty hardware on the PSoC, we did not worry about it for this laboratory.

Appendix

Part 1 Main Code

```c
#include "project.h"
#include <stdio.h>

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g.
MyInst_Start()) */
    ADC_Start();
    IDAC_Start();
    UART_Start();
    ADC_StartConvert();
    char message[32];
    for(int i = 0x00; i < 0xFF; i++){
        IDAC_SetValue(i);
        CyDelay(10);
        ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
        snprintf(message, 32, "%d,%d\n",i,ADC_GetResult16(0));
        message[31] = '\0';
        UART_UartPutString(message);
    }
    for(;;)
    {
        /* Place your application code here. */
    }
}
```

Part 2 Main Code

```c
#include "project.h"
#include <stdio.h>

// Global variables for stuff
uint8 mode = 0;
volatile int buttonFlag;

/*
This function simply initializes temperature readings.

Init mate?
*/
void tempInit/*Mate*/(){
    // Set 6:4 to 100 and 7 to 1
    ADC_SAR_CTRL_REG = (ADC_SAR_CTRL_REG & 0xFFFFFF0F) | 0x000000C0;
    // Set input to 0
    //ADC_SetChanMask(0x0000);
}
/*
This function simply initializes voltage readings.
```

```c
Init mate?
*/
void potInit/*Mate*/(){
	// Set 6:4 to 100 and 7 to 1
	//ADC_Stop();
	ADC_SAR_CTRL_REG = (ADC_SAR_CTRL_REG & 0xFFFFFF0F) | 0x000000E0;
	//ADC_Start();
	// Set input to 0
	//ADC_SetChanMask(0x0001);
}
/*
This function gets the temperature value from the ADC.

It returns the value in degrees celsius
*/
float getTempValue(){
	// Variable for voltage
	float Vout;
	// Starts the conversion in the ADC
	ADC_StartConvert();
	// Wait for ADC to stop
	ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
	// Get result from channel 0
	int out = ADC_GetResult16(0);
	// Stop the ADC, saves power
	ADC_StopConvert();
	// Convert digital to voltage
	Vout = out * (2.048/4095);

	// (Vout - V0)/Tc = Ta
	return (Vout - .400)/(.0195);
}
/*
This converts value in celsius to value in farenheit

Returns value in farenheit
*/
float convertCtoF(float C){
	return (9/((float)5))*C + 32;
}
/*
This function selects the LED based on the temperature.

Blue for 75°F and under
Green for 75°F-80°F
Yellow for 80°F-85°F
Red for 85°F-212°F

Duty cycle goes linearly between these values.
*/
void selectTempLED(float temp){
	PWM_WritePeriod(1000);
	if(temp < 75.0){
		PWM_WriteCompare((int)((temp/75.0) * 1000));
		LED_SEL_Write(2);
```

```c
    } else if(temp < 80.0) {
        PWM_WriteCompare((int)((temp-75.0)/(5.0) * 1000));
        LED_SEL_Write(1);
    } else if(temp < 85.0) {
        PWM_WriteCompare((int)((temp-80.0)/(5.0) * 1000));
        LED_SEL_Write(3);
    } else if(temp < 212.0) {
        PWM_WriteCompare((int)((temp-85.0)/(127.0) * 1000));
        LED_SEL_Write(0);
    }
}
/*
This function selects the LED based on the voltage.

Blue for 0.8V and under
Green for 0.8V-1.6V
Yellow for 1.6V-2.4V
Red for 2.4V-3.3V

Duty cycle goes linearly between these values.
*/
void selectVoltageLED(float voltage){
     PWM_WritePeriod(1000);
    if(voltage < 0.8){
        PWM_WriteCompare((int)((voltage/0.8) * 1000));
        LED_SEL_Write(2);
    } else if(voltage < 1.6) {
        PWM_WriteCompare((int)((voltage-0.8)/(0.8) * 1000));
        LED_SEL_Write(1);
    } else if(voltage < 2.4) {
        PWM_WriteCompare((int)((voltage-1.6)/(0.8) * 1000));
        LED_SEL_Write(3);
    } else if(voltage < 3.3) {
        PWM_WriteCompare((int)((voltage-2.4)/(0.9) * 1000));
        LED_SEL_Write(0);
    }
}
/*
This function gets the voltage value from the ADC.

It returns the value in volts
*/
float getVoltage(){
    // Variable for voltage
    float Vout;
    // Starts the conversion in the ADC
    ADC_StartConvert();
    // Wait for ADC to stop
    ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
    // Get result from channel 1
    int out = ADC_GetResult16(1);
    // Stop the ADC, saves power
    ADC_StopConvert();
    // Converts digital value to voltage
    Vout = out * (3.3/4095);
    return Vout;
```

```c
}

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g.
MyInst_Start()) */
    ADC_Start();
    BTN_ISR_Start();
    LCD_Start();
    PWM_Start();

    tempInit/*Mate*/();
    for(;;)
    {
        // Check for button interrupt
        if(buttonFlag == 1){
            // Clear flag
            buttonFlag = 0;
            // Change mode
            if(mode == 1){
                mode = 0;
                tempInit();
                CyDelay(100);
            } else {
                mode = 1;
                potInit();
                CyDelay(100);
            }
        }
        // Temperature mode
        if(mode ==  0){
            // Delay for refresh rate
            CyDelay(100);
            float temp = getTempValue();
            // Variables for LCD prints
            char topLine[16];
            char bottomLine[16];
            // Format the output
            snprintf(topLine, 16, "Temp: %.2f %cC", temp, 223);
            snprintf(bottomLine, 16, "      %.2f
%cF",convertCtoF(temp), 223);
            // Clear display and print lines
            LCD_ClearDisplay();
            LCD_PrintString(topLine);
            LCD_Position(1,0);
            LCD_PrintString(bottomLine);
            // Select LED
            selectTempLED(convertCtoF(temp));
        }
        // Voltage mode
        else {
            // Refresh rate for LCD
            CyDelay(100);
            float voltage = getVoltage();
```

```
                // Variable for LCD printing
                char topLine[16];
                // Format print
                snprintf(topLine, 16, "Voltage: %.2f V", voltage);
                // Clear display, print string
                LCD_ClearDisplay();
                LCD_PrintString(topLine);
                // Select LED for voltage
                selectVoltageLED(voltage);
            }
        }
}
```

Part 2 BTN.ISR

```
/* `#START BTN_ISR_intc` */
#include "BTN.h"
extern volatile int buttonFlag;
/* `#END` */
...
...
...
CY_ISR(BTN_ISR_Interrupt)
{
    #ifdef BTN_ISR_INTERRUPT_INTERRUPT_CALLBACK
        BTN_ISR_Interrupt_InterruptCallback();
    #endif /* BTN_ISR_INTERRUPT_INTERRUPT_CALLBACK */

    /*  Place your Interrupt code here. */
    /* `#START BTN_ISR_Interrupt` */
    // Small delay for debouncing
     CyDelay(20);
     // Clear interrupt
     BTN_ClearInterrupt();

     // If button is still 1 after delay, set flag, if not, don't
     if(BTN_Read() == 0){
     buttonFlag = 1;
     } else {
     buttonFlag = 0;
     }
    /* `#END` */
}
```