**ENSF 460 Project 2 Group 01**

Cody Casselman, UCID: 30115793

Evan Mann, UCID: 30141069

Keeryn Johnson, UCID: 30170031

# Program Structure

For this assignment, our program is separated into 6 C files along with one Python file for visualization. These files are *Project2Main.c, IOs.c, ADC.c, TimeDelay.c, UART2.c* and *clkChange.c* along with *Project2Python.py*. The program's main functionality happens in *Project2Main.c* while the rest of the files hold functions used for the initialization of peripherals, output visualization and various other functions.

# Initialization

Starting with the following lines of the main C file, we initialize all of the different components of the program including UART, I/O, Timers and the ADC unit.

A black background with white text

Description automatically generated

## UART Setup

For UART, we set up the pins required for sending outputs through the module and assigned the baud rate based on the frequency of the clock. For our program, this is 8MHz, which results in a baud rate of 9600. We elected to use this clock speed because it allows for a faster sampling speed inside the ADC unit, resulting in more precise readings. Finally, we structure the interrupts associated with UART and turn the peripheral on.

A screenshot of a computer program

Description automatically generated

## I/O Setup

The digital input of this project, excluding the potentiometer, was three buttons connected to pins RA2 and RB4, and RA4 labelled as PB1, PB2, and PB3 respectively. Each pin has its TRIS bit assigned to 1 to designate them as inputs; are pulled up to remove input ambiguity and are allowed to trigger CN interrupts. The CN interrupt also has its priority set, its flag cleared and is enabled for the microcontroller in general. Pin RB8 is set to a digital output, representing the LED state. Finally, pin RA3 is assigned as an analog input for the potentiometer to pass its values to be translated to digital values using the microcontroller’s ADC unit.

A screenshot of a computer program

Description automatically generated

## Timer Setup

Our project uses 3 different timers, each with a specific functionality.

Timer 1 has multiple purposes. It controls the length our LED spends blinking (when in ON mode), triggers the reading of an ADC input, and the time the machine spends recording and outputting those ADC inputs. The delay for timer one is set to 10ms, and it is held at this value throughout the duration of the state machine. This 10ms delay is achieved through a prescale value of 1:256. The timer is turned on and off throughout the duration of the machine based on certain triggers and flags.

Timer 2 is used to control the state of the LED. This is done using a two-pronged approach. One is in which timer 2 is variably lengthened and shortened to achieve the PWM signal to cause the light dimness. And the other is by using the timer as a ‘blinking’ timer to control whether the light should just be on or off. To achieve the PWM signal for the light dimming, the PR2 value for timer 2 is variably set in the timer 2 interrupt. This will be further explained later in this report. Timer 2 is given a set prescale value of 1:64.

Lastly, timer 3 is used to control the input buffer delay at 100ms for more accurate input readings and is turned on when every CN interrupt is received. This delay is achieved through a prescale value of 1:1.

Each timer has its prescaler set; its clock source set to the internal clock; is set to run in idle mode; has its priority set; its interrupt flag cleared, and its interrupt enabled. At the end of the timer setup function, all the timers are cleared and turned off so that the main can turn them on as it needs them.

A screenshot of a computer program

Description automatically generated

## ADC Setup

At the core of this assignment is the ADC unit, allowing the input voltages from the potentiometer to be converted into digital readings that can be processed by our program. Knowing the importance of this, careful thought went into the setup of the unit. To begin, the pin that the ADC unit will read its inputs from is assigned by setting AD1PCFG = 0xFFDF, flipping the bit associated with pin A3 to 0. Next, the voltage range used within the conversions is set to 0 – 3.2V and the setup for the sampling the ADC completes is finished. For our program the ADC is initialized such that the internal sampling time is high so that the accuracy of our readings is not affected by any noise. However, it is not set as high as possible as we didn’t want the delay to be too significant that it impacted the regular flow of the rest of the program.

A screenshot of a computer screen

Description automatically generated

# Functionality

## Reading Inputs

To take inputs for the program two main components are used, the CN interrupt which detects any changes in the voltages provided by the buttons and timer 3 which is used as a timer for the input buffer. When a CN interrupt is detected, we store the current state of the button that was pressed (or released) by appending it to the end of a binary number (Called PBXHistory, where X is 1, 2, or 3 depending on the button). We also start timer 3 for the purpose of input buffering which gives us more clarity and precision with our inputs. After a timer 3 interrupt occurs, we then handle whatever button clicks just occurred by isolating the last two bits of these binary numbers. Note, that we define a button “click” as when the statement PBXHistory & 0b11 == 0b10 is true. This means that the last two states of the button were a press and a release, representing a click.

A screenshot of a computer program

Description automatically generated

## The State Machine

A screenshot of a computer screen

Description automatically generated

Our program was made to be a state machine. Initially, we begin in the OFF state. The OFF state has two distinct functionalities depending on whether the program is ‘blinking’ or not. While in OFF, if blinking == 1, we want to make sure that timer 2 is turned on and set to go off every 0.5 seconds. Otherwise, if blinking == 0, we simply want the LED and Timers to be turned off and for the program to wait in Idle. The OFF state reacts to a few different peripherals. If a timer 2 interrupt goes off, the program checks the state of the blinking flag and updates the LED state accordingly. If a PB2 Click is detected, we want to flip the value of the blinking flag. And finally, if a PB1 Click is detected, we want to transition our machine into the ON state by setting the changeMode flag to 1.

A screenshot of a computer

Description automatically generated

Every time the program enters the ON state, we make sure to turn on both timer 1 and timer 2. These are both used for timing ADC readings and performing the PWM signal of the light respectively. The ON state has a few different functionalities depending on certain flags. Since the purpose of the ON state is to record an ADC input and convert that to a PWM signal with an LED, we require a few different flags to make that possible. Namely, we use adc\_flag to determine whether our machine should make an ADC recording. This adc\_flag is set to 1 whenever timer 1 goes off and set to 0 shortly after performing an ADC conversion. The primary method of creating a PWM signal with our LED is by using timer 2 interrupts to cause our LED to blink incredibly fast. If we detect a PB3 click, we flip the value of recordingFlag, making our system either begin recording or stop recording. UART recording is done using our ADC and a running python program. If we detect a PB1 click, we want to begin transitioning to the OFF state, but only if endRecording == 1. This is so that the python half of the program is not left open. It is given an ending transmission before the program transitions to the OFF state.

A screenshot of a computer program

Description automatically generated

Our state machine exists within a switch case within large while (1) loop. After every execution, we ensure that our program idles to reduce power consumption. Only being woken by CN or Timer interrupts.

## Using ADC

The ON state uses the function, *do\_ADC* in *ADC.c*, to read the current value of the potentiometer as an integer. This function takes no inputs and returns the integer value based on the conversion completed by the ADC. To do the conversion the ADC is first turned on, then the SAMP bit is set to begin sampling. A while loop is then used, polling the ADC’s done bit to determine when the ADC has finished sampling the input for the time set during *ADCInit*. Once the sampling finishes, the ADC’s buffer is read into the return value of the function, and the ADC is told to stop sampling and is then turned off.

A screenshot of a computer program

Description automatically generated

## Generating a PWM Signal

To create the impression of a dimming light, we utilize a pulse width modulization signal. This is done only during the ON state of our program and is done using timer 2. In the state ON, we have a variable called intensity. This variable is set by calling do\_adc() (as outlined above) and setting it as an integer from 0 to 1023 representing the brightness we would like the LED to be.

A screenshot of a computer program

Description automatically generated

From intensity, we define two variables, highTime and lowTime using our variable cycleTime which represents the total length of our pulse cycles (approximately 0.01s). These are each used along with a flag named led\_high in our timer 2 interrupt to set the status of the LED and the PR value of timer2 to achieve a PWM signal.

A screenshot of a computer program

Description automatically generated

By variably setting the PR value of timer2, we can essentially slice each of our pulse cycles into two sections. One in which the LED is on, and one in which it is off. Since the pulse cycle happens every 0.01s, this gives the impression that the led is simply dim, while it is blinking incredibly quickly.

## Recording with Python

Recording is performed after a PB3 click is detected while the program is in the ON state. To transmit the ADC values being recorded from do\_ADC, we utilize the given UART functions Disp2Dec, and Disp2String.

When a PB3 click is first detected signifying the program to begin recording, we transmit the string “BEGIN\n” through the UART peripheral. This is then received by the python program, which begins holding onto subsequent transmissions. While outputting ADC values through UART, we ensure that we send the intensity value, a tab, the adc value, and then a newline character in specifically that order. This allows the python program to differentiate between the different values it is receiving. When PB3 is clicked again, or when a minute of recording has passed, we signal to the python program to stop recording by sending the string “END\n” through the UART port.

A screenshot of a computer

Description automatically generated

On the python half, we begin by opening up and listening through a serial port, opened on the COM3 port and listening with a baud rate of 9600. This side of the program begins by being open and waits for the string “BEGIN\n” to come through before collecting data.

A computer screen with text on it

Description automatically generated

After the begin\_detecting flag is set true, we read and record UART data for 10 seconds or until it detects the string “END\n”. We elected to hold the data in a string, and then use various python methods to convert our numbers to floats and give each dataset its own list.

A computer screen shot of text

Description automatically generated

A computer code with colorful text

Description automatically generated with medium confidence

After the function clean\_data is performed, we have three lists; intensities, adcValues, and detectionTimes, each representing the light intensity (in %), the adcValues (as a float), and the time the reading was taken (as a float.) We then go about creating a data frame, csv file, and plots using python’s pandas and plotly modules.

A screenshot of a computer

Description automatically generated