Lab 4: Timer and PWM Operations

Micah Galos - SID: 862082339
Joseph Ayala - SID: 862080244

Due: November 10, 2020 - 11:59PM

Abstract [JDA]

The purpose of this lab is to further develop the implementation of developing an ADC that is capable of reading from a Pulse Width Modulation (PWM) source from an Arduino microcontroller. The arduino microcontroller is utilized in this lab exercise as a signal generator producing a PWM signal on different duty cycles that will be read and interpreted by the K64F and then the respective seven segment displays will illuminate to show the appropriate duty cycle that is being read from the input from PTC10. As in the prior lab the microcontroller is continuously polling the inputs reading in the input from Port C10 to ascertain that value. In the meanwhile the Arduino's main code as seen in the code block below scales up and down the PWM signal that is then displayed.

In the latter portion of the lab the frequency at which these PWM signals were coming in was adjusted from 245 Hz to 490 Hz changing the duty cycle and the duration of these modulated waves which further testbenches the code and the manner in which the information was being read for the Analog Digital Converter.

Several components of this lab are attributed to what was covered in the prior labs with the ADC and continuous polling through the appropriate input pins.

Experiment System Specification

Schematic

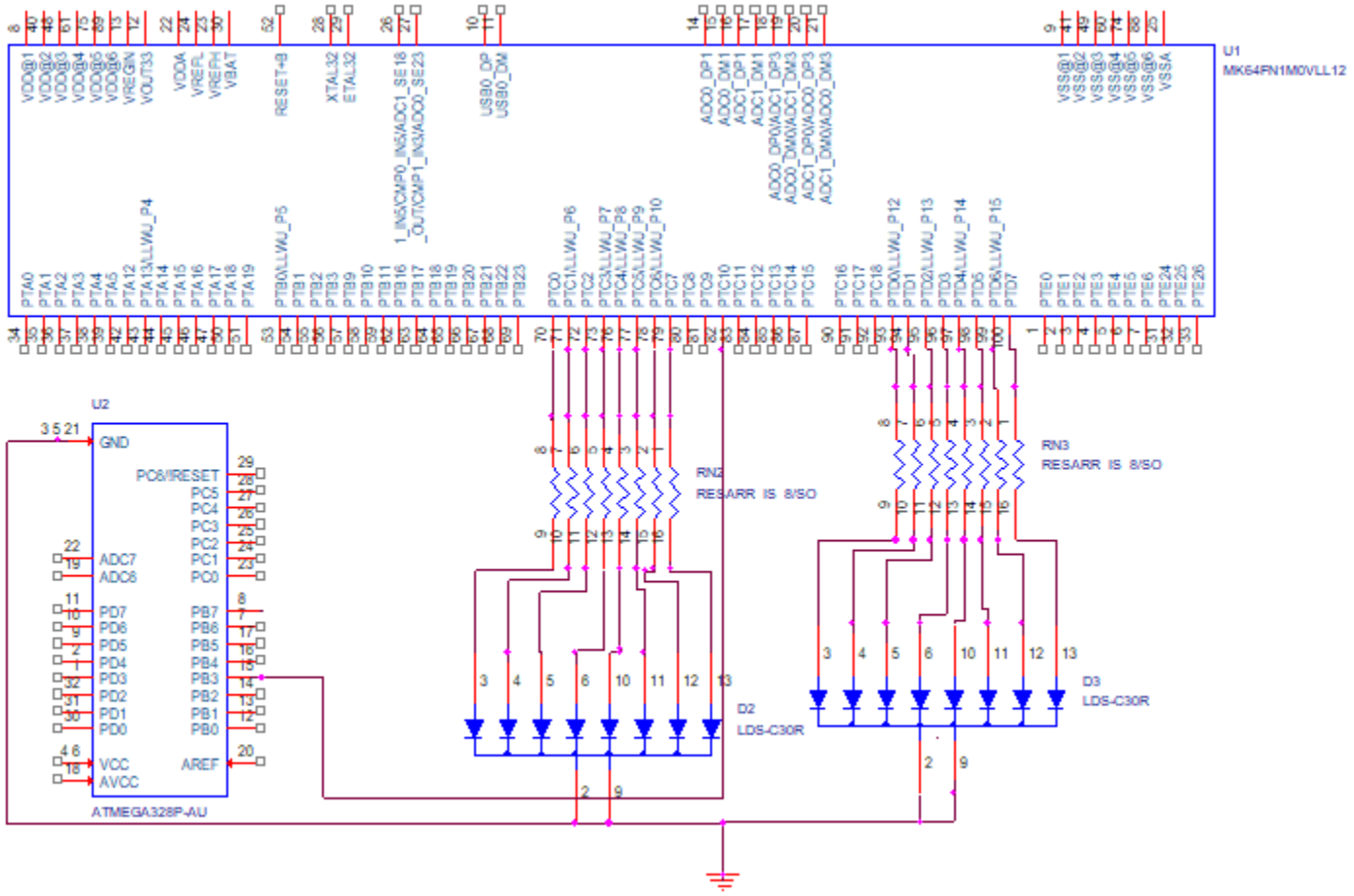

Figure 1

Schematic of Arduino and MK64F

Flowchart

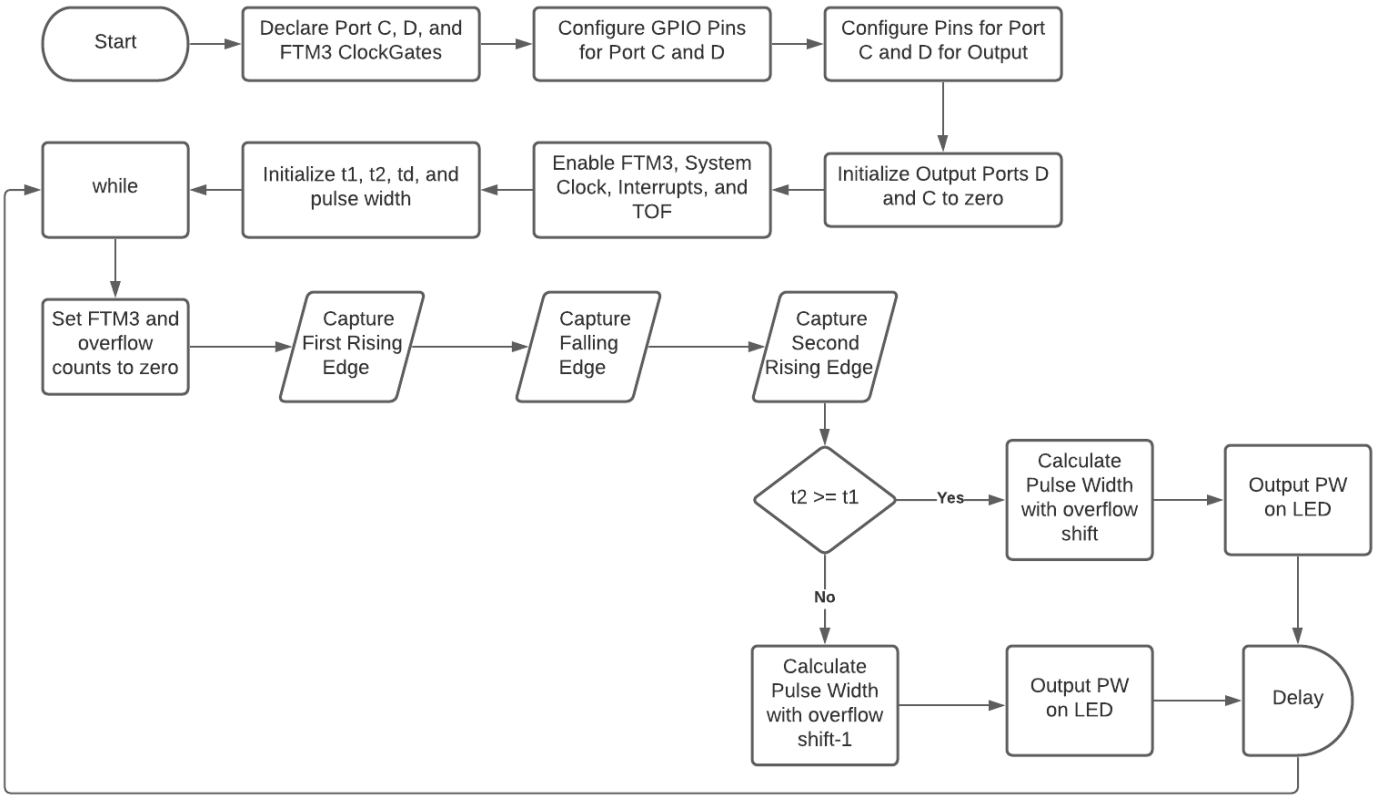

Figure 2
Flowchart of implemented lab exercise

Board & Circuit

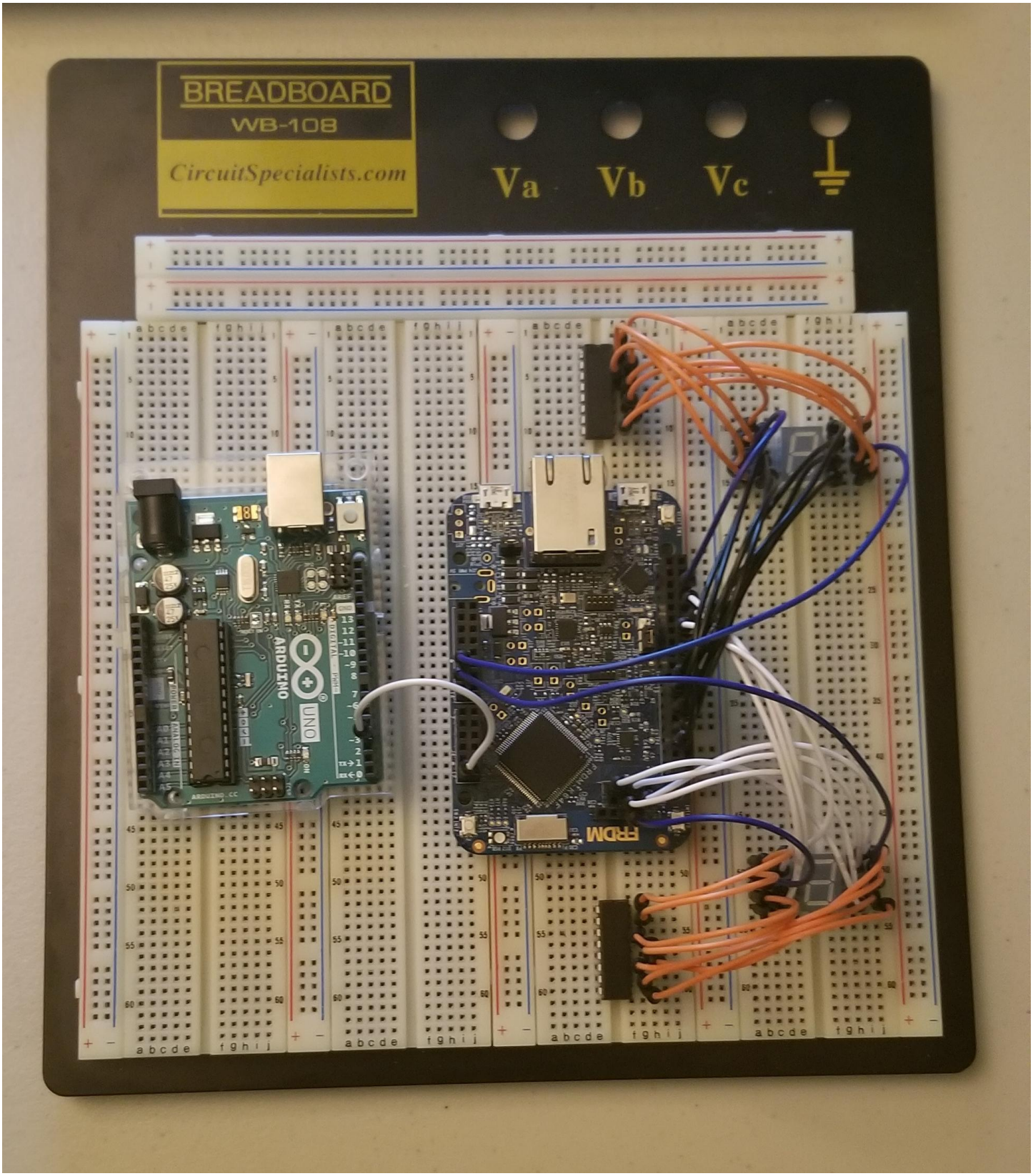

Figure 3
Image of implemented model

Software Design

Arduino Uno Code [JDA]

```
int pwmPin = 3;    // LED connected to digital pin 3
//int pwmPin = 9;    // LED connected to digital pin 9, Lab Question

void setup() {
  // nothing happens in setup
  TCCR2B = (TCCR2B & 0xF8) | 0x05; // Part 3 of Lab 4 Manual
  // 245 Hz

  //TCCR1B = (TCCR1B & 0xF8) | 0x03; // Lab Question
  // 490 Hz
}

void loop() {
  //analogWrite(pwmPin, 1);      // 0 - 0% DC
  //analogWrite(pwmPin, 64);     // 64 - 25% DC
  analogWrite(pwmPin, 128);       // 127 - 50% DC
  //analogWrite(pwmPin, 192);     // 191 - 75% DC
  //analogWrite(pwmPin, 254);    // 255 - 100% DC

  //analogWrite(pwmPin, 77); //Lab Question
  // wait for 1000 milliseconds to see the dimming effect
  delay(1000);
}
```

FRDM-K64F Code [MG]

```
#include "fsl_device_registers.h"

// PORTC 7-SEG
unsigned char seg2[10] = {0xFE, 0xA0, 0x3D, 0xF9, 0xA3, 0xDB, 0xDF, 0xF0,
0xFF, 0xFB};

// PORTD 7-SEG
unsigned char seg1[10] = {0x7E, 0x60, 0x3D, 0x79, 0x63, 0x5B, 0x5F, 0x70,
0x7F, 0x7B};

unsigned long Delay = 0x100000;
unsigned int nr_overflows = 0;

void software_delay(unsigned long delay)
{
     while (delay > 0) delay--;
}
```

```
void outputLED(unsigned int val)
{
	unsigned long D_out = 0x00;
	unsigned long C_out = 0x00;
	unsigned int result = 0x00;

	result = val - (nr_overflows << 16);

	if(result >= 100){
		GPIOC_PDOR = seg2[9];
		GPIOD_PDOR = seg1[9];
	}
	else{
		C_out = ( result / 10 );
		D_out = ( result % 10 );

		if(C_out == 0) GPIOC_PDOR = seg2[0];
		else if(C_out == 1) GPIOC_PDOR = seg2[1];
		else if(C_out == 2) GPIOC_PDOR = seg2[2];
		else if(C_out == 3) GPIOC_PDOR = seg2[3];
		else if(C_out == 4) GPIOC_PDOR = seg2[4];
		else if(C_out == 5) GPIOC_PDOR = seg2[5];
		else if(C_out == 6) GPIOC_PDOR = seg2[6];
		else if(C_out == 7) GPIOC_PDOR = seg2[7];
		else if(C_out == 8) GPIOC_PDOR = seg2[8];
		else if(C_out >= 9) GPIOC_PDOR = seg2[9];

		if(D_out == 0) GPIOD_PDOR = seg1[0];
		else if(D_out == 1) GPIOD_PDOR = seg1[1];
		else if(D_out == 2) GPIOD_PDOR = seg1[2];
		else if(D_out == 3) GPIOD_PDOR = seg1[3];
		else if(D_out == 4) GPIOD_PDOR = seg1[4];
		else if(D_out == 5) GPIOD_PDOR = seg1[5];
		else if(D_out == 6) GPIOD_PDOR = seg1[6];
		else if(D_out == 7) GPIOD_PDOR = seg1[7];
		else if(D_out == 8) GPIOD_PDOR = seg1[8];
		else if(D_out >= 9) GPIOD_PDOR = seg1[9];
	}
}

void FTM3_IRQHandler(void) {
	nr_overflows++;
	uint32_t SC_VAL = FTM3_SC;
	FTM3_SC &= 0x7F; // clear TOF
}
```

```
int main(void) {
	SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK; // Port C clock enable
	SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK; // Port D clock enable
	SIM_SCGC3 |= SIM_SCGC3_FTM3_MASK; // FTM3 clock enable

	/* Configure Port C Pins 0-5 and 7-8 for GPIO; */
	PORTC_GPCLR = 0x01BF0100;
	/* Configure Port D Pins 0-7 for GPIO; */
	PORTD_GPCLR = 0x00FF0100;

	/* Configure Port D Pins 0-7 for Output; */
	GPIOD_PDDR |= 0x000000FF;

	/* Configure Port C Pins 0-5 and 7-8 for Output; */
	GPIOC_PDDR |= 0x000001BF;

	/* Initialize Port C and D to 0; */
	GPIOC_PDOR |= 0x00;
	GPIOD_PDOR |= 0x00;

	PORTC_PCR10 = 0x300; // Port C Pin 10 as FTM3_CH6 (ALT3)
	FTM3_MODE = 0x5; // Enable FTM3
	FTM3_MOD = 0xFFFF;
	FTM3_SC = 0x0D; // System clock / 32 as prescaler
	NVIC_EnableIRQ(FTM3_IRQn); // Enable FTM3 interrupts
	FTM3_SC |= 0x40; // Enable TOF

	unsigned int t1, t2, td, pulse_width;

	while (1) {
		FTM3_CNT = 0; nr_overflows = 0; // initialize counters

		FTM3_C6SC = 0x4; // rising edge
		while(!(FTM3_C6SC & 0x80)); // wait for CHF
		FTM3_C6SC &= ~(1 << 7);
		t1 = FTM3_C6V; // first edge

		FTM3_C6SC = 0x8; // falling edge
		while(!(FTM3_C6SC & 0x80)); // wait for CHF
		FTM3_C6SC = 0; // stop C6
		td = FTM3_C6V;

		FTM3_C6SC = 0x4; // rising edge
		while(!(FTM3_C6SC & 0x80)); // wait for CHF
		FTM3_C6SC &= ~(1 << 7);
```

```
            FTM3_C6SC = 0; // stop C6
            t2 = FTM3_C6V; // second edge

            if (t2 >= t1){
                pulse_width = (nr_overflows << 16) + ( (td - t1) * 100 /
(t2 - t1) );
                outputLED(pulse_width);
            }
            else{
                pulse_width = ( (nr_overflows-1) << 16) + ( (td - t1) * 100
/ (t2 - t1) );
                outputLED(pulse_width);
            }
            software_delay(Delay); // breakpoint here for debugging
        }
}
```

Technical Problems

First issue was figuring out why the pulse width output was not outputting the correct result from the Arduino to the K64F board. The Arduino Uno should not have any problems because we are inputting a signal to the K64F, so the problem was in the K64F. After a few debugs, we noticed we only captured the first rising edge and the falling edge until hitting the second--never captured the second rising edge. We modified the code so that it now captures the input at both falling edge and second rising edge.

Pre-Lab Questions

1. What is the notion of common ground in electronic circuits? Do Arduino and K64F need common ground in this lab?

For the purpose of this lab both the signal source (arduino) and K64F microcontroller will share a common ground to isolate the two different components that share a common reference point.

2. If the Arduino is set up to do PWM at 490 Hz and the duty cycle is 30%, what is the duration that the signal is high and the duration that the signal is low for one cycle?

Period
T = 1/490 Hz = 2 ms

High
490 Hz * 30% = 147 Hz

Duration at High
2 ms * 30% = 0.6 ms

Low
490 Hz * 70% = 343 Hz

Duration at Low
2 ms * 70% = 1.4 ms

Questions

1. When the PWM circuit modulates the LED, what is the minimum frequency you can use before you start to see the LED blink?

From the Arduino Uno, the minimum frequency to be able to see the LED blink was roughly around 50 Hz.

2. For the input capture circuit, what prescale values did you choose to use? When would you make the prescale value higher or lower?

We only used a prescale value 32, 64, and 128 for this lab. According to the arduino pwm cheat sheet, they have frequencies of 980 Hz, 490 Hz, and 245 Hz, respectively. Using these prescaler values are to measure the high and low frequencies. Higher prescale for low frequencies and low prescale for high frequencies.

Conclusion [MG]

Overall, the lab was a success as there were no hardware issues unlike last time. I got to utilize more functions to lessen writing for the main function by separating each process and calling them in main. The lab itself was super straightforward and did not have to think as hard in designing the software since we have the starter code.