.

CMPE 460 Lab 5: K64 Timers, Interrupts, and Analog-to-Digital Converter

Austin Brogan and Susan Margevich
Performed: February 13th, 2019
Submitted: February 27th, 2019

Lab Section: 01L1
Instructor: Dr. Raymond Ptucha
TAs: Pablo Ordorica
Joseph Panaro
Brendan Wells
Jason Blocklove

Lecture Section: 01
Professor: Dr. Raymond Ptucha

Table of Contents

Abstract

The purpose of this exercise was to explore the enabling and use of interrupts in various modules, using timers to measure time, digitize analog signals, and read analog input from a line scan camera. The exercise had three parts that each used the K64 to do different things. Part 1 involved toggling an LED with one button that blinked on and off every second using the PDB as a timer, as well as using the FTM timer to measure how long a second button has been pressed and held, displaying this result in a terminal. In order to do this, LED_Init and Button_Init from previous labs were used to turn the LEDs on via the buttons on the K64 and the initGPIO file used SW3 or SW2 to cause an interrupt when pressed. isr.c contained IRQ handlers for the PDB and FTM as well as functionality determining what happens when each button is pressed. Part 2 involved digitizing an analog signal from a photo-electronic and thermistor device, displaying binary logic to a terminal through UART and converting the values to temperatures so the changes detected by the sensor can be visualized. A voltage divider was built for the photocell with a resistor, connecting to the ADC as well as power and ground from the K64, allowing for the values from the photo-electronic device to accurately measure the ambient light being sensed. The ADC1_Init function was completed in order to enable interrupts and hardware triggering. Part 3 used timers, interrupts, and analog to digital conversion to connect a line scan camera to the NXP Cup Car and allow for a black line on a white surface to be tracked by the camera. These results were shown both in MATLAB and on an oscilloscope, both of which showed graphs displaying the location of the line relative to the camera. This was performed using the CK (clock), SI (serial input to the sensor), and AO (analog output) and a mounted camera on top of the NXP car, the code being written in camera_FTM.c and using timer modules and the ADC. The exercise was incredibly successful in its purpose, and the usage of interrupts, timers, and ADCs was overall both useful and applicable to future exercises.

Design Methodology

Part 1 of the exercise involved testing the timers' and interrupts' functionality in order to use the switches on the K64 to trigger interrupts when the button was pressed or released. After a certain period of time, an interrupt was also generated by the timer. The functionality of Part 1 involved using one of the switches on the K64 to toggle an LED and use the PDB timer to turn the LED on and off every second, and used the other twitch to turn the LED on and use the FlexTimer Module, or FTM, to time how long the LED has been turned on. This value was then displayed in a terminal to the user via the UART.

The first step taken in order to perform Part 1 was to initialize the UART0 in order to allow for the output text to be displayed to the terminal. This function, uart_init(), enables the UART0 clock, configures the port control register to the 16th pin on port B, enables the clock for port B, sets the baud rate, and enabled the ability of the UART to transmit and receive. uart_init() also contained functions that allowed for characters to be put, got, and for strings to be put, in the functions uart_getchar(), uart_putchar(), and put().

The PDB timer and FTM timer were initialized next in main_timers.c. The PDB clock was enabled, as was its timer, and set to continuous mode with a prescaler of 128, a multiplication factor of 20, and software triggering. The mod field was set to trigger two interrupts per second, two mods counting in one second in order to turn the LED both on and off, the calculation performed being multiplying the system clock by thirty, shifting to the left 7 bits, then dividing by 1000. The FTM timer was enabled with write protection disabled, with the FTM0 using an input clock by 128. An overflow interrupt was enabled with a mod field so that the overflow flag was set every millisecond.

The GPIO was initialized next in main_timers.c, allowing for the functionality of the Switch 2 (SW2), Switch 3 (SW3), and the LED. SW2 used the port C clock to be enabled and used the 6th pin on port C. The interrupt configuration was set so that there would be an interrupt on both the rising and falling edge of the button, which would allow for an interrupt when the button was pressed as well as when it was released. SW3 used the port A clock to be enabled and used the 4th pin on port A. This button was only set so that there would be an interrupt on the rising edge, which would only allow for an interrupt when the button was pressed. Both switches were set as inputs on the GPIO data direction register. The port B and port E clocks were enabled for the LEDs, with the LEDs set as an output on the GPIO data direction register. The 21st and 22nd pins on port B as well as the 26th pin on port E was set to the LEDs on the port control register.

A main function was written that initialized the PDB, GPIO, FTM, UART, as well as interrupts. Interrupts were enabled with an NVIC for PDB0, FTM0, port A, and port C. The main function introduced an infinite for loop that waits for interrupts, with global variables in isr.c that stored the state of buttons being pressed, the millisecond count, as well as whether the timer was active or not. The interrupt handler for the SW3 interrupt determines whether the LED is toggled and thus, whether the timer is active, and if this interrupt is enabled, the PDB timer is enabled. The PDB interrupt handler toggles the state of the red LED if toggling is enabled, and when SW3 is pressed again, toggling disables and the LED no longer blinks on and off. The interrupt handler for the SW2 interrupt determines the state of the button and updates when the button is either pressed or released. When the button is pressed, the global variable indicating the counter is set to 0, the FTM counter is reset to 0, and the blue LED is enabled. When the button is released, the global variable indicating the counter is printed using the UART, and the blue LED is disabled. The FTM timer interrupt occurs once every millisecond, and the counter variable increases so that the UART displays how many milliseconds the button has been held down, which is the time between the press of the button and the release.

Part 2 of the exercise involved testing the functionality of the analog to digital converter, the ADC. Using a simple circuit, a voltage divider read the voltage across a photocell CDS cell as well as a TMP35 temperature sensor that outputs a voltage value, the photocell having the ability to measure the input of light and the temperature sensor having the ability to measure temperature.

In order to initialize the ADC, the file main_adc.c was used, with initialization being performed in ADC1_INIT(). The clock for the ADC was enabled, and the ADC was configured to use a divide 50MHz down to 6.25MHz in a 16-bit conversion. Single-ended mode for the ADC1 channel DADP3 was used with the hardware trigger enabled, with the analog data being converted to digital data that can be accessed with the ADC data result register. A main function was written that initialized the ADC, the UART0, and the PDB timer, with an endless loop that waited in between iterations. The value of the ADC data result register was read in, with this value then being converted to a voltage value and printed to the user in both hexadecimal and decimal. The ADC, when connected to the photocell circuit, only prints the voltage reading, and when the ADC is connected to the TMP36, the voltage reading is converted to a Celsius value as well as a Fahrenheit value and printed as such.

The value read in by the ADC was implemented using a simple circuit including a voltage divider, reading the voltage across a photocell. The circuit is shown below using a design from the Interface & Digital Electronics Laboratory Manual by the Department of Computer Engineering in Figure 1.

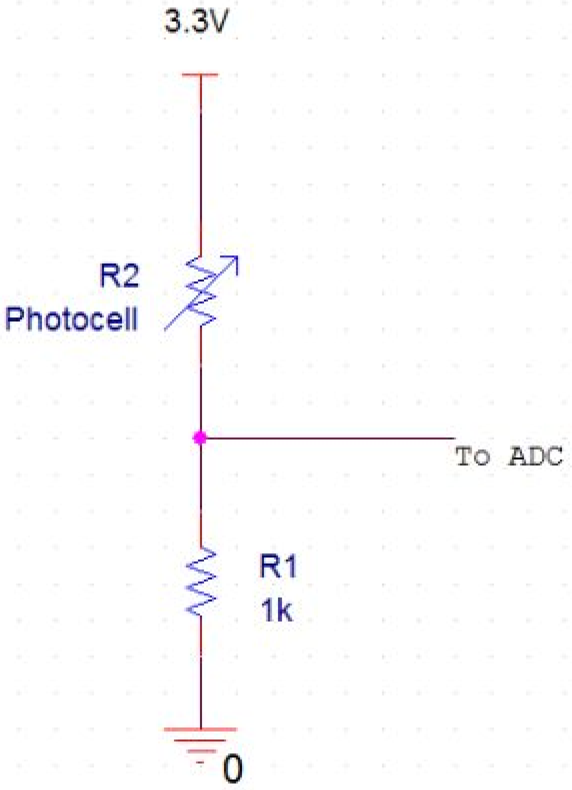


Figure 1: Photocell CDS Cell Circuit with Voltage Divider

The photocell is placed in series with a 1kΩ load resistor, the real value of which being 0.98kΩ with 3.3V of power and ground directly below the load resistor. Power and ground both came from pins on the K64 board. The photocell has a resistance that varies based on the light being exposed to it, changing the voltage that is read by the ADC. By finding the difference between the V_{CC} and the voltage read by the ADC, the voltage across the photocell can be determined and printed to the UART.

The TMP36 worked in a similar fashion as the photocell, with the program reading the value outputted by the sensor. The varying resistance due to the temperature in the sensor changed the voltage read by the ADC, and a variable was created in the main function that converted the value from a voltage to a temperature in Celsius. This Celsius value was also converted into Fahrenheit, and both values were printed to the terminal via the UART.

Part 3 of the exercise involved capturing data from the line cameras using the FTM, a periodic interrupt timer, an ADC, and GPIOs. The CLK and SI were driven with GPIO, whereas the PIT timer controlled the integration period. Interrupts were enabled from the FTM2 module on overflows, making the FTM2 and ADC active for 128 clock cycles to generate camera signals as well as to read the camera output.

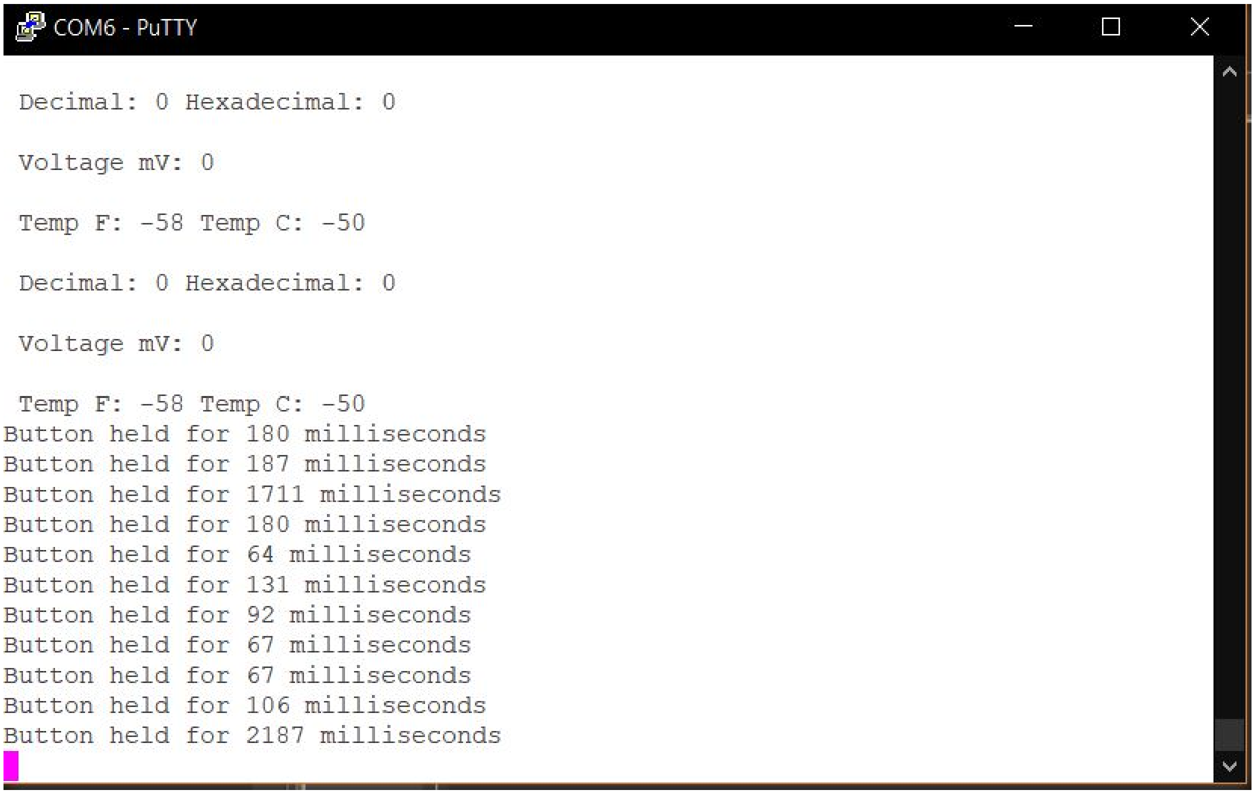
A main function was written to run on an infinite for loop, so that every two seconds, the values put into the array via the UART, printing the values gathered from the line scan camera. The FTM timer handled the camera logic, getting called once every integration period. When the FTM timer is triggered, the SI pulse is given, the clock is toggled, and the line data from the ADC is stored from the ADC and into the line variable. If the clock value is on the falling edge, the ADC reads and the array indexes, otherwise, FTM2 interrupts are disabled until there is a PIT0 overflow. The PIT0 determined the integration period, and when overflowed, the clock logic is triggered and the FTM counter is reset in order to control when the line capture occurs. The FTM was initialized in edge-aligned mode with ELSB set to 1 and ELSA set to 0. The GPIO enabled the LEDs and the GPIO so results could be seen, and the ADC was set up so that camera data could be captured. The ADC was set to single-ended mode, with an FTM2 trigger on ADC0 and interrupts enabled. Clocks were enabled on ports B and E for LEDs, the GPIO signal multiplexer was configured, and the GPIO pins were set to output mode as well as the LEDs being turned off.

MATLAB code was written in order to display graphs showing where the black line on a white sheet of paper was, indicating proper functionality of the line scan camera. The serial port was set to 52, and a five-point average was set up with a range of 3 to 126, using values of i-2, i-2, i, i+1, and i+2 in order to get a proper average. Edge detection was done via thresholding, with the logic that if the smooth trace was greater than 800, the index would be set equal to 1, otherwise, it would be set equal to 0. This allowed for proper locating of the line captured by the line scan camera, and the plot was drawn.

Issues encountered throughout the exercises included difficulties with the oscilloscope, as a bad probe was used initially and had to be replaced. There was also an issue with finding the range in the MATLAB code as well as the value 800 for the period, and a significant amount of trial and error was needed in order to find a value that would properly display on the oscilloscope as well as in MATLAB. There were minor issues including using the incorrect operations on masks (such as not using an or-equals when necessary) and small syntax bugs.

Results and Analysis

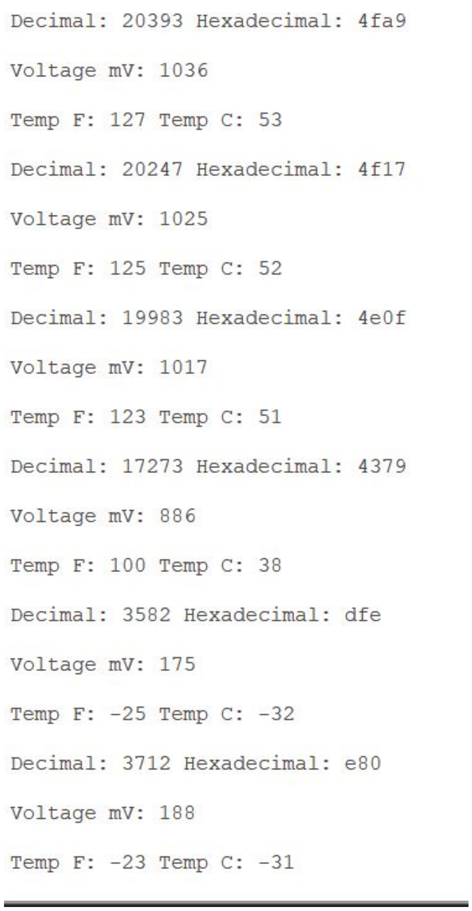
For Part 1, the functionality of SW2 and SW3 was tested by loading the code onto the board and pressing the switches to test their properties. SW3 was tested first, and upon being pressed, the red LED turned on and began blinking once every second. The button, when pressed again, turned the LED off. SW2 was tested by looking at the terminal output from the UART while pressing the button, testing how whether the timers would properly time how long the button had been held down. The shortest amount of time held was 64 milliseconds, and the longest time was 2187 milliseconds. Theoretically, the longest amount of time the button could be held for would be 2,147,483,647 milliseconds, the maximum value for a 32-bit integer. Figure 2 shows the functionality of SW2.



```
 Decimal: 0 Hexadecimal: 0

 Voltage mV: 0

 Temp F: -58 Temp C: -50

 Decimal: 0 Hexadecimal: 0

 Voltage mV: 0

 Temp F: -58 Temp C: -50
Button held for 180 milliseconds
Button held for 187 milliseconds
Button held for 1711 milliseconds
Button held for 180 milliseconds
Button held for 64 milliseconds
Button held for 131 milliseconds
Button held for 92 milliseconds
Button held for 67 milliseconds
Button held for 67 milliseconds
Button held for 106 milliseconds
Button held for 2187 milliseconds
```

Figure 2: Terminal Output for SW2 Functionality

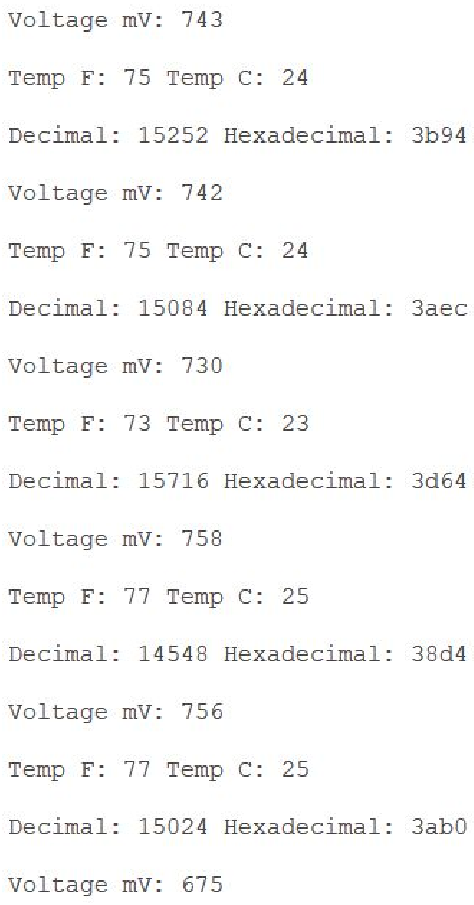
For Part 2, a phone flashlight was shined on the photocell in order to show that the voltage values would increase as the amount of light increased, and the photocell was covered with a finger to show that the voltage values would drop when little to no light was captured by the photocell. Figure 3 shows the functionality of the photocell (with unnecessary conversions into temperature, as the photocell does not capture temperature values.)



```
Decimal: 20393 Hexadecimal: 4fa9

Voltage mV: 1036

Temp F: 127 Temp C: 53

Decimal: 20247 Hexadecimal: 4f17

Voltage mV: 1025

Temp F: 125 Temp C: 52

Decimal: 19983 Hexadecimal: 4e0f

Voltage mV: 1017

Temp F: 123 Temp C: 51

Decimal: 17273 Hexadecimal: 4379

Voltage mV: 886

Temp F: 100 Temp C: 38

Decimal: 3582 Hexadecimal: dfe

Voltage mV: 175

Temp F: -25 Temp C: -32

Decimal: 3712 Hexadecimal: e80

Voltage mV: 188

Temp F: -23 Temp C: -31
```

Figure 3: Terminal Output for Photocell Functionality

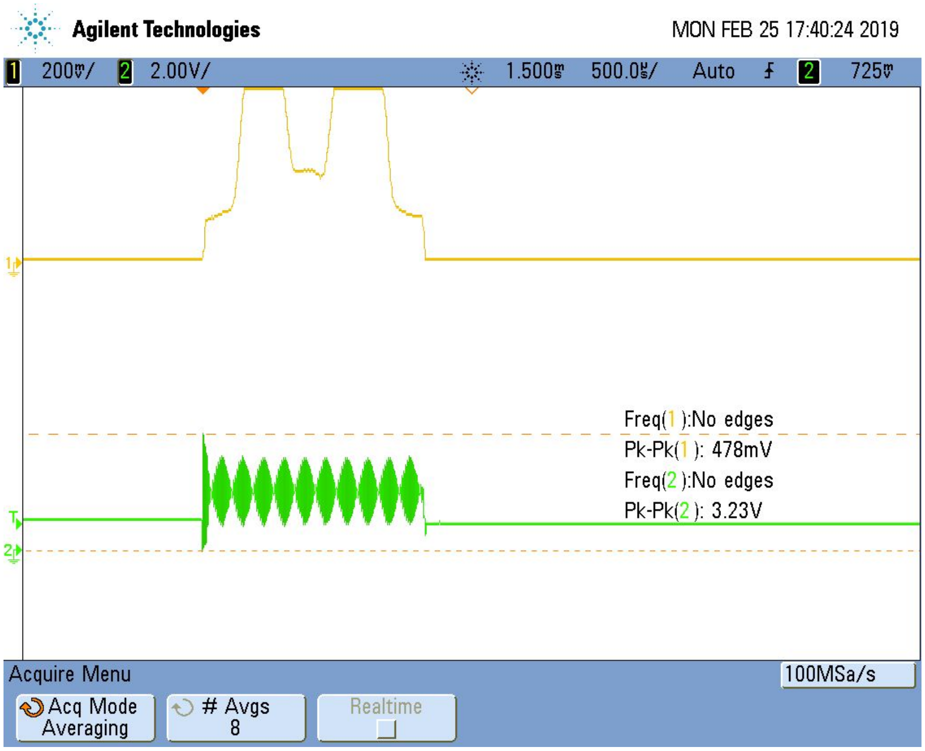
The second part of Part 2 used the TMP36 to measure the temperature being read and was tested by setting the sensor in front of a computer fan to heat it up and then block it from the fan to cool it down. The value went down when the temperature was lowered, and the value went up when the temperature increased. Figure 4 shows the functionality of the temperature sensor.



```
Voltage mV: 743

Temp F: 75 Temp C: 24

Decimal: 15252 Hexadecimal: 3b94

Voltage mV: 742

Temp F: 75 Temp C: 24

Decimal: 15084 Hexadecimal: 3aec

Voltage mV: 730

Temp F: 73 Temp C: 23

Decimal: 15716 Hexadecimal: 3d64

Voltage mV: 758

Temp F: 77 Temp C: 25

Decimal: 14548 Hexadecimal: 38d4

Voltage mV: 756

Temp F: 77 Temp C: 25

Decimal: 15024 Hexadecimal: 3ab0

Voltage mV: 675
```

Figure 4: Terminal Output for TMP36 Functionality

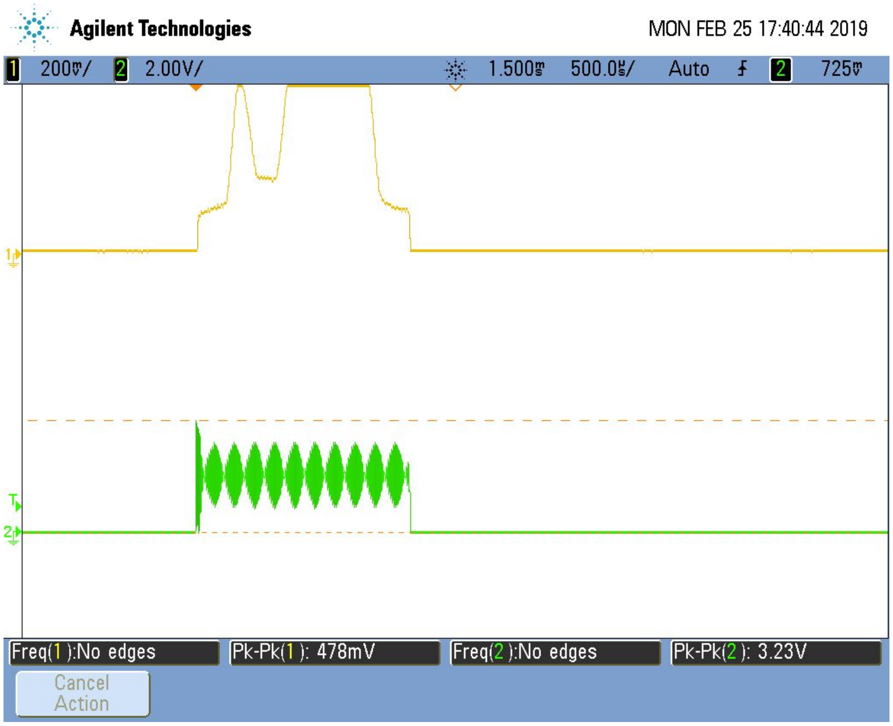
Part 4 of the exercise was verified using both the oscilloscope and a MATLAB script to read in values from the camera using the serial port connected to the UART module on the K64. A sheet of white paper with a large black line was held in front of the camera, and the data from the camera was sent through the UART to the COM port. MATLAB then interpreted this input and graphed it. A similar setup was used with the oscilloscope, and captures of the oscilloscope were captured as well. Figure 5 shows the output on the oscilloscope with the black line held directly in the center of the camera's view, and Figure 6 shows the output on the oscilloscope with the black line offset slightly from the center of the camera. The dip in the values indicates the position of the line on the paper with respect to the camera.



Freq(1):No edges
Pk-Pk(1): 478mV
Freq(2):No edges
Pk-Pk(2): 3.23V

Acq Mode Averaging
Avgs 8

Figure 5: Oscilloscope Capture with Line in Center of Camera



Freq(1):No edges
Pk-Pk(1): 478mV
Freq(2):No edges
Pk-Pk(2): 3.23V

Figure 6: Oscilloscope Capture with Line to the Side of Camera

Figure 7 shows the output to MATLAB with three graphs. The first graph is the raw data from the camera, with a similar dip showing the position of the line in relation to the camera. The second graph shows the same data but plotted smoother, and the third plot is the edge detection plot in binary. Figure 8 shows the same graphs with the line offset slightly to show proper tracking functionality.

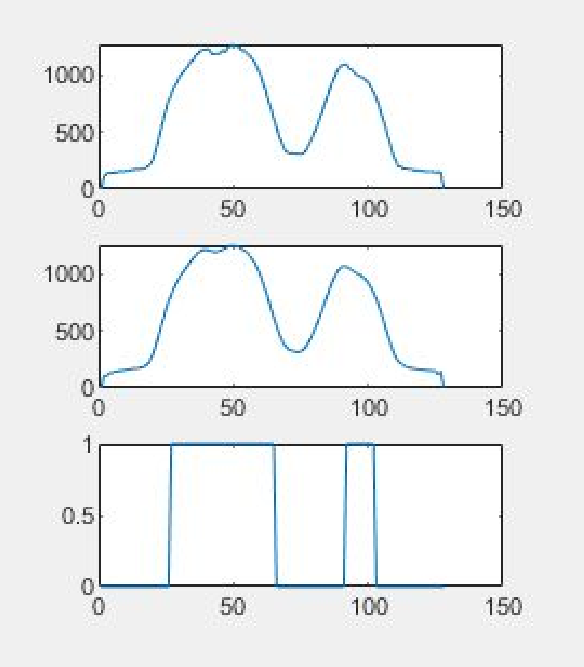


Figure 7: MATLAB Graphs from Line Scan Camera (Centered Line)

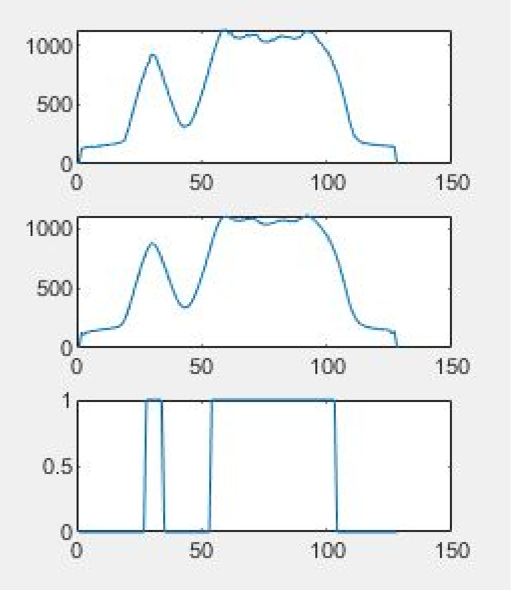


Figure 8: MATLAB Graphs from Line Scan Camera (Displaced Line)

The PDB timer, when operating at a MOD value of 32,000, will reset the counter every 320 ms. This means that the maximum frequency the PDB can run at is 1.5625 KHz, but since the ADC runs at 6.25MHz, aliasing will occur and can be avoided by running the PDB at a frequency twice that of the ADC.

Questions

1. Why do the IRQ flags need to be cleared in the ISR functions?

   The ISR gets branched to when an interrupt flag is set, and the IRQ has to be cleared so the system can be informed that the request was handled. Otherwise, the CPU will continually enter the ISR, as the interrupt flag is never getting cleared, and the program will fail.

2. Which timer would you use to measure time while power is off?

   The RTC, or real-time clock, can be used even when the power is off due to the fact that it is powered by an external source.

3. Why does the LDOK field for the PDB0_SC register need to be asserted?

   The LDOK field is responsible for finalizing the writes written to other registers in the PDB0. When the LDOK is set to 0, the values are not properly set to registers and are only set in buffers.

Conclusion

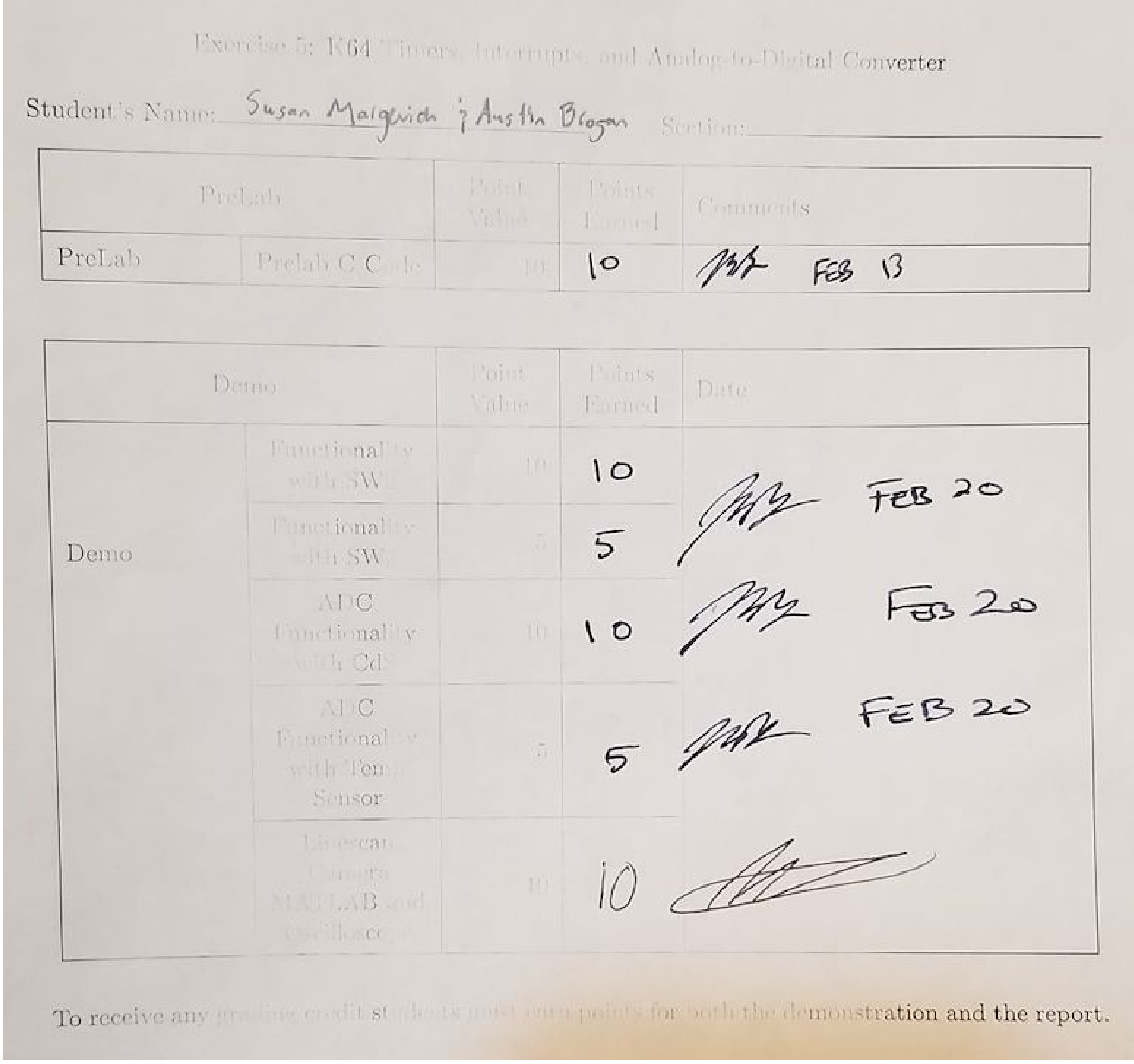
Overall, the lab was incredibly successful in its purpose to explore the enabling and use of interrupts in various modules, using timers to measure time, digitize analog signals, and read analog input from a line scan camera. The switches operated as needed, with the first switch toggling the red LED when pressed and the second switch properly timing how long the button was held for. The ADC properly measured changes in both light and temperature to their respective sensors, with voltages fluctuating as the sensors received different values. The line scan camera successfully found a line on a sheet of paper and outputted the information to MATLAB as well as the oscilloscope. The lab was useful for teaching concepts as well as being applicable to future exercises as well as other projects, as the uses for the information learned can be highly applicable to other uses.

References

Figure 1 was obtained from “Exercise 5: K64 Timers, Interrupts, and Analog-to-Digital Converter” in the Interface & Digital Electronics Laboratory Manual by the Department of Computer Engineering

Appendix

Sign off sheet:



Exercise 5: K64 Timers, Interrupts, and Analog-to-Digital Converter

Student's Name: Susan Malgevich & Austin Brogan Section:

PreLab		Point Value	Points Earned	Comments
PreLab	Prelab C Code	10	10	FEB 13

Demo		Point Value	Points Earned	Date
Demo	Functionality with SW1	10	10	FEB 20
	Functionality with SW2	5	5	
	ADC Functionality with CdS	10	10	FEB 20
	ADC Functionality with Temp Sensor	5	5	FEB 20
	Linescan Camera MATLAB and Oscilloscope	10	10	

To receive any grading credit students must earn points for both the demonstration and the report.