**Design Assignment 4:**

**Fixed Point Numbers, Real Time Response, and Interrupts**

****

**Embedded Systems**

**ELC 411**

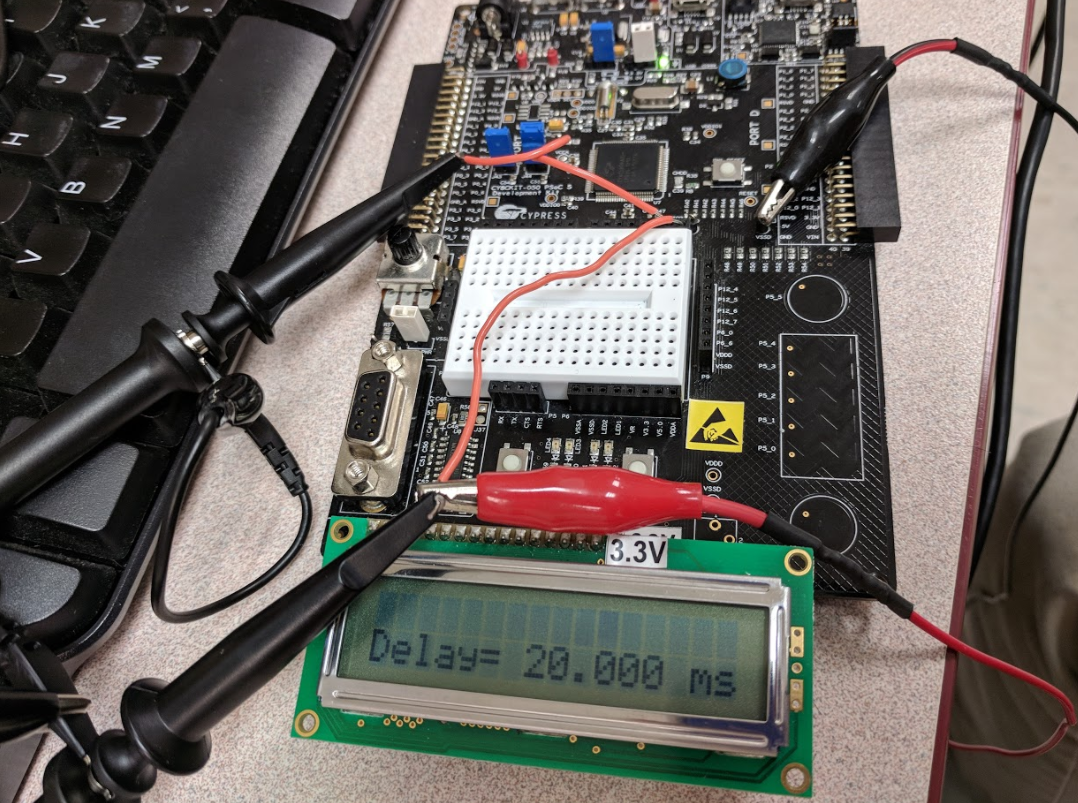
**Matthew Strickland and Jacob Levine**

**Submission: 11/8/17**

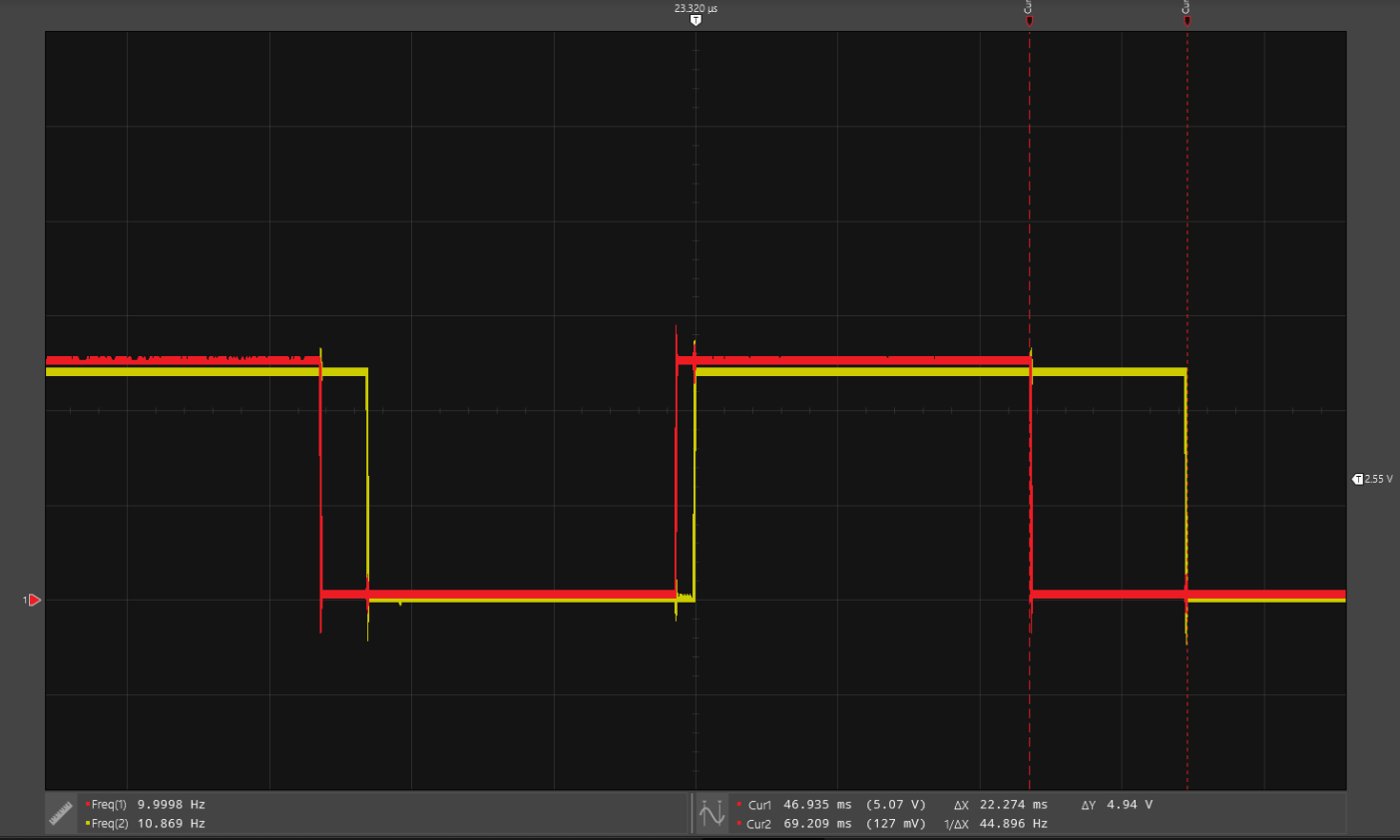
1. **Observations**

***Part I:***

Figure 1 below shows the test setup and LCD display during Part I of this lab experiment. SW3 was used to increment the delay and SW2 was used to decrement the delay. The upper and lower bounds of the delay were 200 mS and 20 mS respectively.



*Figure 1: LCD Displaying Minimum Delay (20 mS)*

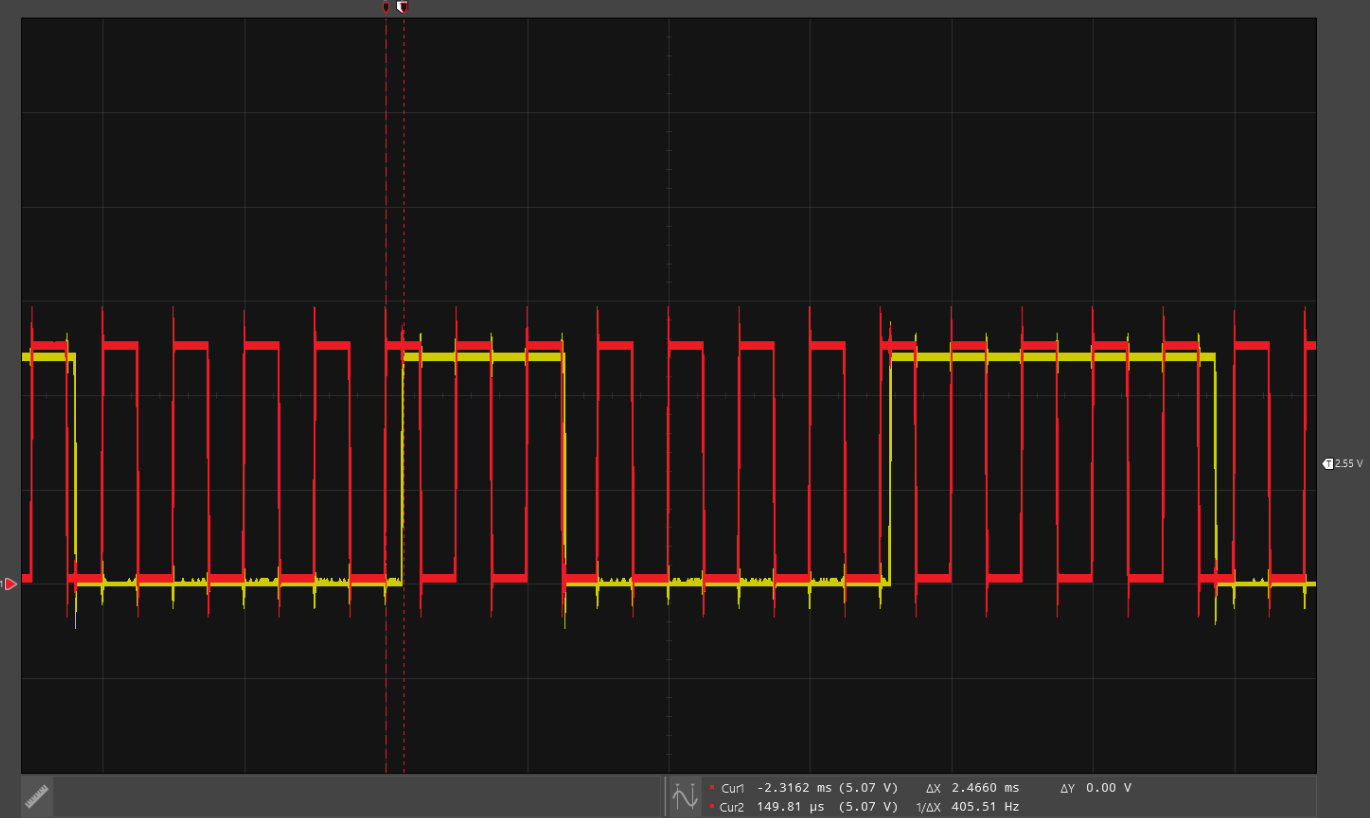
**

*Figure 2: Input (red) to Output (yellow) arbitrary latency (Min delay, 10 Hz input)*

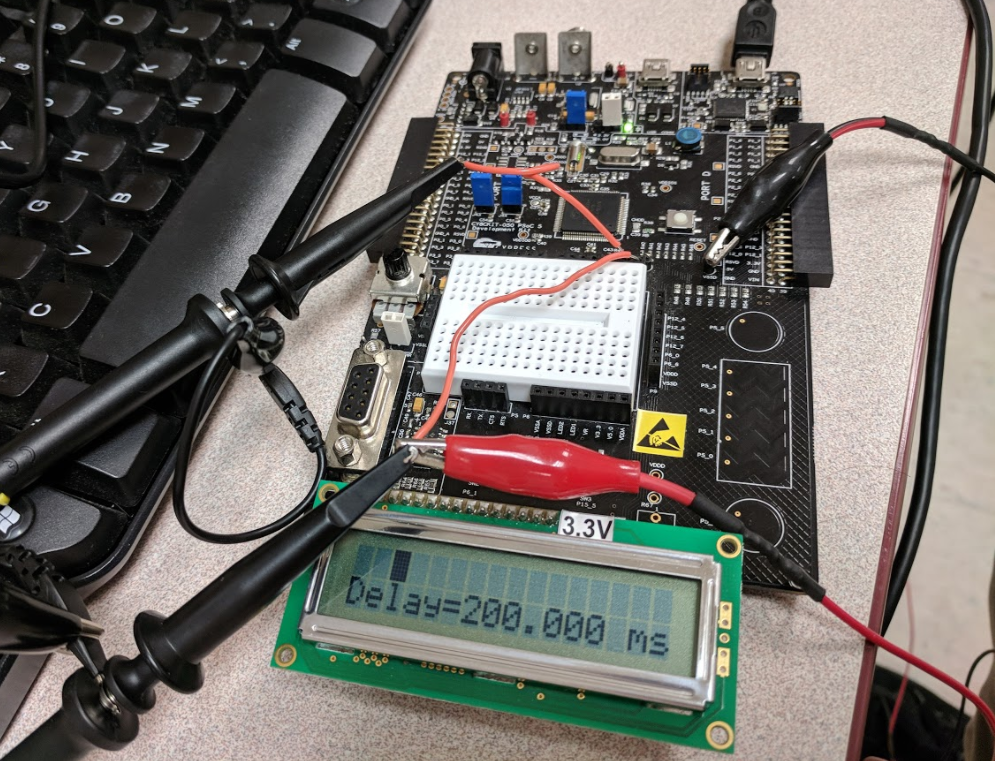
*Please give volts/div and time/div within captions for all scope traces!*

Note the latency in Figure 2 (above), This is using the minimum delay of 20 mS and has a 10 Hz square wave on the input. The on-screen cursors of the oscilloscope 22.5 mS yet the two transitions prior are noticeably shorter. Although the output follows the input, the latency between them varies dependent on where the PC is at in the for loop.

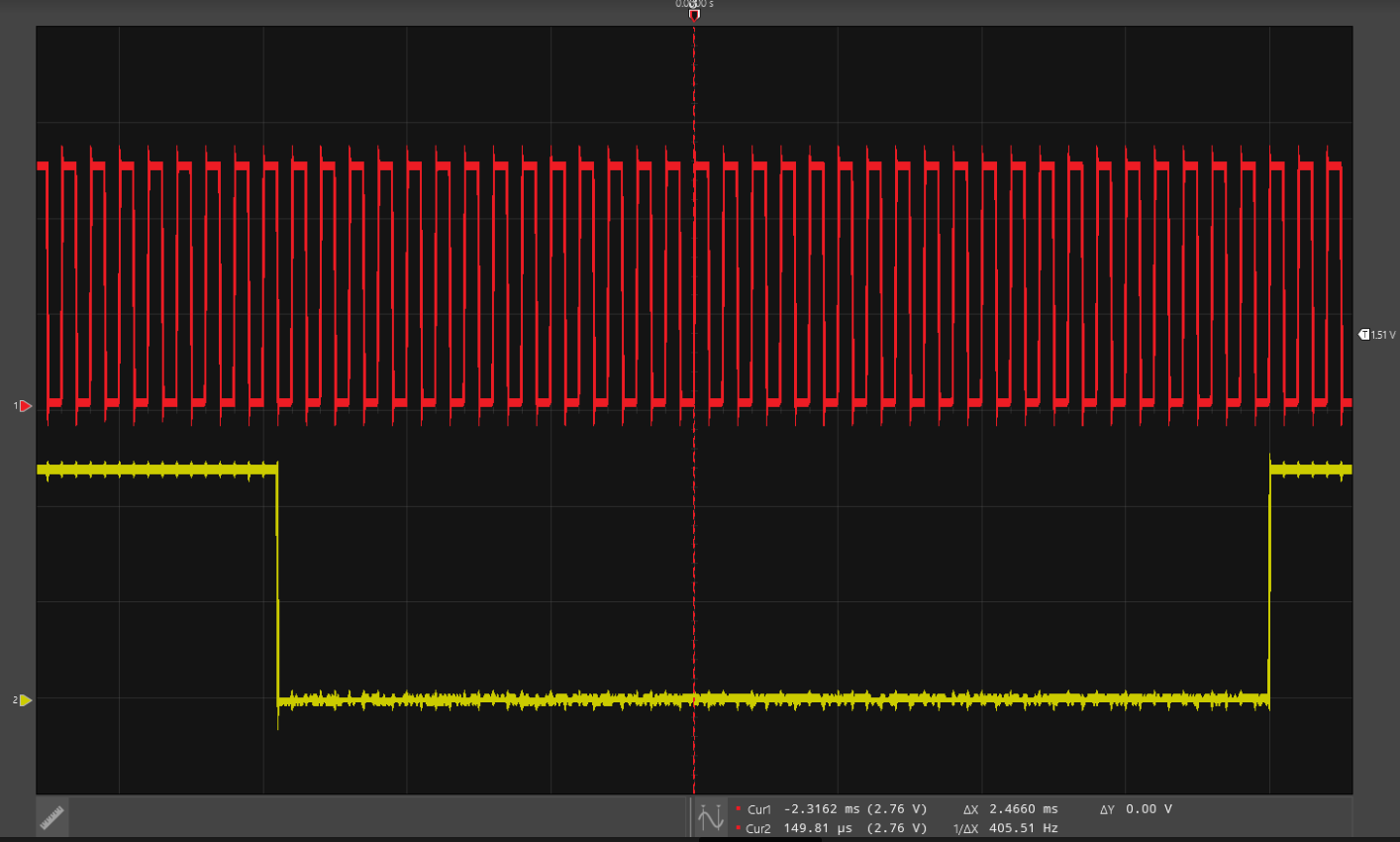
In Figure 3 (below) a 100 Hz square wave excites the input pin. The latency varies a greater magnitude with a higher input frequency. The output pin (yellow) is seen to miss transitions on the input pin between 5 and 9 times before the output pin switches states. The for loop is seen to be much slower than the time between transitions on the input.

**

*Figure 3: Input (red) to Output (yellow) arbitrary latency (Min delay, 100 Hz input)*

**

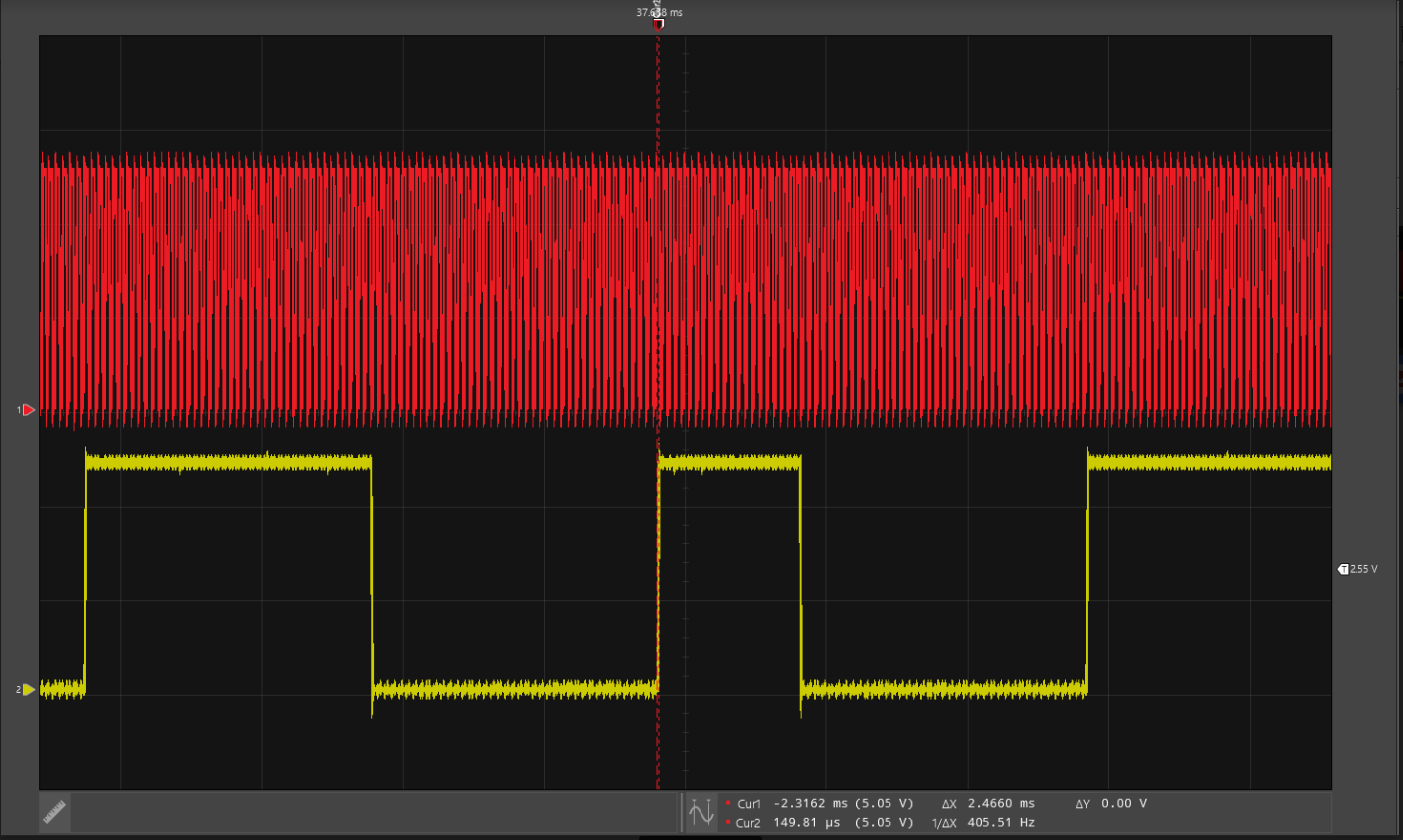
*Figure 4: LCD Displaying Minimum Delay (200 mS)*

****

*Figure 5: Maximum Delay, 10 Hz; Input (red) Output (yellow)*

Figure 5 (above) shows a 10 Hz input while the delay value is set to 200 mS. Figure 2 above also shows a 10 Hz input (using a smaller delay) and the output follows the input. The output in Figure 5 is seen to take many clock cycles before switching. The for loop now takes about 10 times longer than it did previously. Therefore there should be approximately 10 times the transitions in Figure 5 than in Figure 2 (with uncertainty described in Figure 3 description).

Figure 6 (below) shows a 100 Hz square wave on the input pin and the output pins response. Although the output is seen to only transition 5 times, the input pin transitions 100’s of times. To accurately show the output pins state, the oscilloscope horizontal resolution had to be decreased so far that the 100 Hz input cannot be counted. The number of periods between output transitions is estimated between 500 and 900 times based on the system’s response in Figure 3.

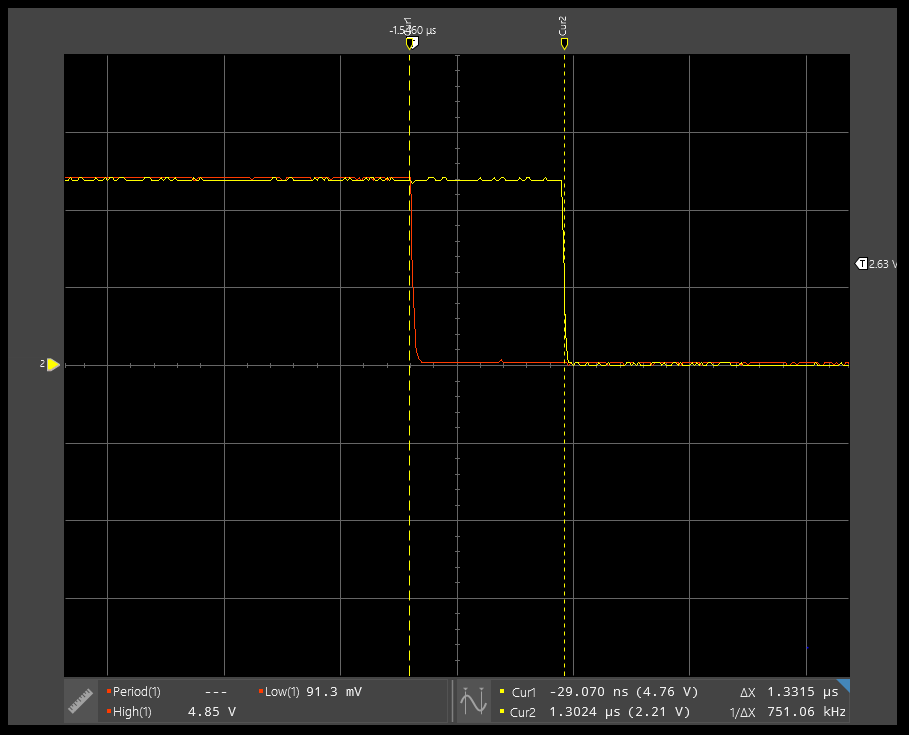
****

*Figure 6: Maximum Delay, 100 Hz; Input (red) Output (yellow)*

**This whole section was very good, but I did also want a tabular view of latency data.**

**Part II:**

The objective of part 2 was to repeat the experiments of part 1 but instead of polling using an interrupt on the input pin. Rather than the state of the output having to wait until a maximum of one complete iteration of the for loop is complete, the output will transition as soon as the interrupt service routine (ISR) code is executed (approximately 1 clock cycle later). Prior to the code written inside the ISR is executed, the CPU must complete a healthy ‘preparation’ to go into the interrupt. The number of cycles required to complete the preparation is evaluated in the Figure 3 below.



*Figure 7: Fixed Latency from Input (red) to Output (yellow) (1.5 uS nominal)*

Figure 7 shows a 1.3 uS delay between the input and the output, the value of the delay varied slightly but had a nominal delay of 1.5 uS. Given the system clock frequency of 24 MHz, there are 36 clock cycles in between input and output. The output signal began to miss clock transitions at a frequency of approximately 200 KHz when using interrupts this is a much faster response than seen in the previous method (polling). Good, but also looking for table.

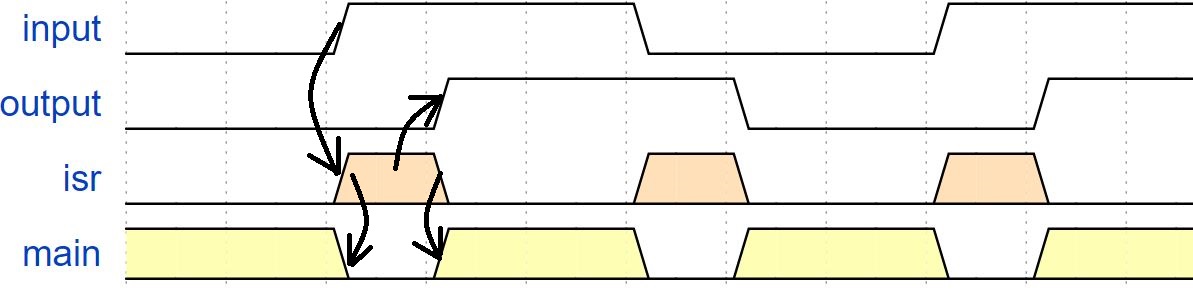
Animation on the LCD display ceased at a much lower frequency. This was because the CPU was spending all of its time in the ISR instead of the for loop where the LCD gets updated.

1. **Analysis**

Part 1 executed a polling method which only updated the output pin when the line to update the output was executed in the for loop. If the input were to change immediately after the update output pin instruction was executed, an entire iteration of the for loop would need to be executed before updating the state of the output pin. This lead to longer and variable delays between updating the output pin and the input pin trigger. Essentially, both parts of the program had the same “importance” to the CPU.

Part 2 executed an interrupt as opposed to polling. When an interrupt is executed, the CPU goes through a fixed set of instructions (Load ‘magic number’ to LR, push specific registers to stack). Immediately after this process is complete, the CPU proceeds to execute the instructions within the interrupt service routine. The instructions in the ISR update the output pin (SW2\_State) based on the state of the input pin (SW2). Since a fixed set of instructions (with a known CPI) are executed before the ISR is ran, the latency between input and output pin will remain constant as the ISR section has a higher priority to the CPU.

CyDelay executes a tight ‘for’ loop with a fixed number of instructions and a known number of clock cycles taken to execute each iteration. CyDelay doesn’t produce a fixed delay because clock cycles are being ‘stolen’ from the for loop to execute the interrupt.



*Figure 8: Timing Diagram Showing Causality*

The CPU of our PSoC 5LP is running at 24 MHz. The example ISR code is 20 cycles long. With a measured ISR latency of ~1.5 µs, this translates to approximately 36 cycles, which is near the estimated ISR example code in terms of real world delay.

**Appendix:** Commented Source Code

Code for parts 1 and 2 are shown below. The sections that were used in part 1 but not in part 2 or vice versa have been noted using a comment above the instruction.

#include <project.h>

#include <stdio.h>

// Unsigned Fixed Point Macros, UQm.n

#define FIX\_n (16) // fixed point 'n' value

#define FIX\_m (16) // fixed point 'm' value

#define FIX\_N (FIX\_n + FIX\_m) // total bits in UQm.n

#define FIX\_FACTOR (1 << FIX\_n ) // fixed point fraction factor (2^n)

#define FIX\_\_0\_5 (1 << (FIX\_n-1)) // 0.5 expressed in UQ16.16

#define FIX\_\_1\_0 (1 << FIX\_n ) // 1.0 expressed in UQ16.16

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// CY\_ISR ( ADDRESS ) is executed on each change seen on the input pin. Write the value of

// SW2\_State (output) based on the value of SW2 (input)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CY\_ISR ( SW2\_HANDLER )

{

SW2\_State\_Write(SW2\_Read());

SW2\_INT\_ClearPending();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// fix2int takes a fix point data type and rounds it to the nearest integer, returns integer

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint32\_t fix2int( uint32\_t fix )

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint32\_t container;

container = (fix + FIX\_\_0\_5) >> FIX\_n; //

return container;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// fix2double casts a fix point data type to a double, returns double

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double fix2double( uint32\_t fix )

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double doubleContainer;

doubleContainer = (double)fix/FIX\_FACTOR;

return doubleContainer;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// double2fix casts a double data type to a fixed point data type.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint32\_t double2fix( double x )

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint32\_t container;

container = (uint32\_t)((x \* FIX\_FACTOR) + 0.5);

return container;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

void fix2decimalstr( uint32\_t x, char \*str, int dotn )

{

int i;

uint64\_t lx;

int len;

// Use 64 bit int to avoid overflow

lx = x;

// Multiply by 10^dotn, to shift all fractional decimal into integer part

for (i = 0; i < dotn; ++i)

{

lx \*= 10;

}

// Get the integer part via rounding by adding half, and right shifting n

lx += FIX\_\_0\_5;

lx >>= FIX\_n;

x = (uint32\_t) lx;

// Print the number, but without decimal point

sprintf( str, "%d", (int) x );

len = strlen(str);

// Insert the decimal point in the correct location

// First move all of the last 'dotn' characters to the right to make space

str[len+1] = '\0';

for (i = 0; i < dotn; ++i)

{

str[len-i] = str[len-i-1];

}

str[len-dotn] = '.';

}

int main()

{

int k; // Current position of bouncing box (relative to LCD)

int direction; // +1 --> move right, -1 --> move left

char num\_str[17]; // Array to render the value of rate as a string

char msg\_str[17]; // Entire message, to write to LCD

int sw2; // Holds current switch state

int sw3; // Holds current switch state

int sw2\_prev; // Holds previous state, for button down detection

int sw3\_prev; // Holds previous state, for button down detection

CyGlobalIntEnable; // Enables global interrupts

// Enables SW2\_INT (comment the line below for Part 1)

SW2\_INT\_StartEx(SW2\_HANDLER);

uint32\_t delay = 20 \* FIX\_\_1\_0; // UQ16.16

uint32\_t llim = 20 \* FIX\_\_1\_0; // Upper limit of delay expressed in UQ16.16

uint32\_t ulim = 200 \* FIX\_\_1\_0; // Upper limit of delay expressed in UQ16.16

uint32\_t incr = double2fix( 10.0/3.0 ); // Represent 3.33... in fixed point

LCD\_Display\_Start(); // Start the LCD component

k = 0; // Initialize position

direction = 1; // and direction

sw2 = sw3 = sw2\_prev = sw3\_prev = 1; // Initialize switch states to open

// Loop forever

for(;;)

{

// Convert current delay to a string, with 3 decimal places precision

fix2decimalstr(delay, num\_str, 3);

// Generate composite message string

sprintf( msg\_str, "Delay=%7s ms", num\_str );

// Render current state onto the display

// Top line is bouncing square

// Bottom line is current delay

LCD\_Display\_ClearDisplay(); // Must clear entire display before new rendering

LCD\_Display\_DrawHorizontalBG(0, k, 1, 5); // Draw the box on top line

LCD\_Display\_Position(1, 0); // Position on bottom line

LCD\_Display\_PrintString(msg\_str); // Print the msg on bottom line

// comment this line out for Part 2. Input was renamed to SW2 and output renamed to SW2\_State

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DIG\_OUT\_PIN\_Write(DIG\_IN\_PIN\_Read());

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CyDelay(fix2int(delay)); // Delay based on integer part of delay

k += direction; // Compute new position

if (k == 15)

direction = -1;

else if (k == 0)

direction = 1;

// Lines below get commented out during Part 2 of the assignment. In part 1, these lines

// are the 'polling' the input pins. Part 2 uses an ISR instead of polling

sw2 = SW2\_Read(); // Get current switch state

sw3 = SW3\_Read(); // Get current switch state

if (sw2 == 0 && sw2\_prev == 1) // If Switch 2 button down event, decrease delay

delay -= incr;

if (sw3 == 0 && sw3\_prev == 1) // If Switch 3 button down event, increase delay

delay += incr;

sw2\_prev = sw2; // Update previous sw2 state

sw3\_prev = sw3; // Update previous sw3 state // end commenting for part 2

// Saturate delay to upper and lower limits

if (delay > ulim) delay = ulim;

if (delay < llim) delay = llim;

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Expected** | **Points** | **Pts. Available** |
| Cover sheet |  | 0.5 | 0.5 |
| Table 1 - latency range vs. animation rate when sq. wave freq=10Hz, no intr | Approximately 0 ms to animation period, for several animation periods | 0.9 | 1 |
| Table 2 - latencies at varying sq. wave rates, with intr | About 1.6 usec, independent of square wave rate, and independent of animation rate. | 0.9 | 1 |
| During Part 2, at what rate did animation cease altogether? Why | When the edge toggle period is equal to the time required by the ISR then the foreground routine will fail to get any cycles, and animation ceases. | 1 | 1 |
| Screen captures of input and output waveforms, with and without using interrupts |  | 0.9 | 1 |
| Describe response in Part 1, without intr, and explain | Latency depends on delay in the forever loop | 0.8 | 1 |
| Describe response in Part 2, with intr, and explain | Latency is constant and short, independent of the delay in the forever loop | 0.9 | 1 |
| Why did CyDelay function fail to produce constant delay in main loop? | The period between input signal edges approached the amount of time required to enter the ISR, run the ISR, and return from the ISR. | 1 | 1 |
| Timing diagram |  | 1 | 1 |
| BONUS: How many cycles, clock rate? Does latency make sense? |  | 0.2 | 0 |
| main.c file, fully commented and formatted |  | 1.5 | 1.5 |
| **TOTAL** |  | **9.6** | **10** |