Design Assignment 4: Fixed Point Numbers, Real Time Response, and Interrupts



Alexis Adie and Madison Mastroberte

ELC 411-01: Embedded Systems

Submitted:

November 9th 2017

Design Assignment 4: Fixed Point Numbers, Real Time Response, and Interrupts

Alexis Adie and Madison Mastroberte

Department of Electrical and Computer Engineering

The College of New Jersey

2000 Pennington Road, Ewing, NJ 08618, USA

(adiea1, mastrom7)@tcnj.edu

# Introduction

The purpose of this experiment was to implement software that could quickly react to changing inputs. Two tasks with their own timing requirements were required to be carried out by the students to display the challenges in managing more than on task. The second part of the experiment had the group use an interrupt service routine (ISR) to carry out the multiple events with the desire to learn more about the benefits of their implementation.

# II. Results

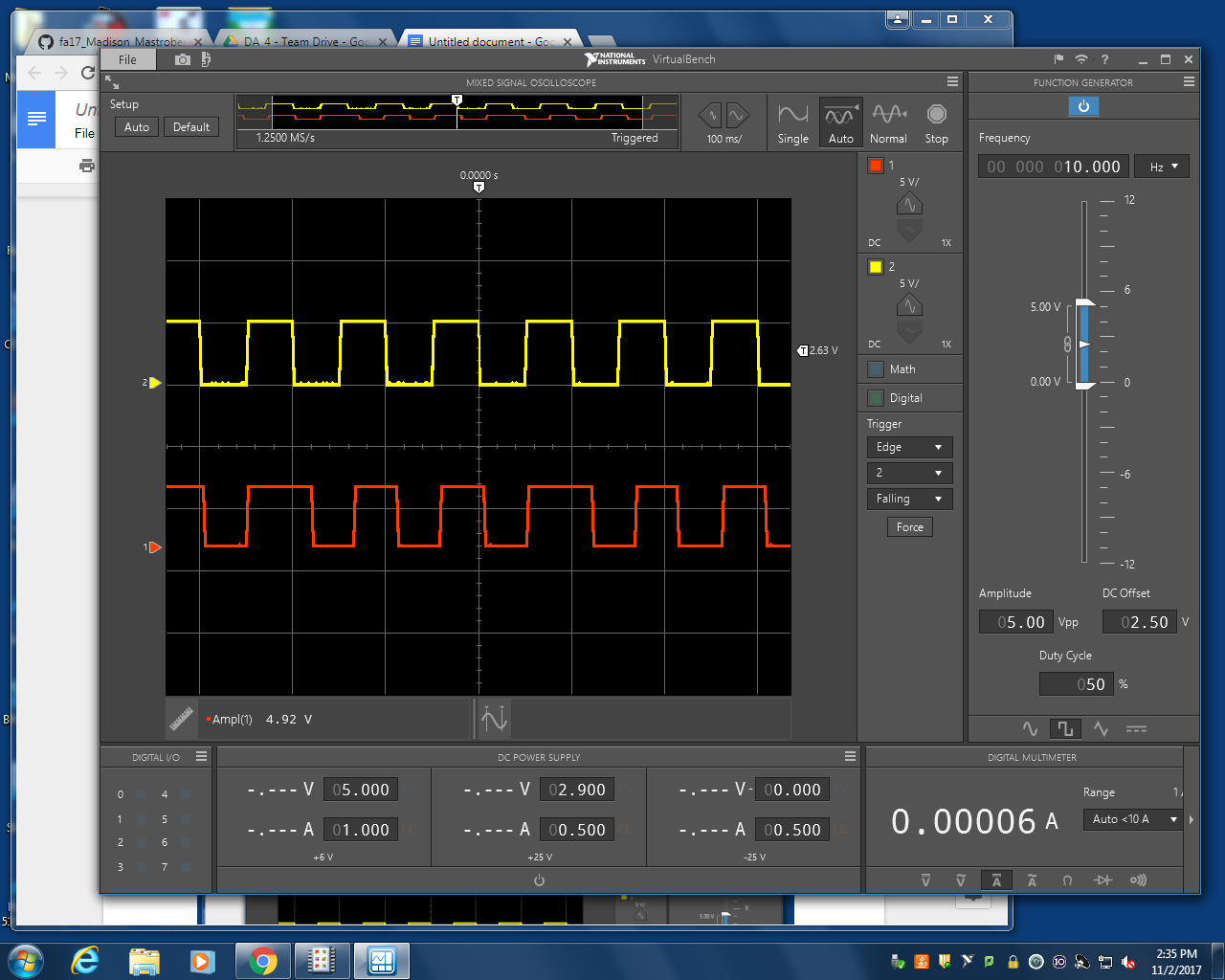
*A. Real Time Response and Waveforms*

**Table 1.** Latency range for a 10 Hz wave frequency with various animation rates without interrupts.

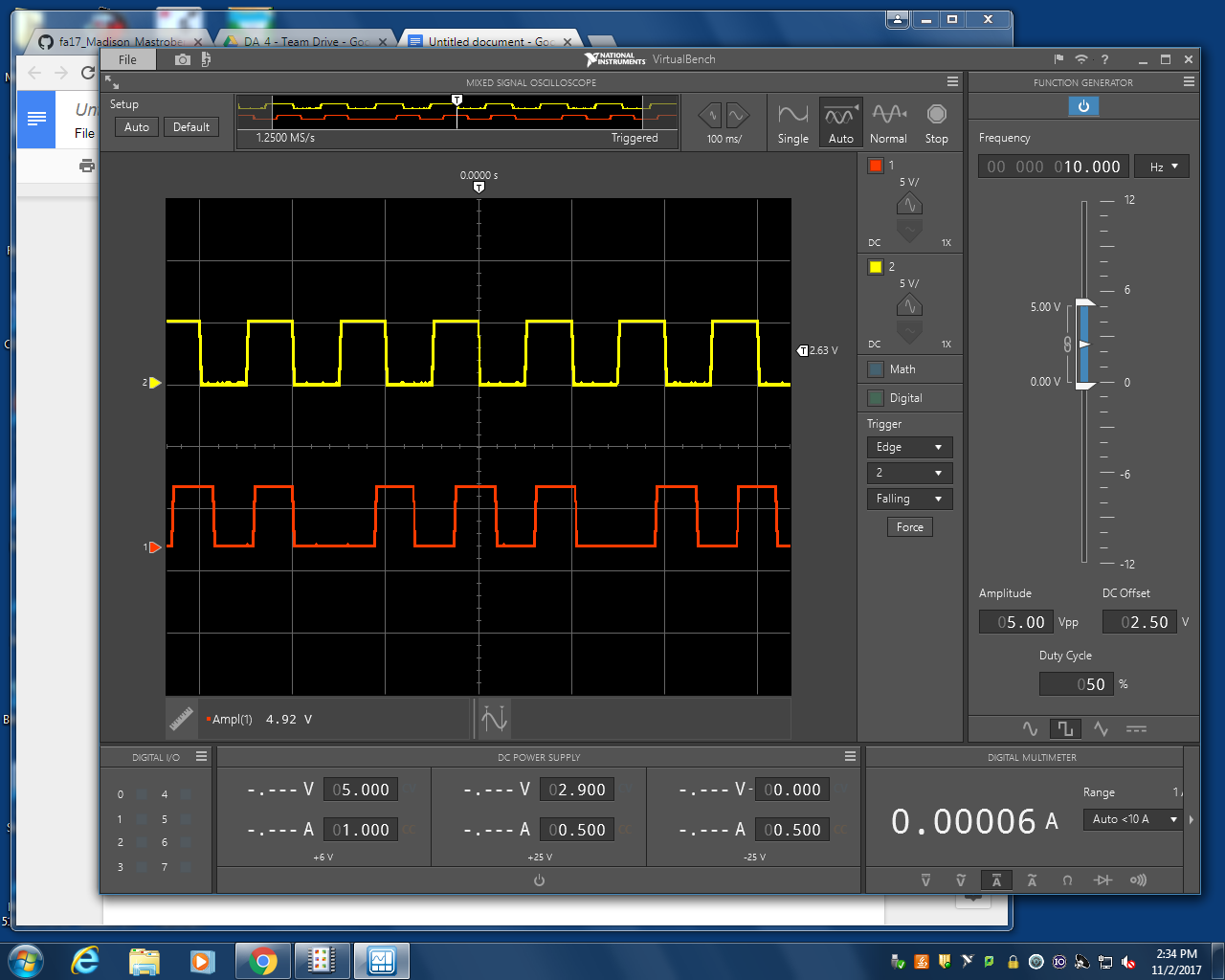
|  |  |
| --- | --- |
| Animation Rate | Estimated Latency |
| 20 ms | 21.5 ms |
| 40 ms | 37.5 ms |
| 60 ms | 82.2 ms |

**Table 2.** Latency range for various animation rates with interrupts.

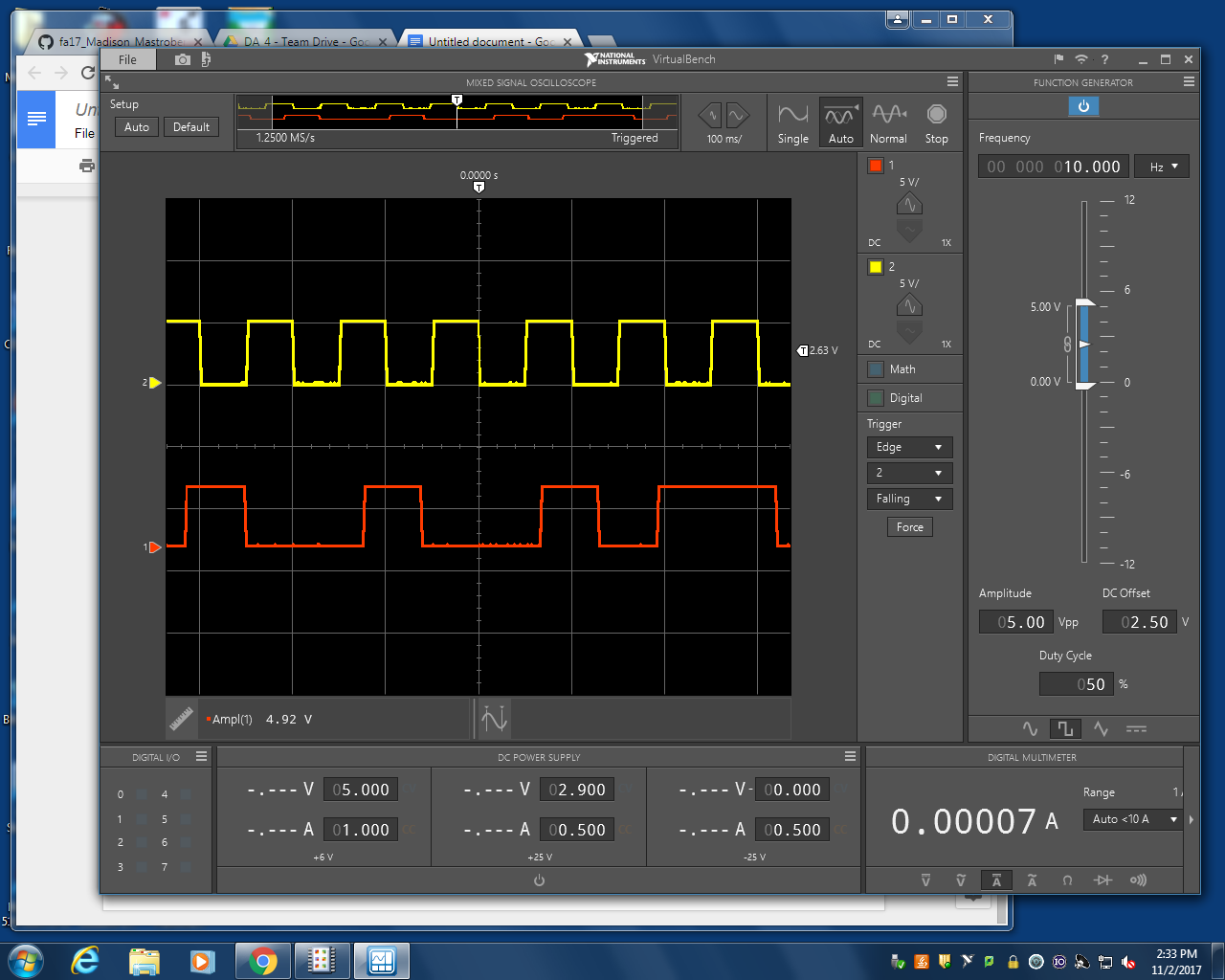
|  |  |
| --- | --- |
| Frequency | Latency |
| 10 Hz | 0.0 ms  value should be in usec, about 1.6 usec for all cases |
| 100 Hz | 0.0 ms |
| 100 kHz | 11.3 ms |
| 110 kHz | 12.8 ms |
| 150 kHz | 15.5 ms |



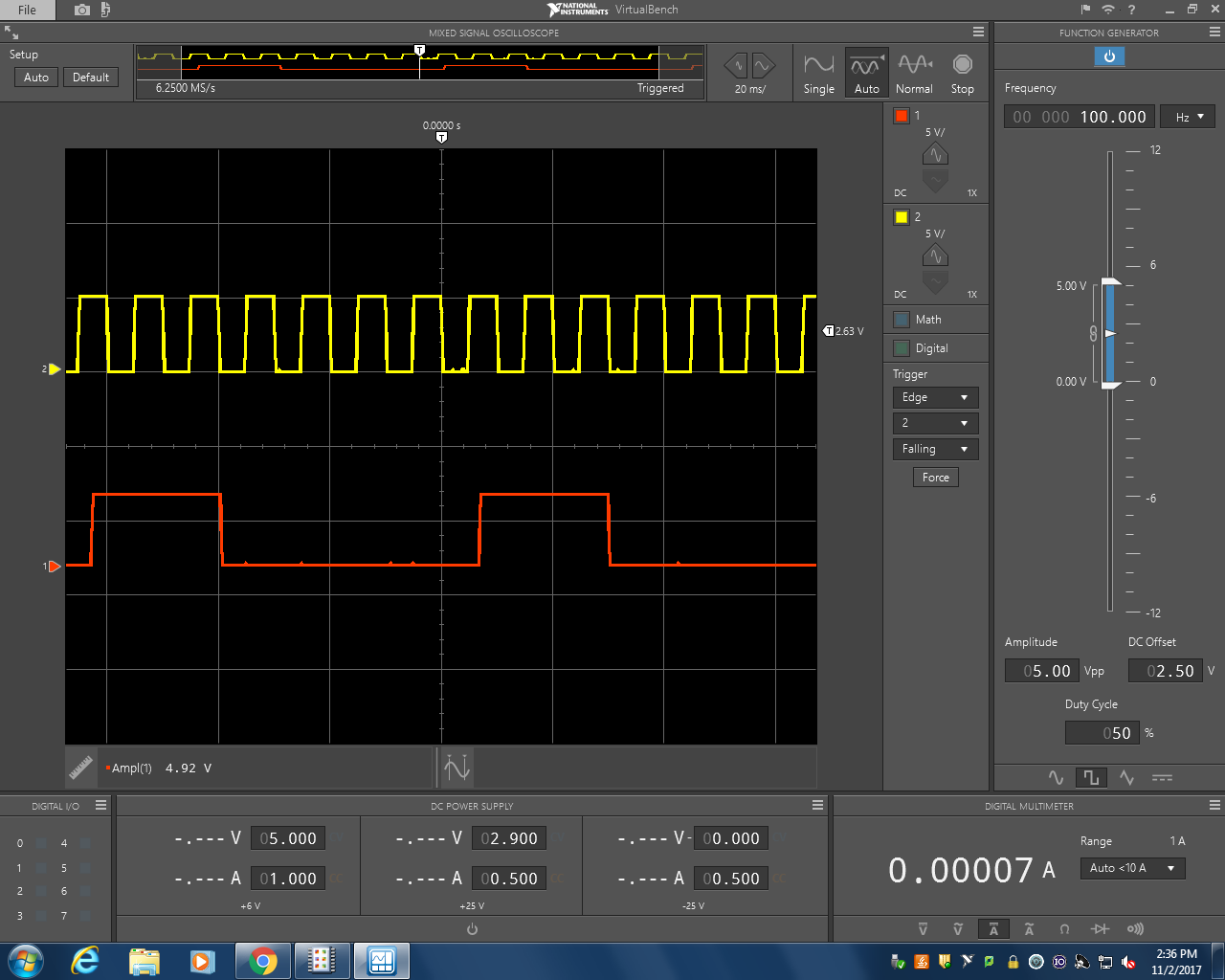
**Figure 1.** Input (yellow) and output (red) square waveforms at 10 Hz with an animation delay of 20 ms [100 ms/div | 5V/div].



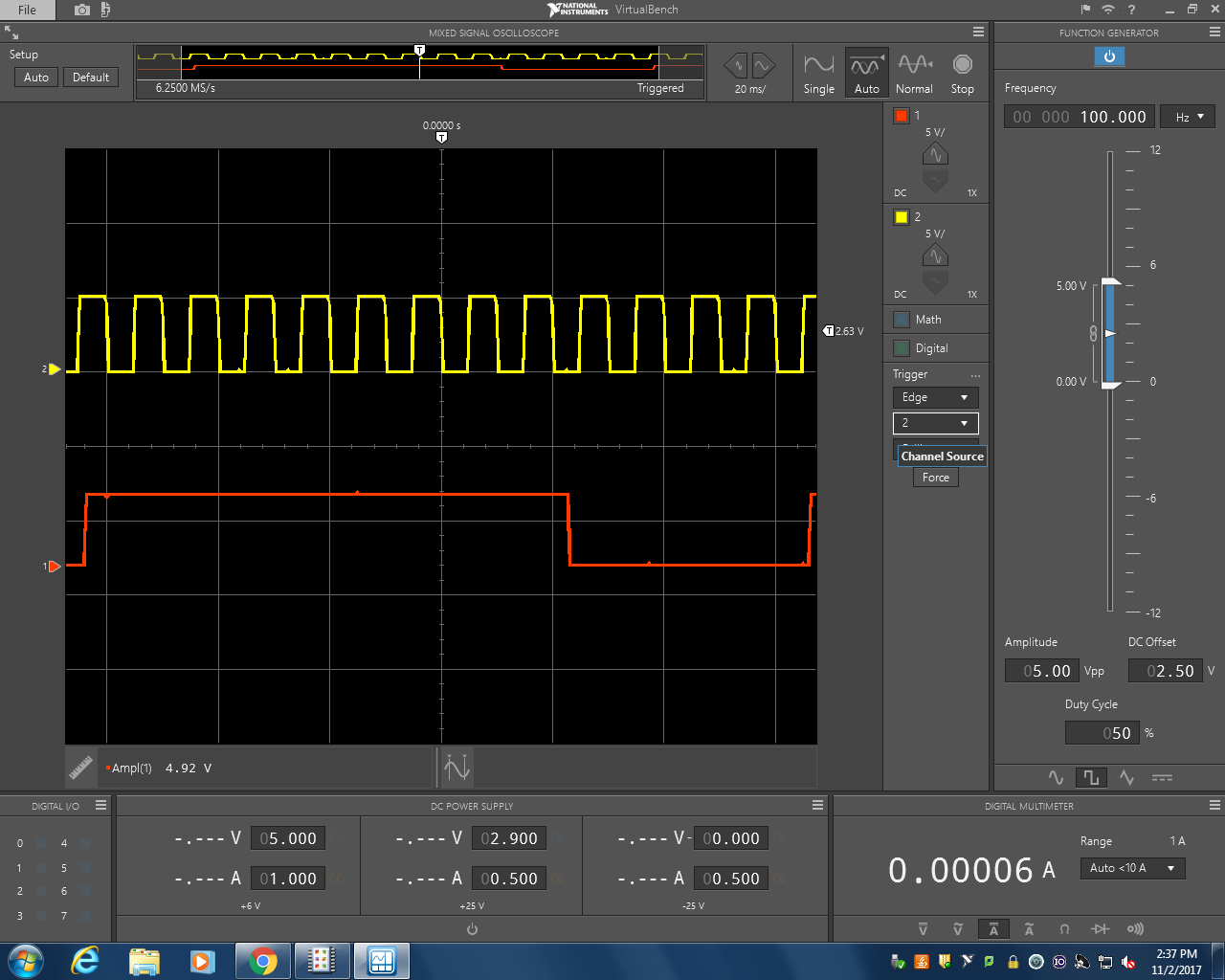
**Figure 2.** Input (yellow) and output (red) square waveforms at 10 Hz with an animation delay of 40 ms [100 ms/div | 5V/div].



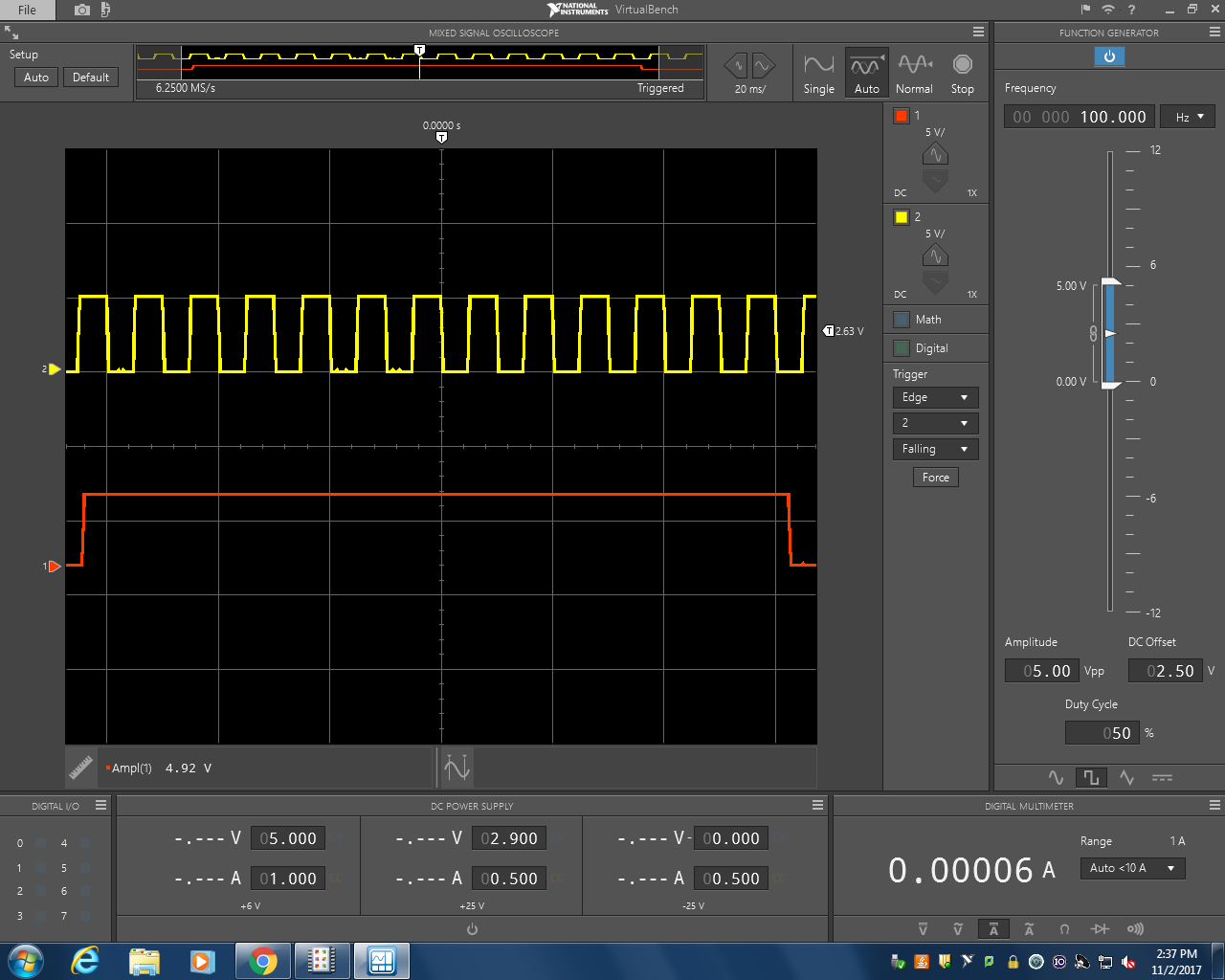
**Figure 3.** Input (yellow) and output (red) square waveforms at 10 Hz with an animation delay of 60 ms [100 ms/div | 5V/div].



**Figure 4.** Input (yellow) and output (red) square waveforms at 100 Hz with an animation delay of 20 ms [20 ms/div | 5V/div].

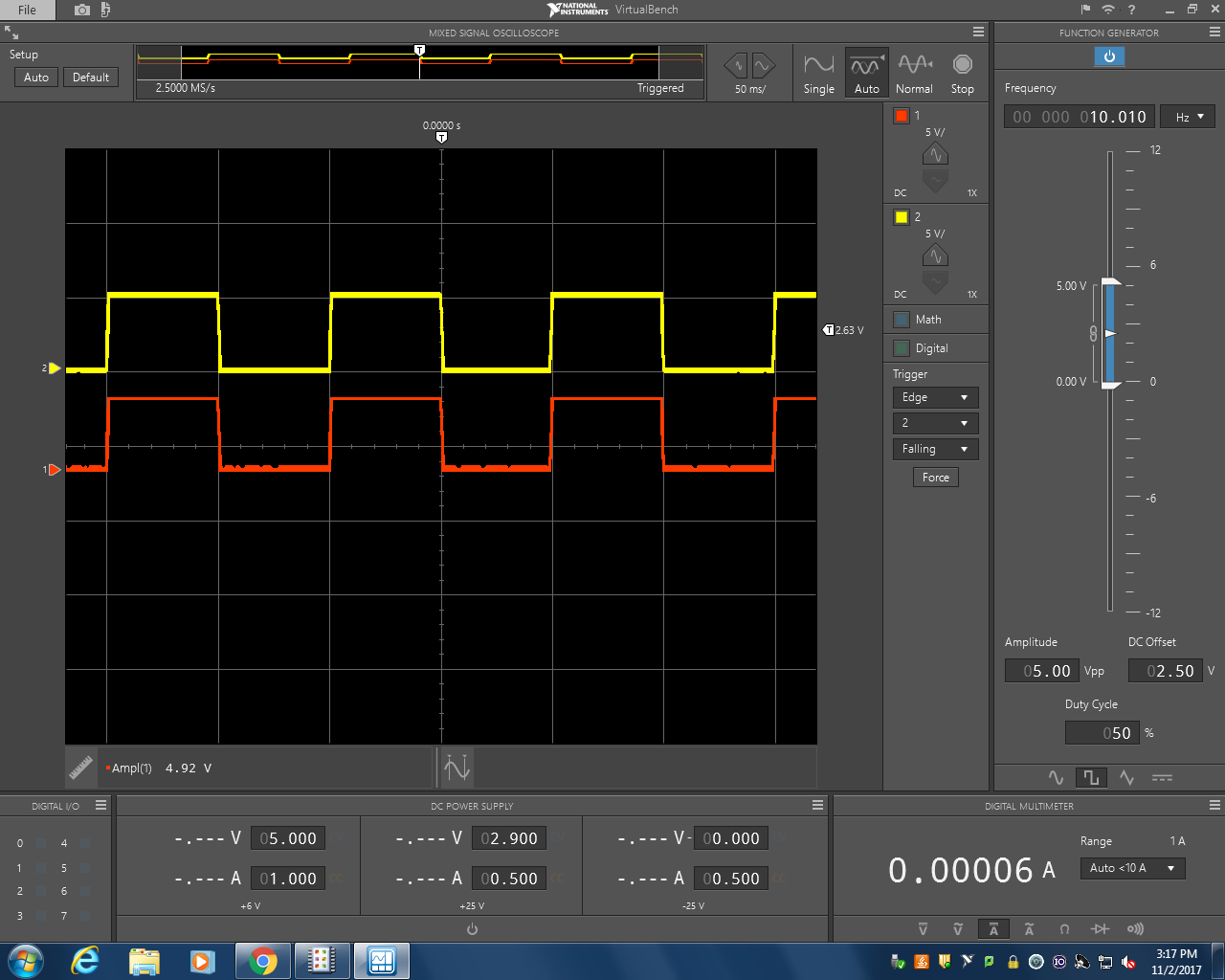


**Figure 5.** Input (yellow) and output (red) square waveforms at 100 Hz with an animation delay of 40 ms [20 ms/div | 5V/div].

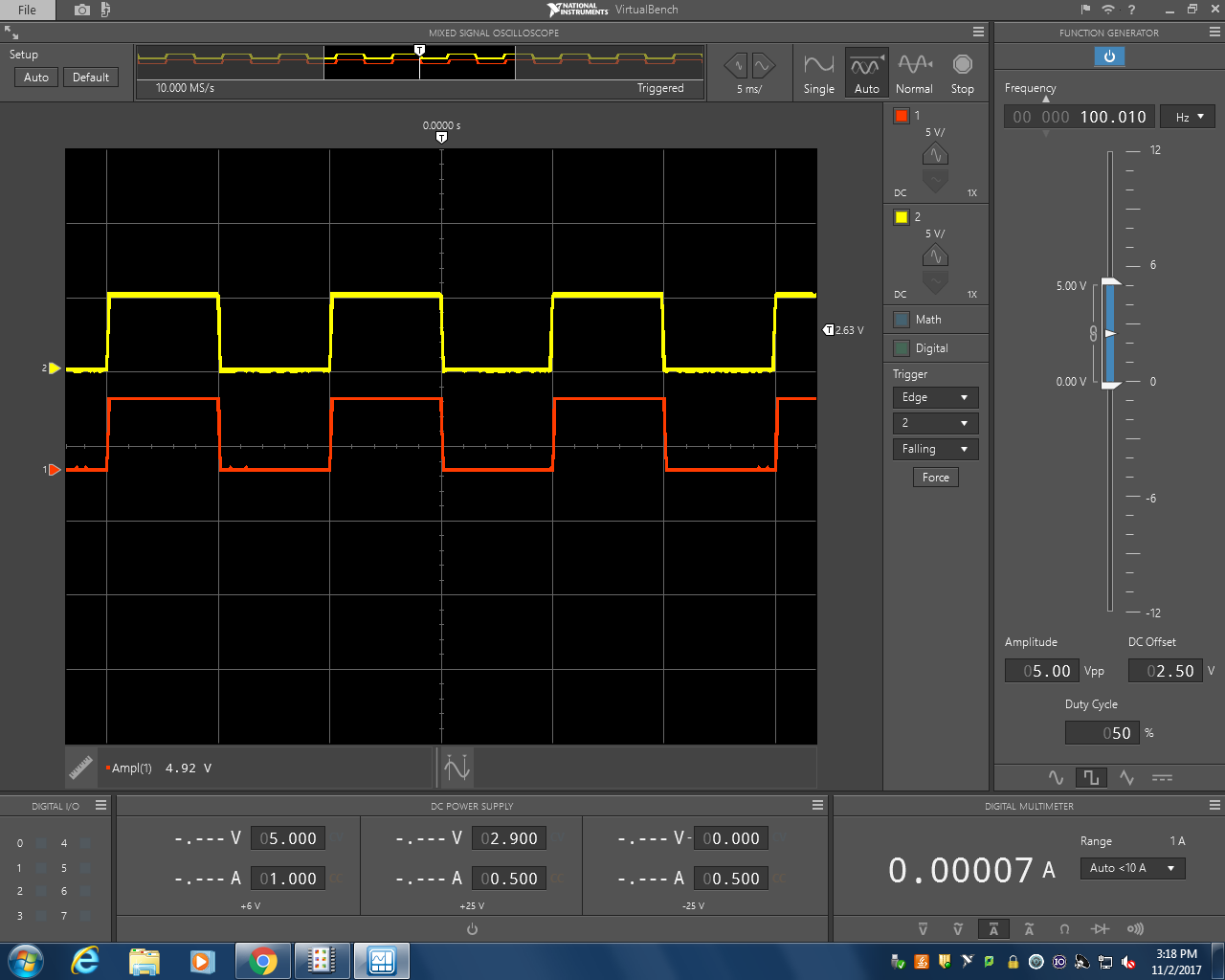


**Figure 6.** Input (yellow) and output (red) square waveforms at 100 Hz with an animation delay of 60 ms [20 ms/div | 5V/div].

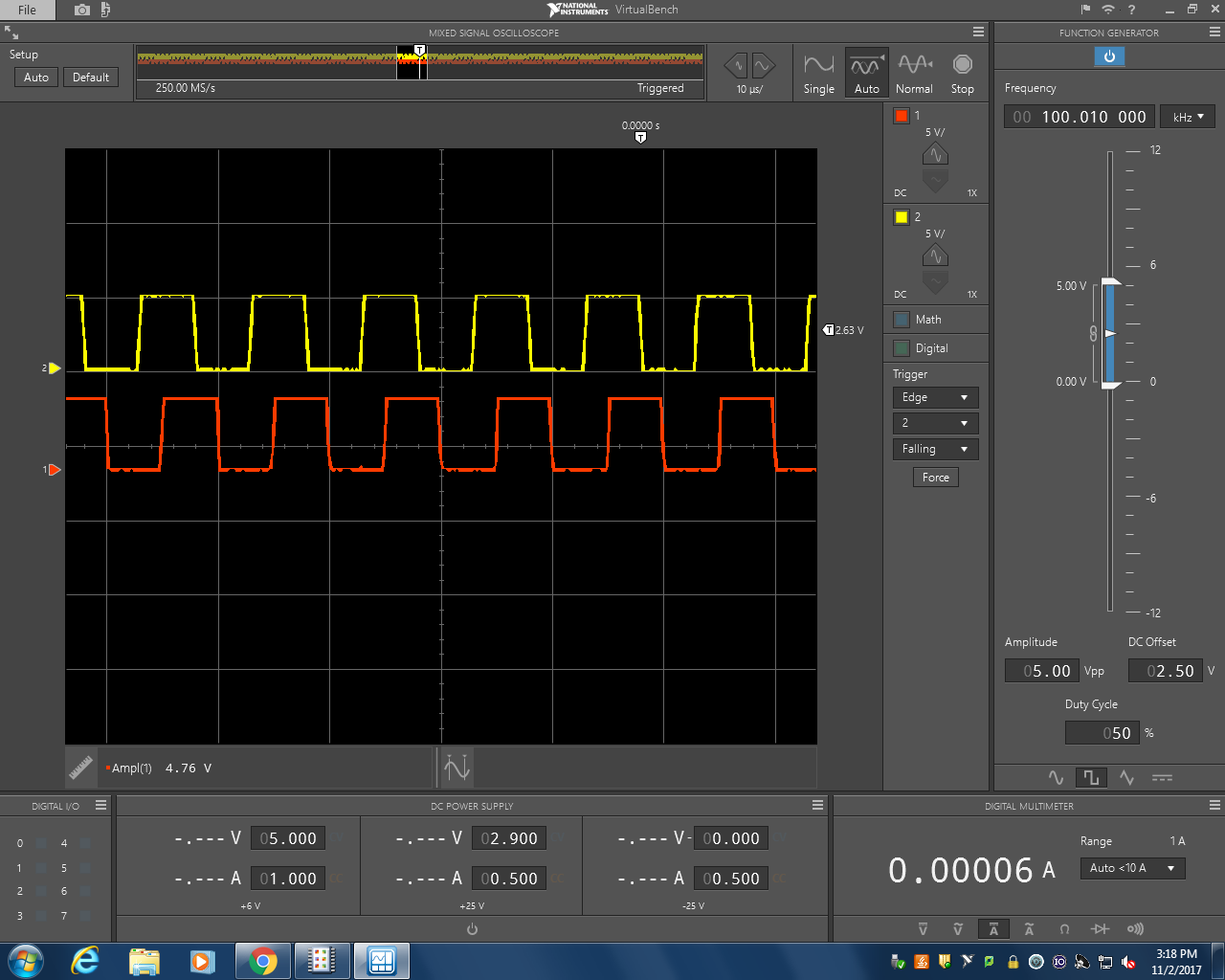
*B. Interrupts*



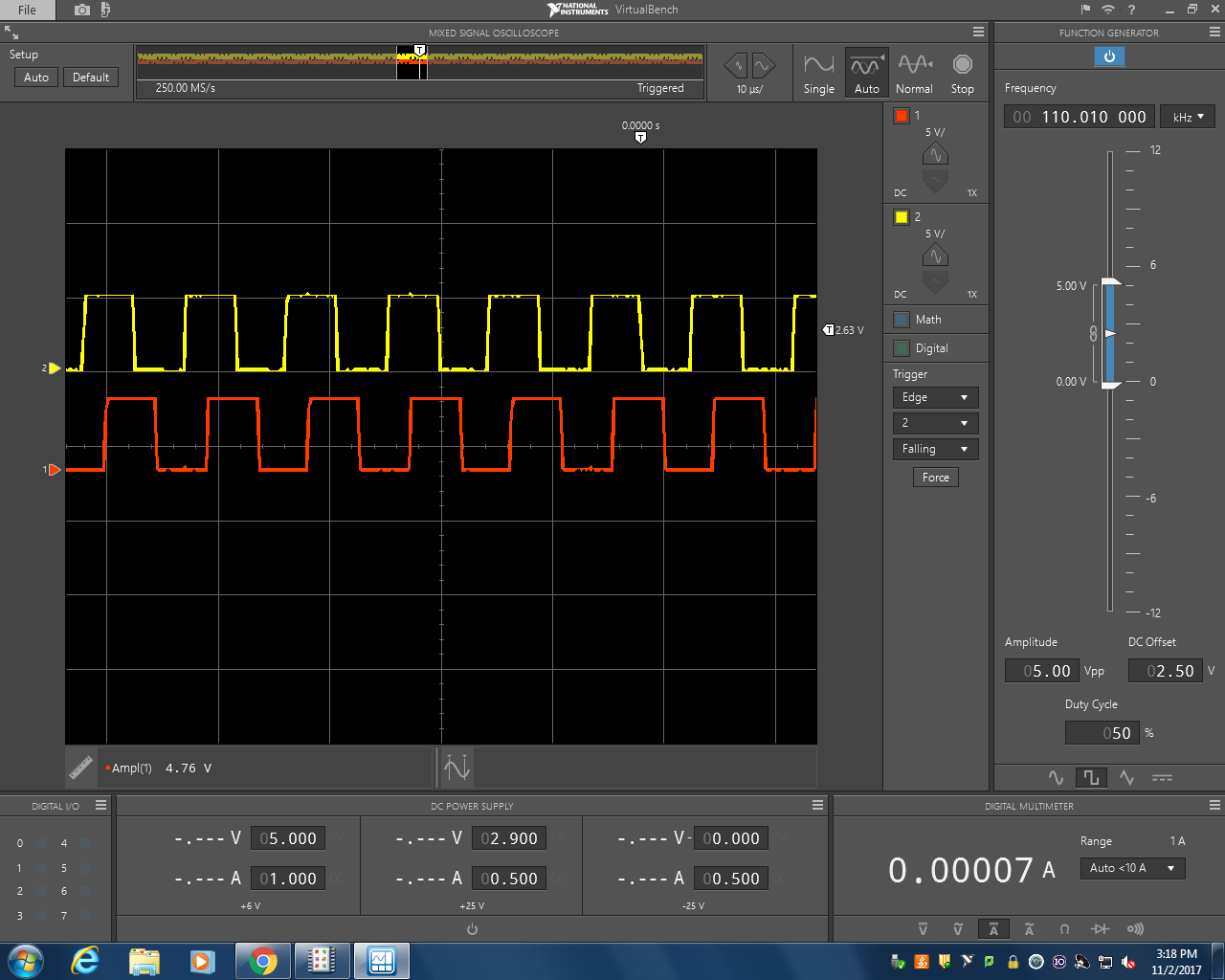
**Figure 7.** Input (yellow) and output (red) square waveforms at 10 Hz with interrupts [50 ms/div | 5V/div].



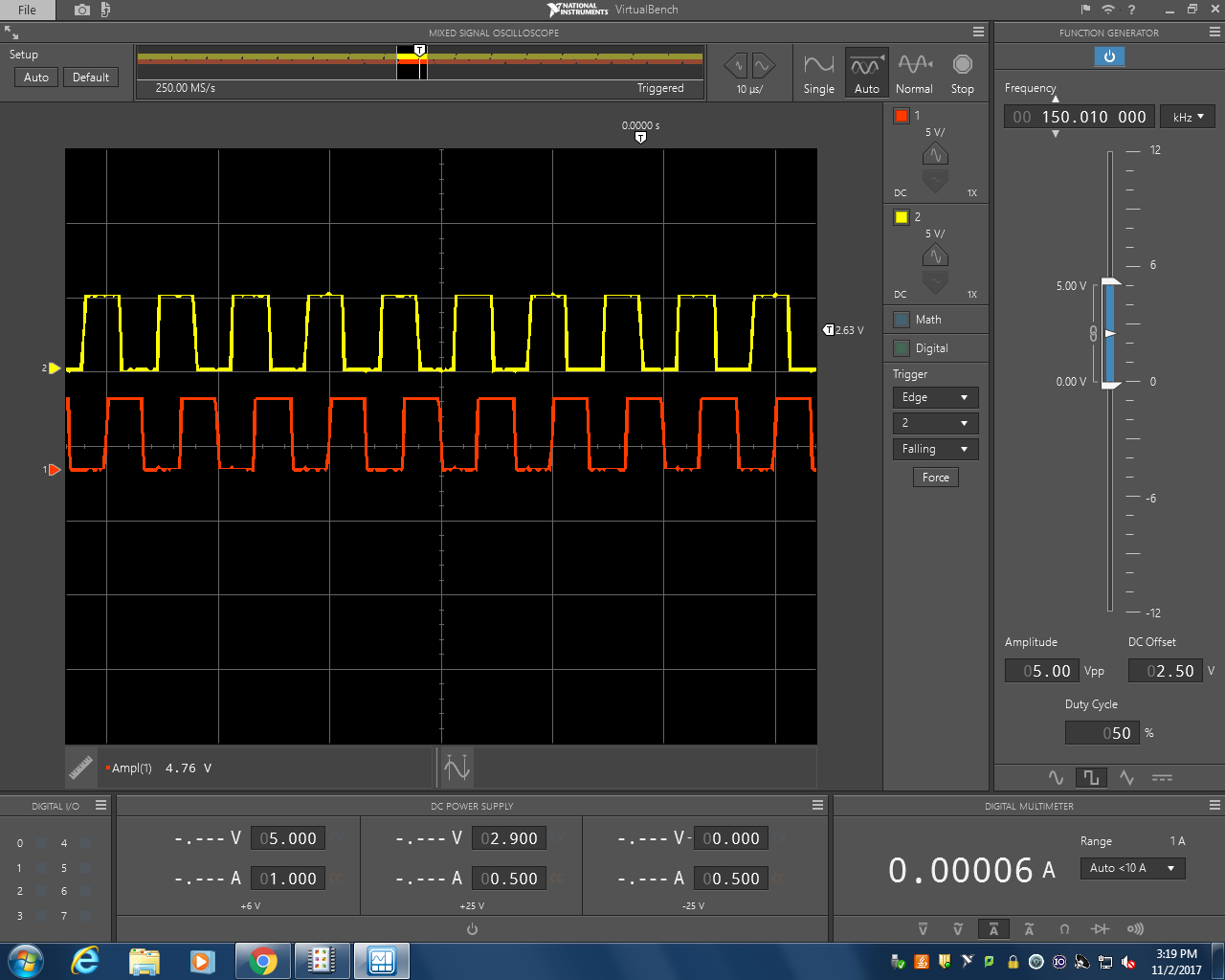
**Figure 8.** Input (yellow) and output (red) square waveforms at 100 Hz with interrupts [50 ms/div | 5V/div].



**Figure 9.** Input (yellow) and output (red) square waveforms at 100 kHz with interrupts [50 ms/div | 5V/div].



**Figure 10.** Input (yellow) and output (red) square waveforms at 110 kHz with interrupts [50 ms/div | 5V/div].



**Figure 11.** Input (yellow) and output (red) square waveforms at 150 kHz with interrupts [50 ms/div | 5V/div].

# III. Discussion

i. Animation Failure

In part two, the animation failed at any frequency over 150kHz. This failure was due to the interrupt period approaching the time that the ISR requires to start, run and return. limitations in the hardware. The program was going through too many iterations for the board to update.

1. System Response without Interrupts

The system response in part 1 was as expected. As the delay on the LCD screen got larger, the output represented by DIG\_OUT\_PIN received a greater delay. However, the actual latency measured was never completely accurate with the error ranging from 7.5-37%. The error seemed to increase as the delay was increased.

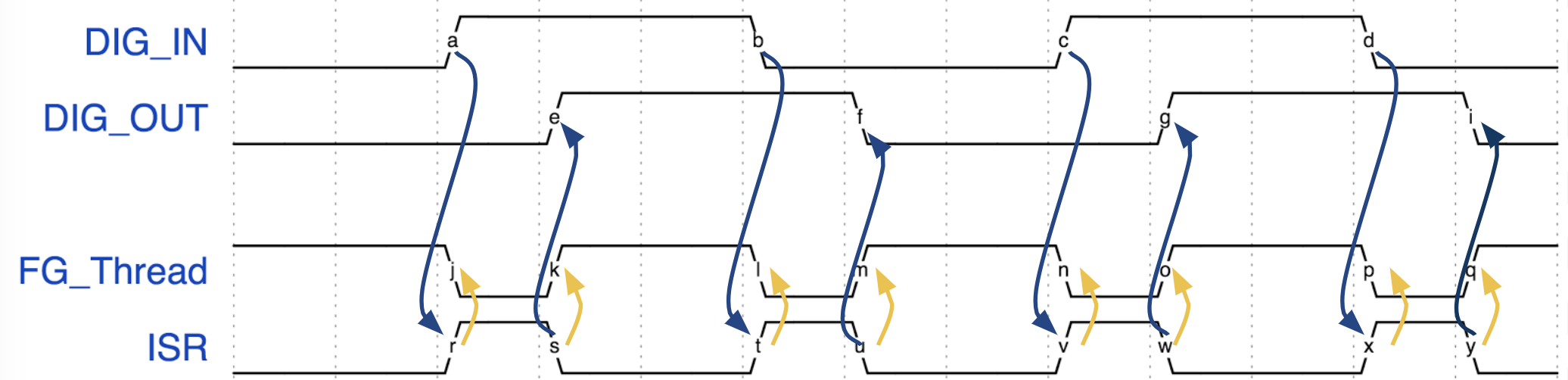
1. System Response with Interrupts

The system response with interrupts showed the benefit of including them in a design. At the initial frequencies the latencies recorded were zero. As the frequency was increased, however, there were latencies present. These findings can be explained by the CyDelay function, No, I would not expect so!.

1. CyDelay Function Failure

As the program went through more iterations, the CyDelay function began to “steal”cycles from the isr, causing latencies at higher frequencies. No – the converse is true – the ISR was stealing cycles from the CyDelay function!

1. Timing Diagram

****

**Figure 12.** Timing diagram for ISR and foreground service routines, with dependencies.

# IV. CONCLUSION

The purpose of this experiment was to introduce students to the use of interrupts as well as fixed-point integers in order to better implement a program design. The latency expected while a computer completes two or more tasks was reduced greatly with the implementation of an interrupt. By learning this concept, the team will be able to produce more efficient code in the future.

# VI. Appendix

**Madison Mastroberte’s Code**

#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 Qm.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

// Procedure:

// fix2int - round a fixed point value to the nearest integer

// Inputs:

// fix - fixed point value to round

// Return value:

// result of rounding the fixed point value to nearst integer, in uint32\_t container

uint32\_t fix2int( uint32\_t fix )

{

//Adds 0.5 in fixed point and discards the 16 fractional bits to zero

fix = (fix + FIX\_\_0\_5)>> 16 ;

return fix;

}

// Procedure:

// fix2double - Convert a fixed point value to double precision

// Inputs:

// fix - value to convert

// Return value:

// double precision representation of fixed point value

double fix2double( uint32\_t fix )

{

//Adds 0.5 in fixed point and discards the 32 fractional bits to zero

fix = (fix + FIX\_\_0\_5) >> 16;

return fix;

}

// Procedure:

// double2fix - Convert a double precision value to fixed point, with rounding

// Inputs:

// x - value to convert

// Return value:

// fixed point approximation of the input value, in uint32\_t container

uint32\_t double2fix( double x )

{

//Adding 0.5 in floating point

x = (x + 0.5);

//Rounding fractional bits

//Casting explicitly

int y = (int)x;

y = y >> 16;

return y;

}

// Procedure:

// fix2decimalstr - convert fixed point value to decimal string

// Inputs:

// x - fixed point value to convert to string

// str - pointer to destination string

// dotn - desired decimal precision

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; /\* Enable global interrupts. \*/

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

DIG\_IN\_PIN\_Read() ;

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;

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

// Saturate delay to upper and lower limits

if (delay > ulim) delay = ulim;

if (delay < llim) delay = llim;

}

}

**Alexis Adie’s Code**

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 Qm.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  
  
// Procedure:  
// fix2int - round a fixed point value to the nearest integer  
// Inputs:  
// fix - fixed point value to round  
// Return value:  
// result of rounding the fixed point value to nearst integer, in uint32\_t container  
uint32\_t fix2int( uint32\_t fix )  
{  
 //n bits  
 float bitn= 10.0\*n;  
 fix =(int) (fix\*bitn+0.5)/bitn;  
   
 //fix= (int) (fix\*100+0.5)/100.0;  
   
 //the variable F would receive the value 2.468, not the rounded value of 2.47 which would   
 //be appropriate for most monetary uses. To assign the rounded result of 2.47 into variable R, we would use the expression  
  
 // R = (int) (F\*100+0.5) / 100.0;  
  
 // which would offset the decimal point two places by multiplying F by 100 (resulting in 246.8),   
 //then bump the value to 247.3, and then truncate the .3 portion by casting the value into integer   
 //form, and finally reposition the decimal point by dividing the result be 100.0. It is essential to w  
 //rite the value 100 in floating point notation (with the .0 attached) to prevent C from performing integer   
 //division which would corrupt the results.  
 //round(uint32\_t fix);  
  
}  
  
// Procedure:  
// fix2double - Convert a fixed point value to double precision   
// Inputs:  
// fix - value to convert  
// Return value:  
// double precision representation of fixed point value  
double fix2double( uint32\_t fix )  
{  
  
 fix= (int) (fix+0.5);  
   
}  
  
// Procedure:  
// double2fix - Convert a double precision value to fixed point, with rounding  
// Inputs:  
// x - value to convert  
// Return value:  
// fixed point approximation of the input value, in uint32\_t container  
uint32\_t double2fix( double x )  
{  
 x= (int) (x+0.5);  
}  
  
// Procedure:  
// fix2decimalstr - convert fixed point value to decimal string  
// Inputs:  
// x - fixed point value to convert to string  
// str - pointer to destination string  
// dotn - desired decimal precision  
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; /\* Enable global interrupts. \*/  
  
 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  
  
   
   
DIG\_IN\_PIN\_Read() ;  
DIG\_OUT\_PIN\_Write(DIG\_IN\_PIN) ;  
  
   
 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;  
  
 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  
  
 // Saturate delay to upper and lower limits  
 if (delay > ulim) delay = ulim;  
 if (delay < llim) delay = llim;  
 }  
}

**Commented and Debugged Code**

#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 Qm.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

extern uint32 volatile int\_count;

// Procedure:

// fix2int - round a fixed point value to the nearest integer

// Inputs:

// fix - fixed point value to round

// Return value:

// result of rounding the fixed point value to nearest integer, in uint32\_t container

uint32\_t fix2int( uint32\_t fix )

{

//Adds 0.5 in fixed point and discards the 16 fractional bits to zero

return (fix +FIX\_\_0\_5) >> FIX\_n;

}

// Procedure:

// fix2double - Convert a fixed point value to double precision

// Inputs:

// fix - value to convert

// Return value:

// double precision representation of fixed point value

double fix2double( uint32\_t fix )

{

//Adds 0.5 in fixed point and discards the 32 fractional bits to zero

fix = (fix + FIX\_\_0\_5) >> 16;

No – this just takes the integer part, and discards the fractional information.

The ideal equation is:

double x;

x = (double) fix / FIX\_FACTOR;

return x;

}

// Procedure:

// double2fix - Convert a double precision value to fixed point, with rounding

// Inputs:

// x - value to convert

// Return value:

// fixed point approximation of the input value, in uint32\_t container

uint32\_t double2fix( double x )

{

return (uint32\_t) (x \* FIX\_FACTOR + 0.5); // Good!

}

// Procedure:

// fix2decimalstr - convert fixed point value to decimal string

// Inputs:

// x - fixed point value to convert to string

// str - pointer to destination string

// dotn - desired decimal precision

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()

{

isr\_1\_Start();

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; /\* Enable global interrupts. \*/

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(;;)

{

// Code is not indented properly because TAB characters are used!

// 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 );

sprintf(msg\_str,"Cnt=%12lu",int\_count );

// 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

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

//DIG\_IN\_PIN\_Read() ;

// 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;

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

// 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 | 1 | 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. | 0.7 | 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. | 0.7 | 1 |
| Screen captures of input and output waveforms, with and without using interrupts |  | 1 | 1 |
| Describe response in Part 1, without intr, and explain | Latency depends on delay in the forever loop | 1 | 1 |
| Describe response in Prat 2, with intr, and explain | Latency is constant and short, independent of the delay in the forever loop | 0.7 | 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. | 0.7 | 1 |
| Timing diagram |  | 1 | 1 |
| How many cycles, clock rate? Does latency make sense? |  | 0 | 0 |
| main.c file, fully commented and formatted |  | 1.2 | 1.5 |
| **TOTAL** |  | **8.5** | **10** |