|  |  |
| --- | --- |
|  | School of Engineering |

|  |  |  |
| --- | --- | --- |
| **ELC 411-02** | **Embedded Systems** | **Fall 2019** |

**LAB REPORT**

**Design Assignment 4: Timers & Interrupts**

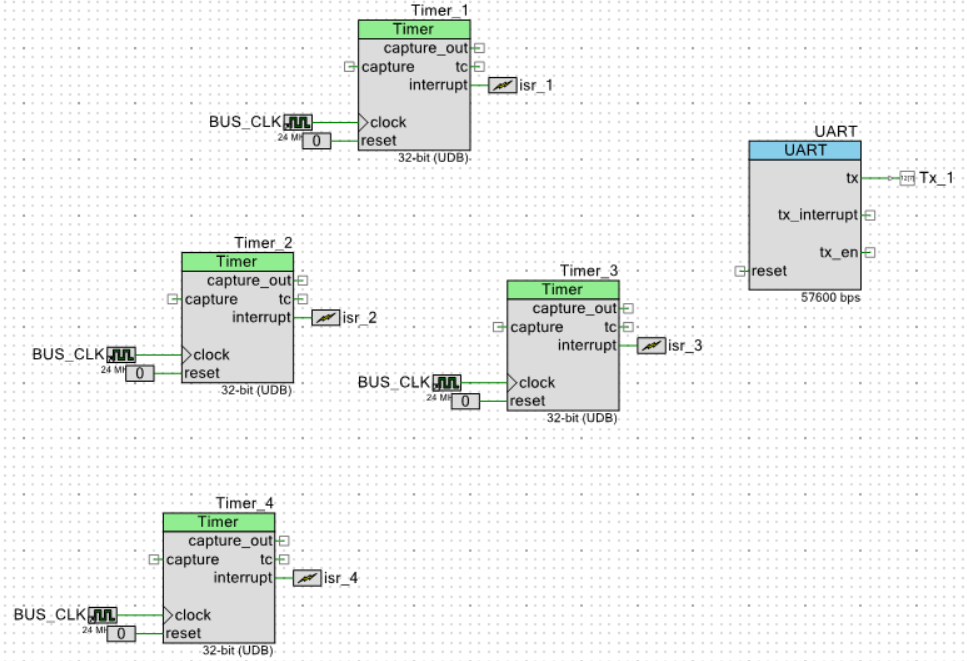


**Date: November 15th, 2019**

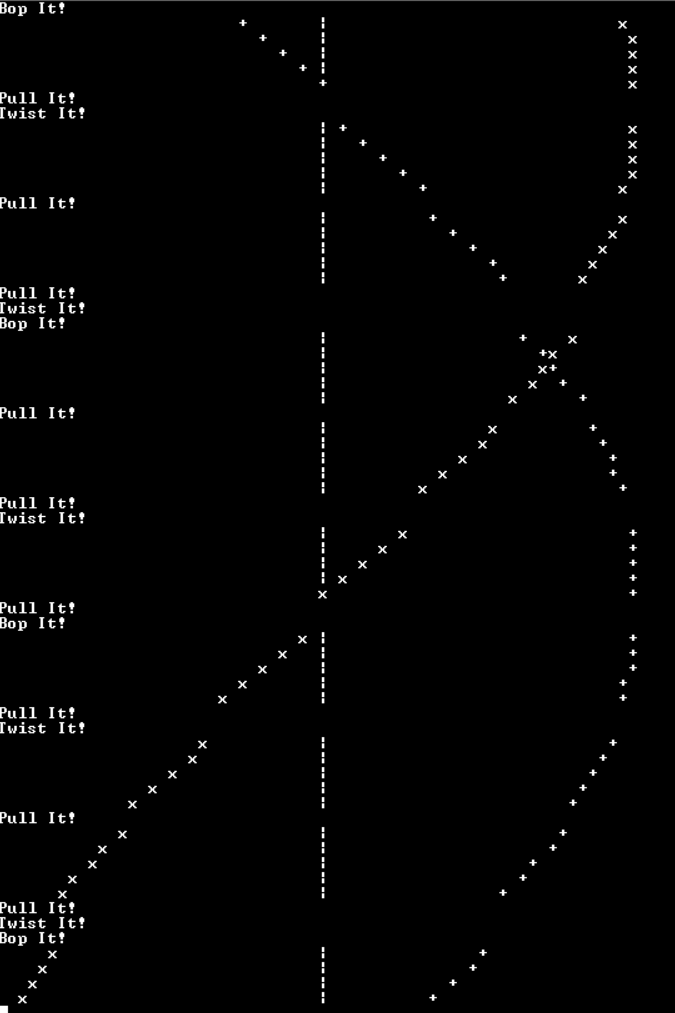
**Submitted by: Jordan Sinoway, Condor Gao**

**Instructor: Dr. Larry Pearlstein**

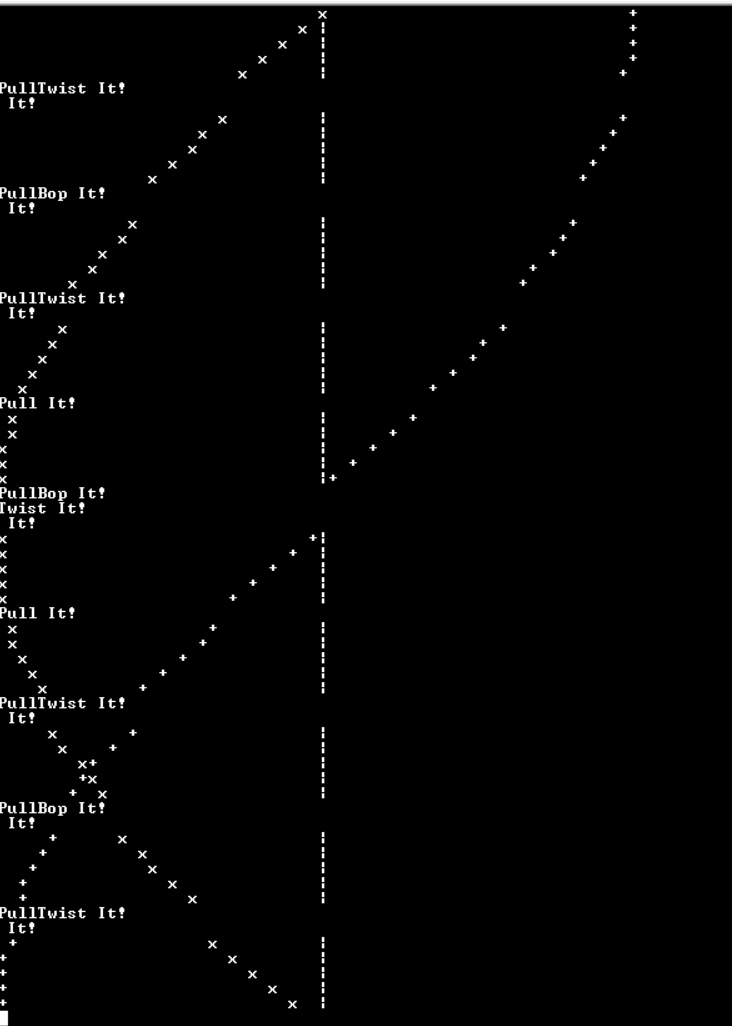
**PSoC Schematic:**



**Tera Term Window Based on Section 3:**

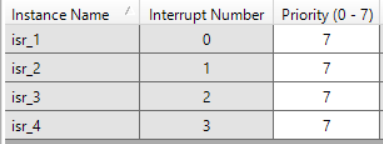


**Tera Term Window Post Section 5:**

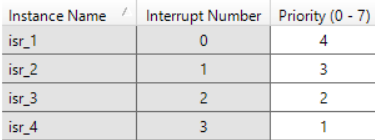


**Explanation:  
What are the default priorities on all four interrupts?**

Default:

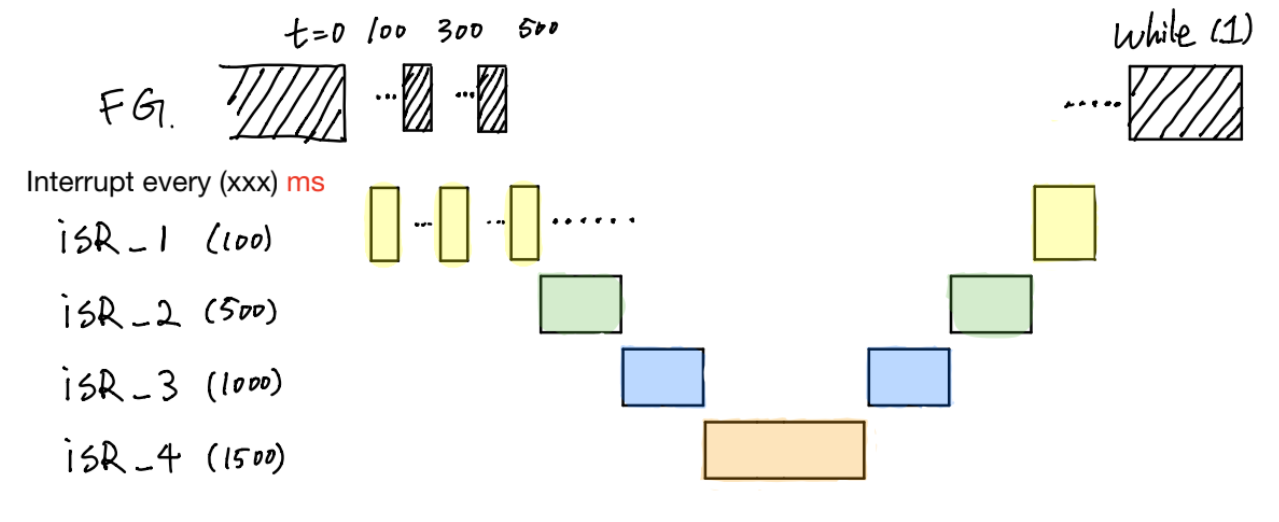


Modified:



**Modify the interrupt priorities to match those shown below. Rerun. What do you see? Can you explain that? Can you draw a timing diagram that helps to explain the situation?**

As shown above in the 3rd figure, now that the interrupts have priorities have been assigned, the interrupts are interrupting each other, as well as the main program. Previously, all of the interrupts were assigned with the same priority, so they operated sequentially, as shown in the 2nd figure. Now, the interrupts can be interrupted by other interrupts, and will finish after the higher priority interrupts have run. For example, the 3rd figure shows “PullTwist It! It!”, which occurs because isr\_3 (“Twist It!”) has a higher priority than isr\_2 (“Pull It!”). A timing diagram, shown below, describes the behavior of the interrupts.



**Code:**

**ISR\_1:**

CY\_ISR(isr\_1\_Interrupt)

{

#ifdef isr\_1\_INTERRUPT\_INTERRUPT\_CALLBACK

isr\_1\_Interrupt\_InterruptCallback();

#endif /\* isr\_1\_INTERRUPT\_INTERRUPT\_CALLBACK \*/

/\* Place your Interrupt code here. \*/

/\* `#START isr\_1\_Interrupt` \*/

char msg\_str[67];

int64\_t rsq; // Holds square of real component

int64\_t isq; // Holds square of imag component

int32\_t sgn\_real; // +/-1, based on x\_real > 0 or < 0

int32\_t sgn\_imag; // +/-1, based on x\_imag > 0 or < 0

x\_real = x\_real\*COS\_0 - x\_imag\*SIN\_0; // QM.N \* QM.N -> Q2M.2N

x\_real = (x\_real + FIXED\_ONE\_HALF) >> FIXED\_N; // convert back to QM.N, with rounding

x\_imag = x\_real\*SIN\_0 +x\_imag\*COS\_0; // QM.N \* QM.N -> Q2M.2N

x\_imag = (x\_imag + FIXED\_ONE\_HALF) >> FIXED\_N; // convert back to QM.N, with rounding

sgn\_real = x\_real >= 0 ? 1 : -1; // signum of x\_real

sgn\_imag = x\_imag >= 0 ? 1 : -1; // signum of x\_imag

rsq = SQR(x\_real); // QM.N \* QM.N -> Q2M.2N

rsq = (rsq + FIXED\_ONE\_HALF) >> FIXED\_N; // convert back to QM.N, with rounding

isq = SQR(x\_imag); // QM.N \* QM.N -> Q2M.2N

isq = (isq + FIXED\_ONE\_HALF) >> FIXED\_N; // convert back to QM.N, with rounding

// The goal is for 'x' to represent a value on the unit circle, i.e. complex magnitue

// should be unity. Because of quantization error (which will even occur with double

// precision, need some way of avoiding collapsing to zero, or growing to infinity

//

// My quick and dirty solution is to push both real and imaginary part down toward

// zero a smidgen if the magnitude is greater than one, and away from zero otherwise

if (rsq+isq > FIXED\_ONE)

{

x\_real -= sgn\_real;

x\_imag -= sgn\_imag;

}

else

{

x\_real += sgn\_real;

x\_imag += sgn\_imag;

}

build\_plot\_str( msg\_str, (int) x\_real, (int) x\_imag ); // Builds the string for output (at 57,600 baud, to serial port

UART\_PutString(msg\_str); // Sends string to serial port

Timer\_1\_ReadStatusRegister();

/\* `#END` \*/

}

**ISR\_2:**

CY\_ISR(isr\_2\_Interrupt)

{

#ifdef isr\_2\_INTERRUPT\_INTERRUPT\_CALLBACK

isr\_2\_Interrupt\_InterruptCallback();

#endif /\* isr\_2\_INTERRUPT\_INTERRUPT\_CALLBACK \*/

/\* Place your Interrupt code here. \*/

/\* `#START isr\_2\_Interrupt` \*/

char msg[32];//message string

sprintf(msg, "Pull It!\r\n"); //printing formatted string

UART\_PutString(msg); //sending formatted string to UART

Timer\_2\_ReadStatusRegister(); //check timer 2

/\* `#END` \*/

}

**ISR\_3:**

CY\_ISR(isr\_3\_Interrupt)

{

#ifdef isr\_3\_INTERRUPT\_INTERRUPT\_CALLBACK

isr\_3\_Interrupt\_InterruptCallback();

#endif /\* isr\_3\_INTERRUPT\_INTERRUPT\_CALLBACK \*/

/\* Place your Interrupt code here. \*/

/\* `#START isr\_3\_Interrupt` \*/

char msg[32];

sprintf(msg, "Twist It!\r\n");//printing formatted string

UART\_PutString(msg); //sending formatted string to UART

Timer\_3\_ReadStatusRegister();//check timer 3

/\* `#END` \*/

}

**ISR\_4:**

CY\_ISR(isr\_4\_Interrupt)

{

#ifdef isr\_4\_INTERRUPT\_INTERRUPT\_CALLBACK

isr\_4\_Interrupt\_InterruptCallback();

#endif /\* isr\_4\_INTERRUPT\_INTERRUPT\_CALLBACK \*/

/\* Place your Interrupt code here. \*/

/\* `#START isr\_4\_Interrupt` \*/

char msg[32];

sprintf(msg, "Bop It!\r\n");//printing formatted string

UART\_PutString(msg); //sending formatted string to UART

Timer\_4\_ReadStatusRegister();//check timer 4

/\* `#END` \*/

}