Experiment 5
Building a Stopwatch

Course: EE 425
Section: 1 XB
Semester: Summer 2016
Instructor's name: Alfredo Cano
Students: Aviv Arusi and William-Jack Dalessandro
Group 7

Table of Contents:

Objective:

The objective of this assignment is to write the code for and build a stopwatch that counts milliseconds, seconds, and minutes utilizing the screen on the Microcontroller. The stopwatch needs to be able to pause for however long we choose and reset itself both when the time reaches an hour and when engaging the interrupt.

Overview:

In order to build this stopwatch we will need to program a few things. Firstly, we need to make the clock loop. In the program each digit of the timer is nested within the loop of the digit to the right (the smaller one). For example the seconds that are counted will be nested in the loop of the hundred milliseconds which are nested in the ten millisecond loop. And that continues up until the tens of minutes which is the leftmost digit.

Secondly, we will need to program the screen to display the digits with colons in between every two digits, the word STPWATCH when in the stopwatch function, and RESET when the stopwatch is reset.

Finally, we will need to program in the interrupts. These interrupts will be used to pause the stopwatch, only to continue when we push a certain button, controlled by the low interrupt, and to reset the the stopwatch which will be controlled by the high interrupt.

Procedure/Analysis:

Part 1

In the first part of this experiment, we had to come up with the assembly logic of how our stopwatch would run. After some original thoughts that were written on paper and testing the code as it went from pseudo to working, everything eventually came together. The idea behind how the stopwatch would run was to increment the display on the screen every 10ms and have the digits count up to their respective maximum. This simply means that every 1000 milliseconds would be 1 second and every 60 seconds would equal 1 minute, as common knowledge would dictate. To accomplish this in the mainline routine, a series of loops made from simple conditions and branches were set up that would check and increment or decrement certain variables. The initialization of each variable used in the mainline routine is shown below.

```
Initialization of the variables used in the mainline subroutine

            MOVLF 10, numcount1
            MOVLF 10, numcount2
            MOVLF 10, numcount3
            MOVLF 6, numcount4
            MOVLF 10, numcount5
            MOVLF 10, numcount6

            MOVLF 0x30, value1
            MOVLF 0x30, value2
            MOVLF 0x30, value3
            MOVLF 0x30, value4
            MOVLF 0x30, value5
            MOVLF 0x30, value6
```

Value1 has to do with the number that will be outputted on the LCD screen for each digit shown. This is initialized with the address that will show the ASCIII value of “0”. Numcount is used as a counter for each digit to give the conditions for how much each digit will increment in their nested loops in the mainline routine. In other words, this will determine when a specific digit on the LCD screen will cycle from 0 to 9 or from 0 to 5. Numcount is initialized with values that will be reinstated each time a nested loop cycles from its max value to its lowest one. A logic diagram of mainline routine is shown in figure 1. In order to save space and not make the flow diagram very convoluted, the loops for digits 3-5 are not shown.

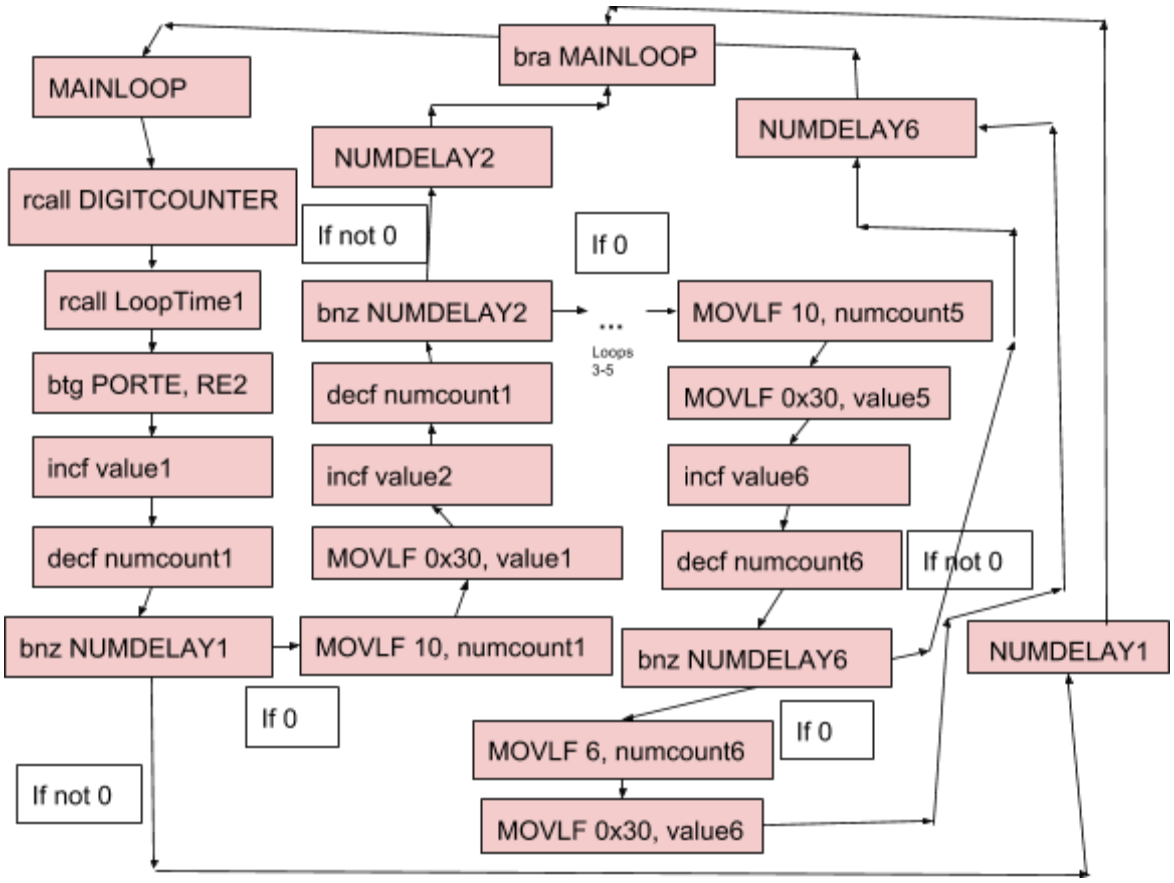

Figure 1: Mainline subroutine flow diagram

Each time a loop is iterated in the mainline, the value variables are increased while numcount is decreased. This continues until numcount equals 0, and then value is again set to its ASCII address of “0” and numcount is set to 10 or 6, depending on the digit it’s representing. While running our program, to ensure that this mainline loop was in fact taking 10 ms between each iteration, the btg PORTE, RE2 command was included. This is so that we could measure the square wave output that came from that port to indeed confirm that it toggled every 10ms. A picture of this square wave is shown in Figure 2. From Figure 2, one can see that it is toggled every 10ms.

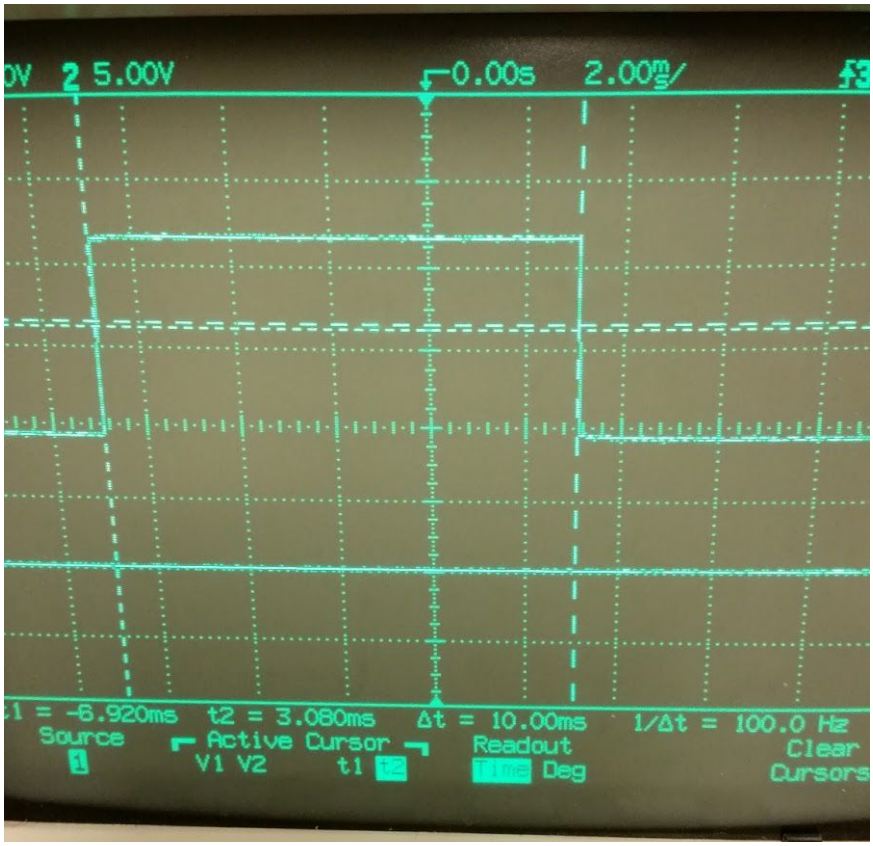

Figure 2: 10ms Square Wave Output

Part 2

In the second part of this experiment, we had to set the specific parts of the screen to display what we want. In our code the loop occurs every ten milliseconds; every time this loop occurs the screen redisplays all of the information and shows the changes that happened in the previous loop. This part of the code is programmed in the DIGITCOUNTER subroutine which is the first line in the MAINLOOP. Our display is broken up into two rows. The first row will either display STPWATCH or RESET. The second row however, displays the actual clock counter 00:00:00 counting minutes, seconds, and ten milliseconds. The second row is all controlled by the variable DIGIT and its 9 corresponding variables, DIGIT+1, DIGIT+2, …, DIGIT+9. DIGIT controls the location (0xc0) and the last one, DIGIT+9, is the null (0x00) which marks the end. This leaves 8 digits in between which each represent a space on the second row of the display. Some of these are constant (like the colons) while the other are constantly changing. The initialization of DIGIT and the DIGITCOUNTER subroutine is displayed below.

```
*Initialization of DIGIT and its corresponding variables*

           DIGIT:10

DIGITCOUNTER

           POINT  STOPWATCH                  ;Display "STPWATCH"
           rcall  DisplayC

           MOVLF 0xc0, DIGIT
           MOVLF 0x3a, DIGIT+3
           MOVLF 0x3a, DIGIT+6
           MOVLF 0x00, DIGIT+9

           movff value1, DIGIT+8
           movff value2, DIGIT+7
           movff value3, DIGIT+5
           movff value4, DIGIT+4
           movff value5, DIGIT+2
           movff value6, DIGIT+1

           lfsr  0,DIGIT
           rcall  DisplayV

      return
```

It can be seen from the code above that the location, null and two colons are constant while the other locations (DIGIT+8 for example) are given values which are subject to change throughout the loops. In the main code it can be seen that the variables value1-value6 change depending on which digit they represent. Their values increment by one until either 9 or 5 depending on their location and the number they represent. The flow diagram for the DIGITCOUNTER subroutine is shown in Figure 3.

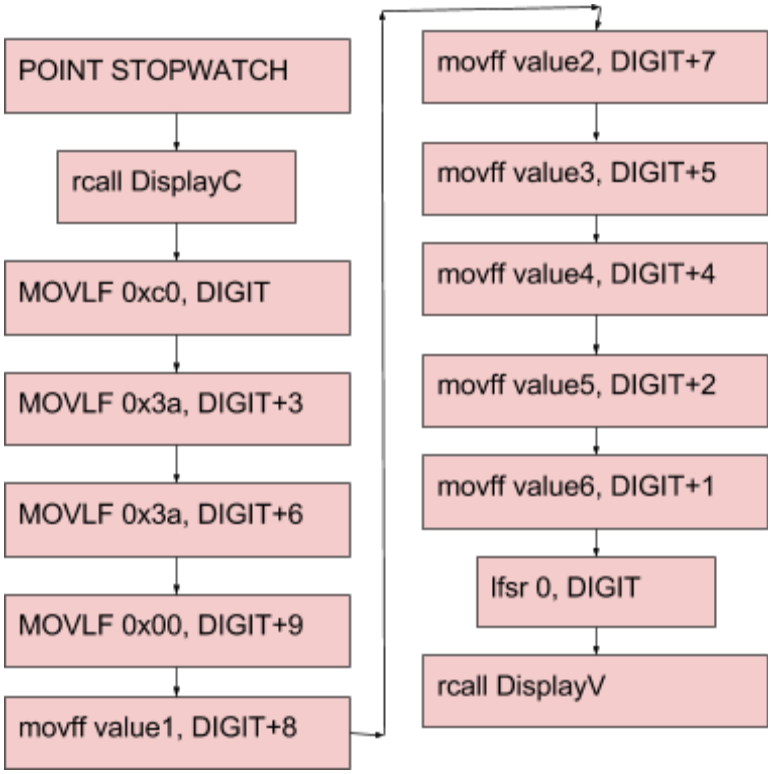

Figure 3: DIGITCOUNTER Flow Diagram

Part 3

In the third and final part of this experiment, we had to set the interrupts in order to control the stopwatch once the program has started running. As stated earlier the low priority interrupt pauses the stopwatch while the high priority interrupt resets the stopwatch.

The high priority interrupt is triggered by PORTB, B2 and it is set to the rising edge which means that when the input at B2 changes from ground to high, the high priority interrupt is triggered. The high and low priority initialization for the ports is done through bits 7 and 6 in INTCON3. The rising edge selection is controlled from INTCON2. The initialization for each of the INTCON registers is shown below.

```
MOVLF  B'11010000',INTCON
MOVLF  B'11110000',INTCON2
MOVLF  B'10011000',INTCON3
```

In order to pause the stopwatch the low priority interrupt subroutine starts a loop. The loop is set to continue looping until PORTD, RD3 is set from high to low. This specific port is a button on the Qwikflash board. The port is always set to high but when the button is pressed it grounds the input to the port, takes the program out of the loop, out of the LPI, and back to the MAINLOOP which continues the stopwatch from where it left off. The code for the LPI subroutine is displayed below.

```
LPI
            movff WREG, WREGSAVE
            movff STATUS, STATUSSAVE

            LPIloop
                  btfsc PORTD, RD3
                  bra LPIloop

            movff WREGSAVE, WREG
            movff STATUSSAVE, STATUS
            bcf INTCON3,INT1IF ;clear flag
RETFIE
```

The flow diagram for the low priority interrupt is shown in Figure 4.

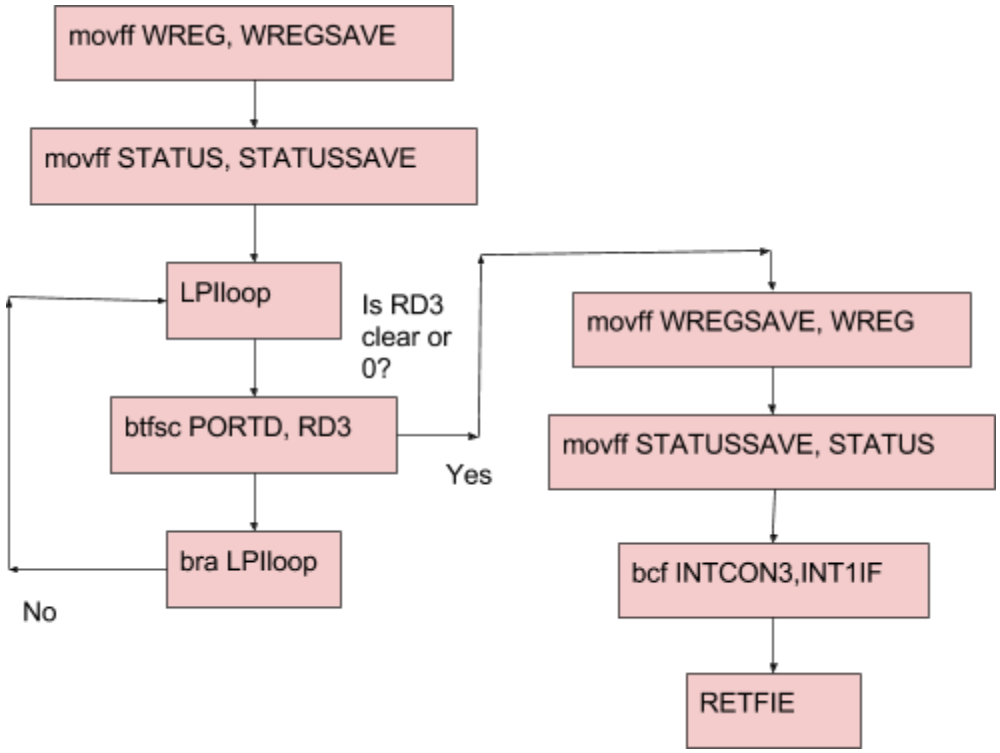

Figure 4: LPI Flow Diagram

The low priority action can be seen in Figure 5. The button used to break out of the low priority interrupt, SW3, is also shown in Figure 5.

Figure 5: Paused Stopwatch

The high priority interrupt subroutine is triggered by PORTB B2. Once in the high priority interrupt subroutine, the code is fairly simple. We reset the screen by displaying the screenclear string, which is just a string that displays whitespace. Then we display RESET on the first row, wait two seconds and then reset all of the values in the loops including the variables that keep track of what numbers are displayed, as well as the values contained in the display.

```
HPI
            POINT screenclear1     ;clear screen
            rcall DisplayC
            POINT screenclear2
            rcall DisplayC

            POINT RESET1
            rcall DisplayC
```

```
        rcall OneSecDelay      ;wait 2 seconds
        rcall OneSecDelay

        MOVLF 10, numcount1
        MOVLF 10, numcount2
        MOVLF 10, numcount3
        MOVLF 6, numcount4
        MOVLF 10, numcount5
        MOVLF 10, numcount6

        MOVLF 0x30, value1
        MOVLF 0x30, value2
        MOVLF 0x30, value3
        MOVLF 0x30, value4
        MOVLF 0x30, value5
        MOVLF 0x30, value6

bcf INTCON3,INT2IF ;clear flag
retfie FAST
```

The high priority interrupt action is shown in Figure 6.

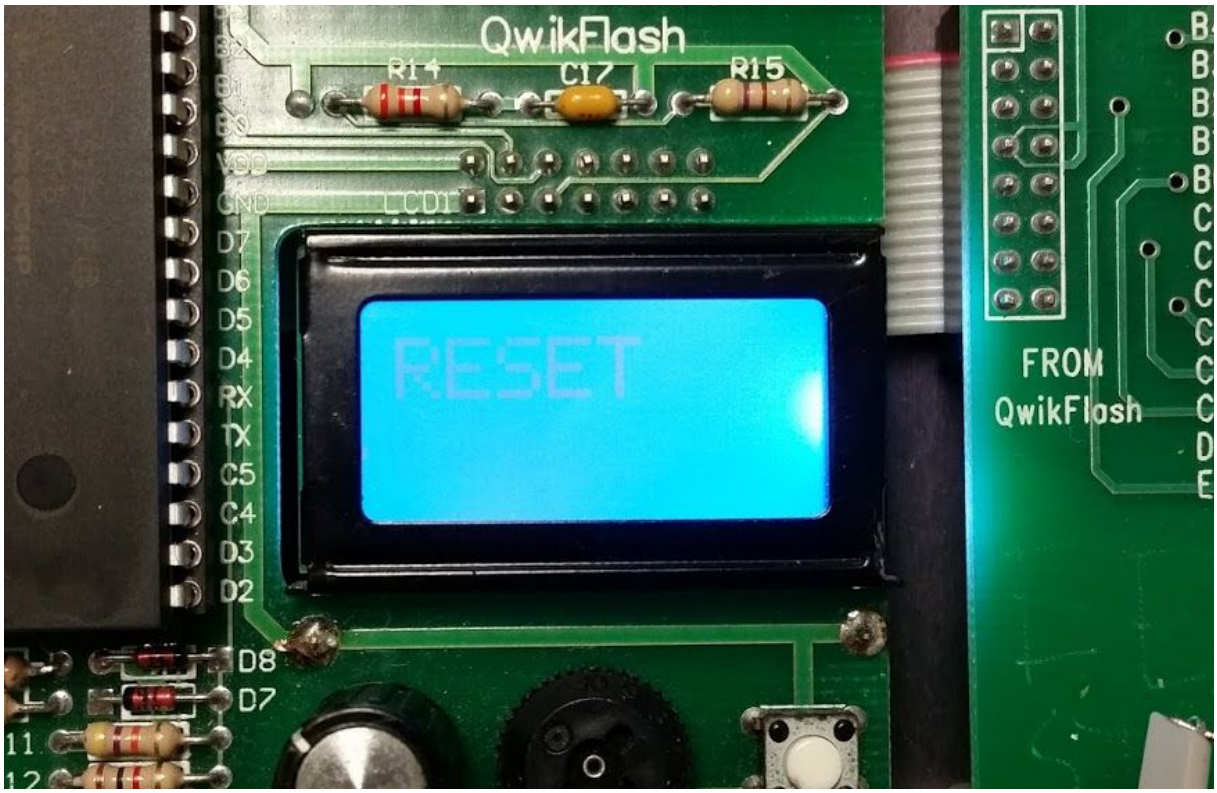

Figure 6: High Priority Interrupt RESET Screen

Conclusion:

We successfully built a stopwatch that had the capabilities we were hoping to create. We ran into issues when learning how to correctly display the digits in an organized way and had to undergo quite a bit of trial and error in order to find a sufficient design. We also experienced a setback with the mainline code,as we originally had each value change based off of its own counter, however, after some thought we decided to change that and have one single loop with the other loops nested within it to ensure that all of the individual loops were synchronized. Overall we learned a lot about controlling loops in assembly and manipulating code to display what we wanted on the LCD screen.