

**Sports Watch**

**Department of Computer Science**

**Introduction to Microprocessor System – CDA 3331C**

**Richard Santiago**

**Monday, April 22, 2019**

Table of Contents

[Abstract 3](#_Toc6910013)

[Design Methodology or Theory 3](#_Toc6910014)

[Implementation 4](#_Toc6910015)

[Conclusion 13](#_Toc6910016)

[Acknowledgements 13](#_Toc6910017)

[References 14](#_Toc6910018)

[Appendix Page 15](#_Toc6910019)

# Abstract

The objective of this project is to design and implement a sports watch on a 7-segment display with three modes: clock, temperature, and stopwatch. These modes will then be capable of minor configurations by the end user.

# Design Methodology or Theory

I began by looking into the temperature sensor I would utilize**.** Initially I analyzed project #13 from Lucio Di Jasio’s book in 10 lines of Code. This project uses the Master Synchronous Serial Port module (hereafter ‘MSSP’), a master device able to communicate with other slave devices, such as a temperature sensor, through the Inter-Integrated Circuit bus (I2C). As I begun setting up the MPLab Code Configurator (MCC) for the MSSP, I realized issues would arise, as the Serial Clock and Serial Data (‘SCL’ and ‘SDA’, respectively) of this master-slave interface were hardwired to function through pins RC4 and RC3. Since the 7-segment LED display is easily plugged into the header of the circuit board I was using, and two of the pins were RC4 and RC3, this would leave these pins unavailable for use.

Shortly after reading about the included timers on my circuit board to determine which would be best for the ‘stopwatch’ and the ‘clock modes’, I found a lot of great information on <microchipdeveloper.com>, such as how ‘TIMER O’ (*tmr0*) module can execute ‘counter’ and ‘timer modes’, although they are not proper modes which can be selected but must be implemented. Initially, I thought I would use the ‘counter mode’ for both the clock and stopwatch, but I found that *tmr0* required an external clock to be set up on one of the pins for this type of functionality.

For ‘timer mode’, the *tmr0* register valuewas then incremented on every instruction clock pulse, which was appropriate for this design because selecting a system clock and pre-scaler to adjust the given period would allow the calculation of pulses corresponding to some time (a second, a minute, *etc*). But the *tmr0* already had better functionality than this process because the requested period set in the MPLab Code Configurator (MCC) was the time, at overflow, of the *tmr0* register, as well as also including anoverflow function. The MCC is a graphical user interface which allows a user to select modules and different options (such as I/O ports, periods, *etc*) on the basis of individual functionality.

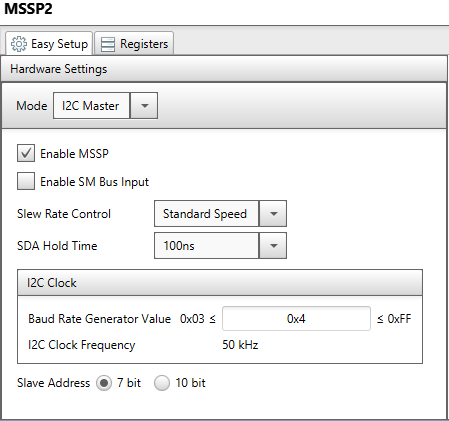
As I moved onto the timer module which I would later use for the ‘clock mode’, I was uncertain about whether to useTIMERs 2, 4, 6(identical functionality)orTIMER1**.** I was initially deterred from TIMER1 because it was a 16-bit timer that might be simpler but more taxing on the system. Ultimately, I chose it for simplicity because it lacked other control modes which would not be needed. Using TIMER1, I realized I could achieve an exact period of one second. This resulted from the use of 16-bit precision, FOSC/4 (frequency of oscillator) as the clock source, and a pre-scaler ratio of 1:4, resulting in even divisions with no remainders. This is, of course, ideal for keeping time through a 24-hour period.

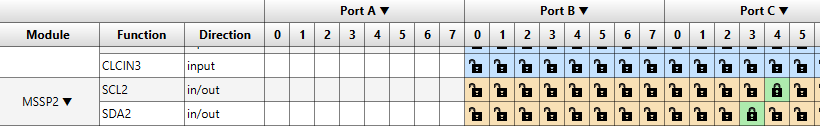
To switch between the modes of the sports watch, the potentiometer was used: a three-terminal resistor able to be adjusted to divide the voltage. Through the Analog-to-Digital Converter with Computation (‘ADCC’), the input values are converted to discrete values and are later used as reference for a given mode.

To configure the push button to change modes accurately, it was first necessary to debounce the input signal from the button. This was done using a timer and a Configurable Logic Cell (‘CLC’) module. The timer was then configured to ‘monostable mode’, which is a high limit timer, such that if the button was pressed the timer would begin and go for the predetermined time (period). If the signal (of the button) was still high at the end, then the timer produces a signal to indicate activation. The CLC was used as a ‘JK flip-flop’ with high values to both inputs such that every button press will toggle the value output by the CLC (switching modes with every press).

# Implementation

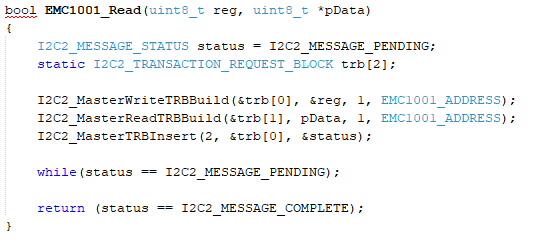
For my design, an 8-bit microcontroller was used (exact model: PIC16F18855). In the design for the temperature mode, I first began by configuring the MCC, enabling the MSSP module in I2C master mode with slew rate control at standard speed (change in output voltage over time), and baud rate generator value to 0x4, resulting in a 50kHz clock frequency for the I2C (*below*).

 (Screenshot of *MSSP MCC setup*)



(Screenshot of *MSSP configuration in Pin Manger*)

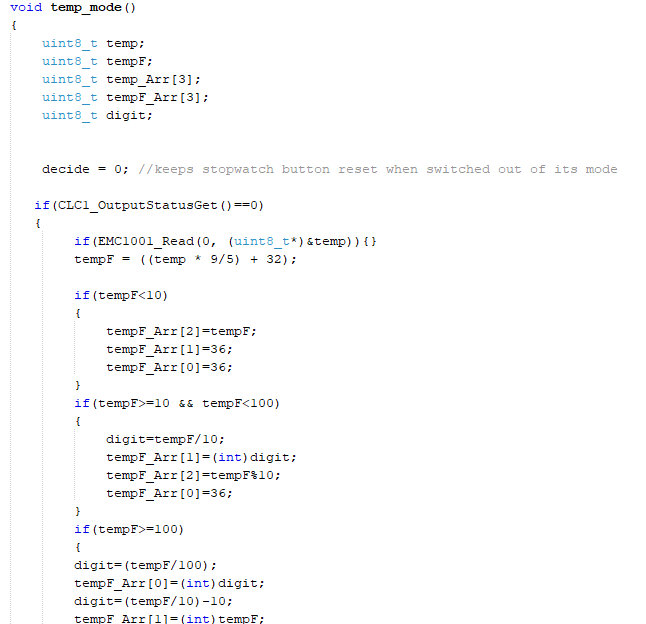
The SCL and SDA pins of the MSSP were then locked to RC3 and RC4 as seen above. The following screenshot (*below*) displays the code from Project #13 used to read temperature values from the temperature sensor.

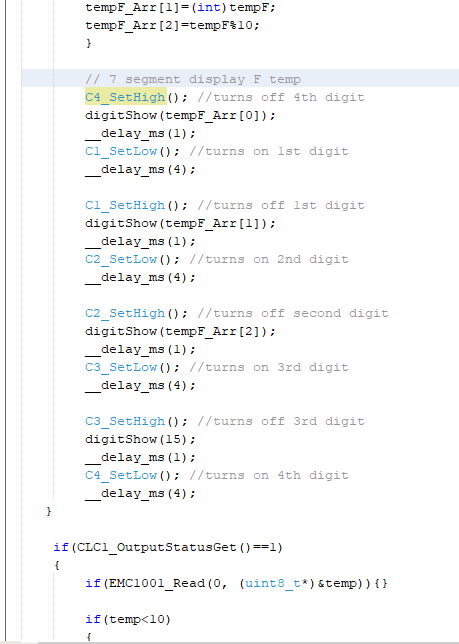


(Screenshot of *temperature read function(above)*)

Here we see the communication from a master device through the I2C to retrieve the current temperature value (in degrees Celsius) from the temperature sensor. Reading a register value is a two-step process that first requires selecting the register by a write function and then reading the value with a read function. The great thing about the I2C is it is based-on interrupts and performs these functions automatically. ‘TRB’ here stands for Transaction Request Block and the TRBInsert function passes a status variable, constantly checking if a value has been retrieved (all done asynchronously).

The two pictures below display the C code implementation for outputting the temperature to the 7-segment display. The first IF statement receives a value from the CLC which is tied to the button input to determine ‘Celsius’ or ‘Fahrenheit mode’. The next IF statement retrieves the temperature in Celsius from the EMC1001 (onboard temperature sensor) and then calculates the correct Fahrenheit temperature from that value. The three following IF statements separate the digits of the integer value such that they may be placed individually into array indexes and then put into the digitShow function to display on the 7-segment LEDs (further discussion on 7-segment LEDs and digitShow to follow).

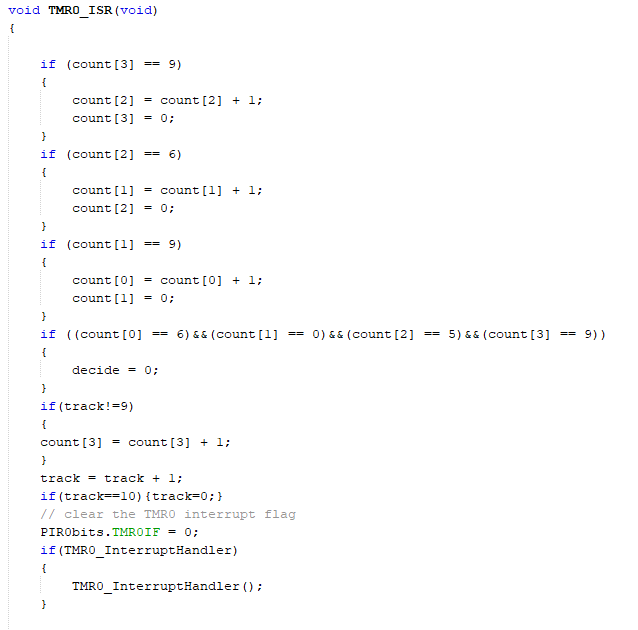
(Screenshot of ‘*temperature mode’ function*)

 (Screenshot 2 of ‘*temperature mode’ function*)

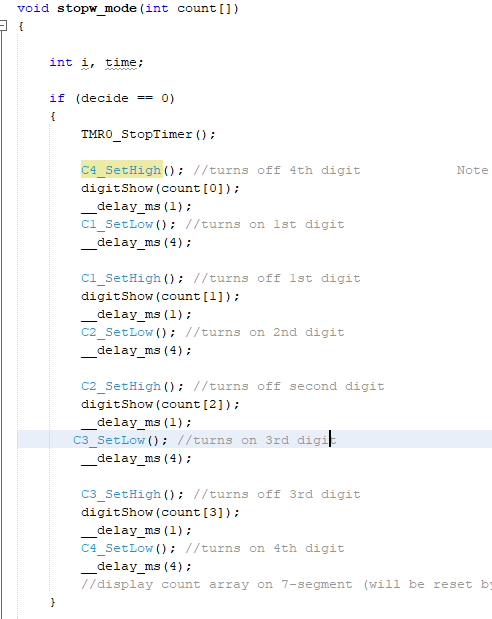
The ‘stopwatch mode’ for me was the trickiest and took the longest time. I began by setting the *tmr0* to a 1 second period (actual period: 999.424ms). Following that, I recalled there being a *tmr0* overflow function and so immediately began implementing it into my main function.

I set this up with a FOR loop, using an *i*-index, set to increment an array with 4 integers (0,0,0,0) to a very large number. This then had IF statements set to increment the first position of the array upon timer overflow, then increment the second position if the first was == 9 and so on. But I soon realized that the IF overflow loop resulted in a 1000ms delay before turning on the LEDs each time. This resulted in the LEDs not being perceived to be continuously lit by the human eye. I was also not getting any output to the 7-segment LED at first and went on to realize I had to start and stop the timer before and after every overflow.

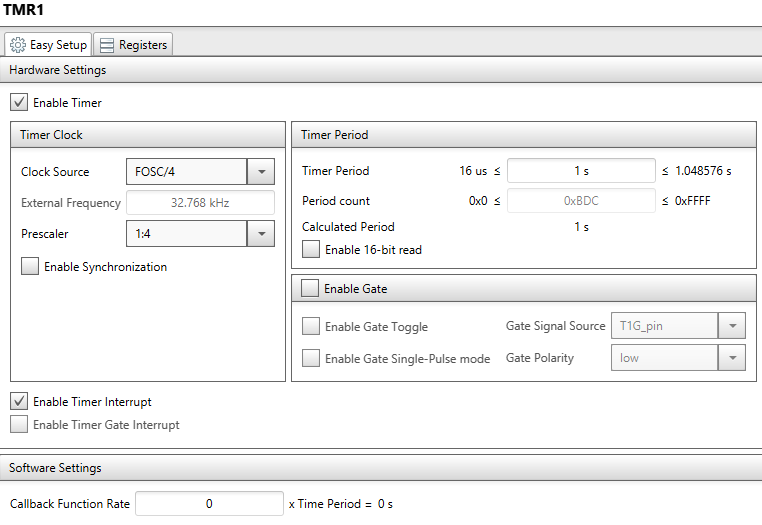
Thus, I went on to enable a *tmr0* interrupt making the function much simpler. I was able to move the count array into the *tmr0* ISR (or, ‘Interrupt Service Routine’) along with the IF statements, in order to increment each corresponding index. This interrupt is called automatically after each overflow, so no ‘FOR loop’ nor any additional coding were needed; this only requires care be taken to clear the *tmr0* flag bit after each interrupt or the program would get stuck.

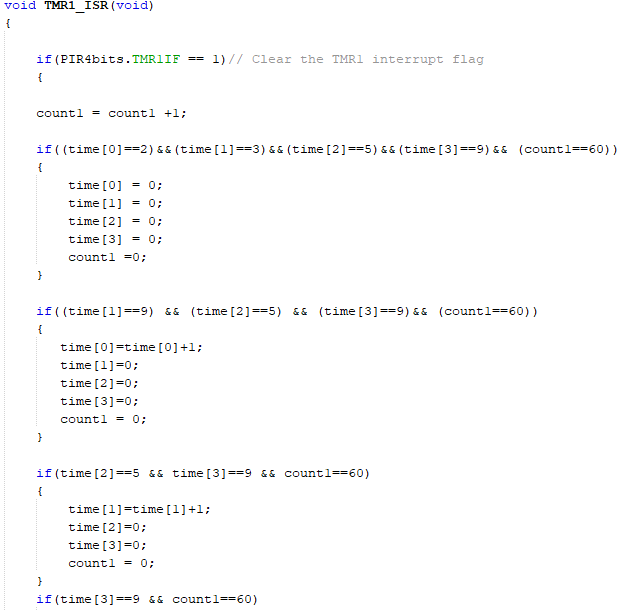
(Screenshot of *tmr0* ISR)

The picture below shows the user-defined ‘stopwatch function’, beginning with an IF statement to verify the variable *decide* is at ‘0’, where it should be when first initiating the loop. This is made sure to be TRUE by resetting the value of *decide* to ‘0’ in each of the other modes. The IF loop for *decide* == 0 initially shows four zeros on the display, and then once the button is pressed, the timer begins. If pressed again, the timer will stop by calling the *TMR0*\_StopTimer(); function and the current value will be constantly displayed.

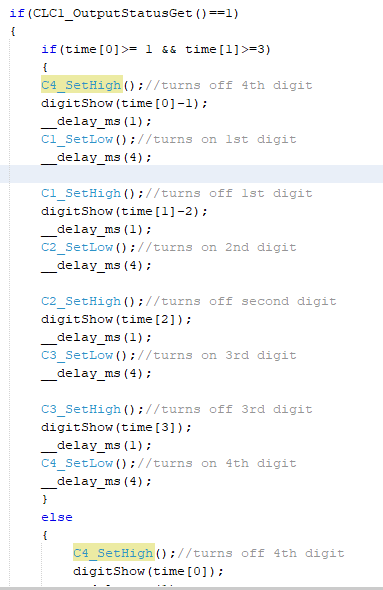
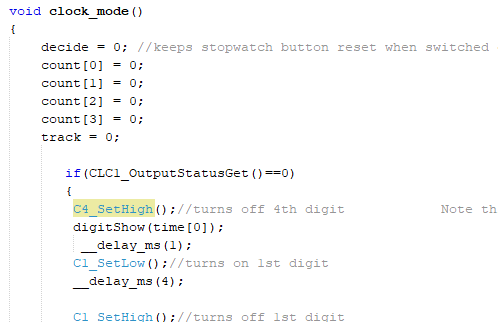
 (Screenshot of *‘stopwatch mode’*)

The ‘clock mode’ was done similarly to the ‘stopwatch mode’. I began by setting up TIMER 1 with a period of exactly 1 second (through the aforementioned 16-bit and /4/4) and by enabling the timer interrupt. Then, I went into the *tmr1.c* file and added my interrupt code into the TMR1\_ISR function so that my variable would increment every 1s and once it reached 60s, it would then increment my array for time accordingly, through a series of IF statements. This clock would run asynchronously and would keep incrementing the array even when inside other modes.

(MCC setup of TIMER 1)

(Screenshot of TIMER 1’s ISR)

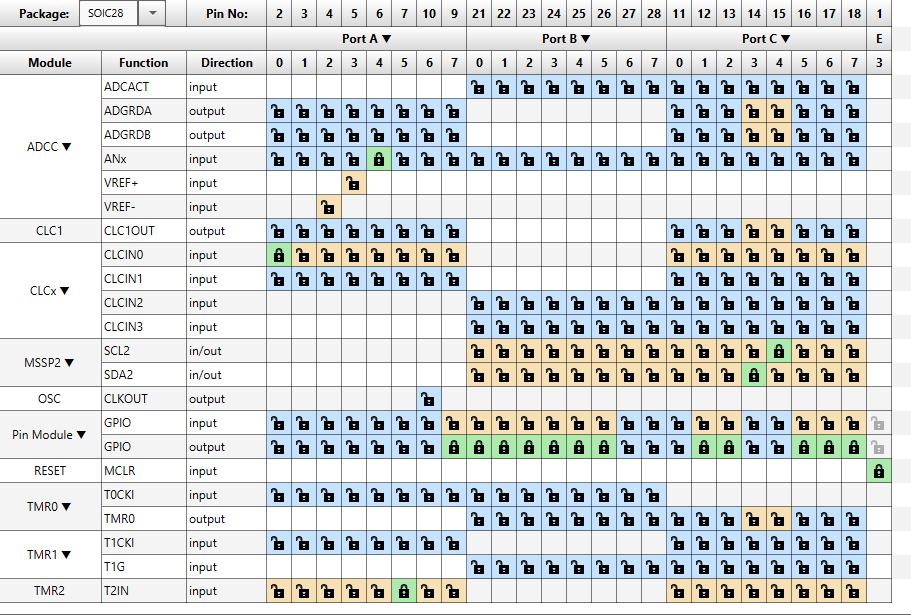
In my main function, the clock mode was a user-defined function which simply displays the time array in 24-hour format to the 7-segment LED display, or in 12-hour format, if the button has been pressed to switch modes. Below, two screenshots of my implementation are displayed.

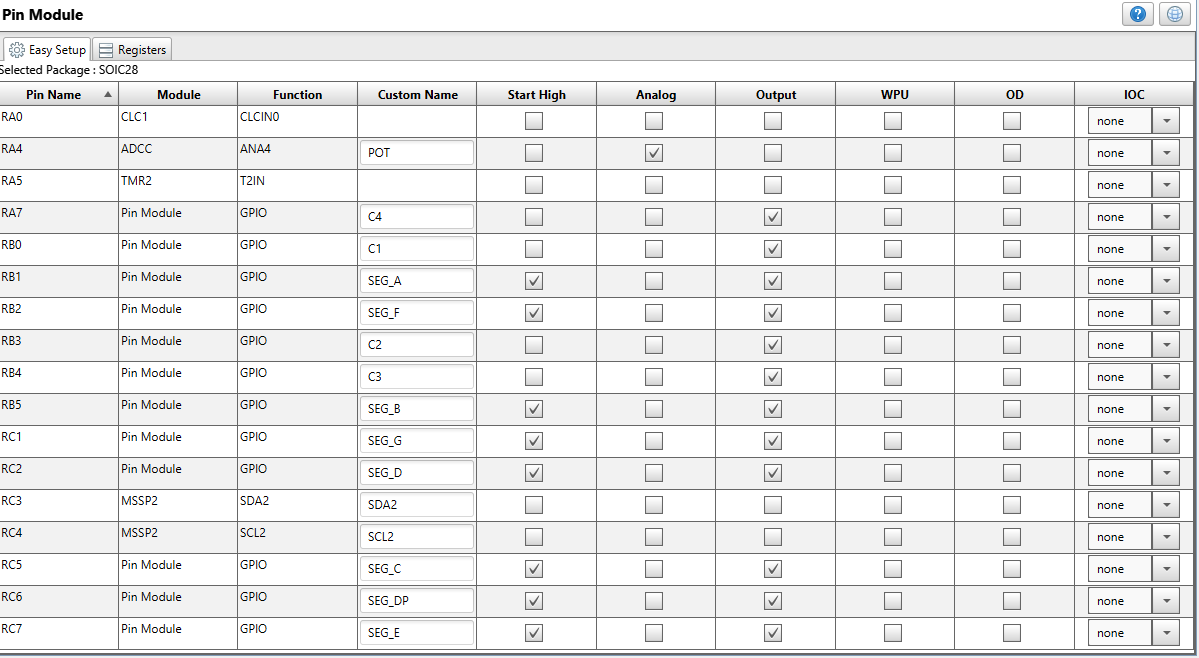
 (Screenshots of ‘clock mode’ function)

The 7-segment LED display (3641AS) was attached to a printed circuit board, designed by FAU's lab manager, Perry Weinthal, through which the device was able to connect its pins to the microcontrollers header (also attached by Mr. Perry Weinthal). As stated earlier however, this was problematic because the MSSP module required the need of those two pins.

As a work-around, I shifted the 7-segment display down and connected the bottom six pins RC1, RC2, RC5, RC6, RC7 as well as RA7 using male-female jumper wires. These pins each correspond to either a different segment of the display (SEG\_A, SEG\_F, *etc*), or to the electrical current of one of four individual 7-segment LEDs (C1, C2, *etc*).

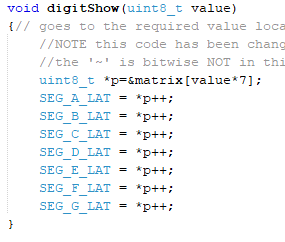
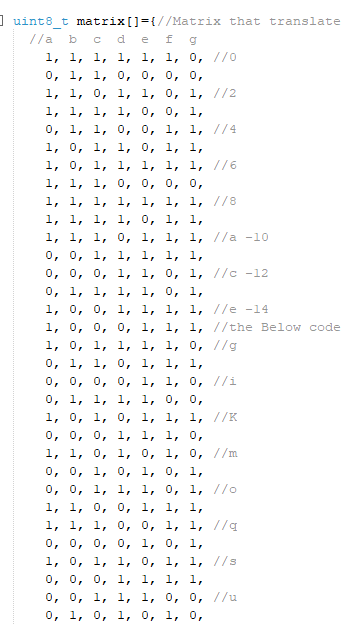
In the below screenshot, pins assigned to their respective module are visible. In the subsequent screenshot, they are assigned names as appropriate depending on their intended function. Further customization can be chosen such as analog/digital, input/outputs.

(Screenshot of *Pin Manager*)



(Screenshot of *Pin Module*)

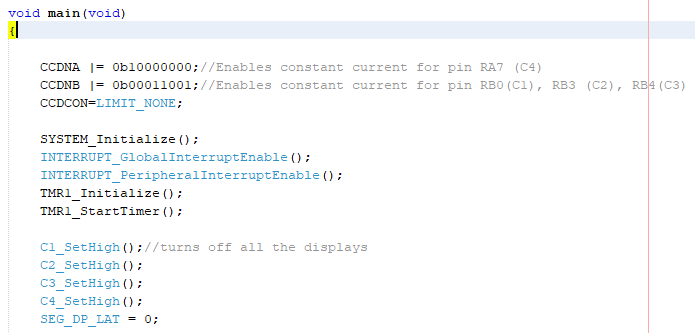
A matrix (integer multidimensional array) with strings of ones and zeros is displayed, wherein each row represents a different number (0-9) or a letter of the alphabet (*below right*). Through this matrix we then can use the function digit show to pass it an integer value so that we can easily select the corresponding segments of the 7-segment LEDs. The matrix and digitShow function were both designed by Douglas Athenosy.



(Screenshot of function *digitShow* (left))

(Screenshot of matrix (right))

The main function begins by enabling constant current for pins RA7, RB0, RB3, and RB4 which correspond to C1, C2, C3, and C4 of the 7-segment LED display. Following this, I applied ‘no limit’ to the CCDCON (‘Current-controlled Drive Mode Control’) register to keep the LEDs at their brightest. Then, initializing functions are called and ‘global/peripheral’ interrupts are enabled.

(Screenshot of *main function*)

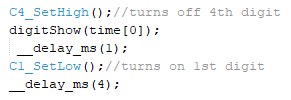
The *while(1)* loop is the simplest part of the program and is the function constantly running and checking the IF loops while the device is powered. These IF loops are checking to see the current value of the potentiometer. This function is what enables the user to effect changes by turning the knob to produce different values (modes). The ADCC returns a number to the accuracy of 5 bits, giving a range of 2^5 (=32). Here, I have divided the first third of the values for the ‘clock mode’, the second third for the ‘temperature mode’, and the last third for the ‘stopwatch mode’.



(Screenshot of *while(1)* loop in main function)

Returning to the topic of 7-segment LEDs and the function digitShow, the screenshot below provides a sample of lines of code where function C4\_SetHigh turns the electrical current off from the panel displaying the 4th digit of the display. The function digitShow(time[0]) is passing the integer value of *time*[0] to its function, where the integer corresponds to a row in the matrix, as displayed above.

The corresponding row then enables all the segments (on the display) that are needed to show that digit or letter (represented by the 1s and 0s). In the intervening period between turning off the last lit segment, and turning on the current for the next (C1\_SetLow();), segments are chosen, and a one millisecond delay is added to remove the LEDs from appearing to bleed into one another to the human eye. Additionally, a miniscule delay of four milliseconds is added afterwards so that the LEDs appear as if they were constantly on.

 (Screenshot of *7-segment code*)

Conclusion

The final compilation of my program gave quite the sense of accomplishment. I was able to implement a sports watch that had three significant modes. ‘Clock mode’ being the first third of potentiometer values, which had functionality to switch between a 12-hour clock and a 24-hour clock. ‘Temperature mode’ came next, initially displaying the temperature in degrees Fahrenheit and with a button press showed the degrees Celsius. The last third ‘Stopwatch mode’ was a timer that would start with a button press and stop with a subsequent press. The mode would keep its value until switched out of, which it then was cleared to all zeros.

The overall functionality of the design was flawless and worked to all required specifications. There was though a bit of flicker to the LEDs and I believe this was due to the numerous lines of code.

The design took me several days and hours of work and headache. From compiling, and recompiling hundreds of times to deleting sections of code or starting over completely. At times I became quite flustered but, in the end, I was able to really enjoy the whole process and even have fun throughout. I am grateful for the experience and knowledge that I was able to gain from the beginning to end and I cannot wait for my next design.

# Acknowledgements

I would like to express great thanks to my professor Ravi Shankar for giving me a great basis of understand on microprocessors, and Doug Athenosy for supplementing that as TA.

# References

[1] L. Di Jasio, *In 10 Lines of Code,* Lulu Enterprises, Inc., 2016, p.118-124

# Appendix Page

Listing 1: Main.c

//Sports Watch

//Richard Santiago

#include "mcc\_generated\_files/mcc.h"

#define EMC1001\_ADDRESS 0x38

#define LIMIT\_NONE 0X00

#define LIMIT\_10mA 0X80

#define LIMIT\_5mA 0X81

#define LIMIT\_2mA 0X82

#define LIMIT\_1mA 0X83

uint8\_t matrix**[]={**

//a b c d e f g

1**,** 1**,** 1**,** 1**,** 1**,** 1**,** 0**,** //0

0**,** 1**,** 1**,** 0**,** 0**,** 0**,** 0**,**

1**,** 1**,** 0**,** 1**,** 1**,** 0**,** 1**,** //2

1**,** 1**,** 1**,** 1**,** 0**,** 0**,** 1**,**

0**,** 1**,** 1**,** 0**,** 0**,** 1**,** 1**,** //4

1**,** 0**,** 1**,** 1**,** 0**,** 1**,** 1**,**

1**,** 0**,** 1**,** 1**,** 1**,** 1**,** 1**,** //6

1**,** 1**,** 1**,** 0**,** 0**,** 0**,** 0**,**

1**,** 1**,** 1**,** 1**,** 1**,** 1**,** 1**,** //8

1**,** 1**,** 1**,** 1**,** 0**,** 1**,** 1**,**

1**,** 1**,** 1**,** 0**,** 1**,** 1**,** 1**,** //a -10

0**,** 0**,** 1**,** 1**,** 1**,** 1**,** 1**,**

0**,** 0**,** 0**,** 1**,** 1**,** 0**,** 1**,** //c -12

0**,** 1**,** 1**,** 1**,** 1**,** 0**,** 1**,**

1**,** 0**,** 0**,** 1**,** 1**,** 1**,** 1**,** //e -14

1**,** 0**,** 0**,** 0**,** 1**,** 1**,** 1**,**

1**,** 0**,** 1**,** 1**,** 1**,** 1**,** 0**,** //g

0**,** 1**,** 1**,** 0**,** 1**,** 1**,** 1**,**

0**,** 0**,** 0**,** 0**,** 1**,** 1**,** 0**,** //i

0**,** 1**,** 1**,** 1**,** 1**,** 0**,** 0**,**

1**,** 0**,** 1**,** 0**,** 1**,** 1**,** 1**,** //K

0**,** 0**,** 0**,** 1**,** 1**,** 1**,** 0**,**

1**,** 1**,** 0**,** 1**,** 0**,** 1**,** 0**,** //m

0**,** 0**,** 1**,** 0**,** 1**,** 0**,** 1**,**

0**,** 0**,** 1**,** 1**,** 1**,** 0**,** 1**,** //o

1**,** 1**,** 0**,** 0**,** 1**,** 1**,** 1**,**

1**,** 1**,** 1**,** 0**,** 0**,** 1**,** 1**,** //q

0**,** 0**,** 0**,** 0**,** 1**,** 0**,** 1**,**

1**,** 0**,** 1**,** 1**,** 0**,** 1**,** 1**,** //s

0**,** 0**,** 0**,** 1**,** 1**,** 1**,** 1**,**

0**,** 0**,** 1**,** 1**,** 1**,** 0**,** 0**,** //u

0**,** 1**,** 0**,** 1**,** 0**,** 1**,** 0**,**

0**,** 1**,** 1**,** 1**,** 1**,** 1**,** 1**,** //w

1**,** 0**,** 0**,** 1**,** 0**,** 0**,** 1**,**

0**,** 1**,** 1**,** 1**,** 0**,** 1**,** 1**,** //y

1**,** 1**,** 0**,** 1**,** 1**,** 0**,** 1**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0 //SPACE

**};**

void digitShow**(**uint8\_t value**)**

**{**

uint8\_t **\***p**=&**matrix**[**value**\***7**];**

SEG\_A\_LAT **=** **\***p**++;**

SEG\_B\_LAT **=** **\***p**++;**

SEG\_C\_LAT **=** **\***p**++;**

SEG\_D\_LAT **=** **\***p**++;**

SEG\_E\_LAT **=** **\***p**++;**

SEG\_F\_LAT **=** **\***p**++;**

SEG\_G\_LAT **=** **\***p**++;**

**}**

bool EMC1001\_Read**(**uint8\_t reg**,** uint8\_t **\***pData**);**

void clock\_mode**();**

void temp\_mode**();**

void stopw\_mode**(**int count**[]);**

bool decide **=** 0**;**

int time**[**4**]** **=** **{**0**,** 0**,** 0**,** 0**};**

int count**[**4**]** **=** **{**0**,** 0**,** 0**,** 0**};**

int track**,** count1**=**0**;**

void main**(**void**)**

**{**

uint8\_t digit**;**

CCDNA **|=** 0b10000000**;**//Enables constant current for pin RA7 (C4)

CCDNB **|=** 0b00011001**;**//Enables constant current for pin RB0(C1), RB3 (C2), RB4(C3)

CCDCON**=**LIMIT\_NONE**;**

SYSTEM\_Initialize**();**

INTERRUPT\_GlobalInterruptEnable**();**

INTERRUPT\_PeripheralInterruptEnable**();**

TMR1\_Initialize**();**

TMR1\_StartTimer**();**

C1\_SetHigh**();**//turns off all the displays

C2\_SetHigh**();**

C3\_SetHigh**();**

C4\_SetHigh**();**

SEG\_DP\_LAT **=** 0**;**

**while** **(**1**)**

**{**

//beginning of clock

**if((**ADCC\_GetSingleConversion**(**POT**)>>**5**)** **<**10.6**)**

**{**

count**[**0**]** **=** 0**;**

count**[**1**]** **=** 0**;**

count**[**2**]** **=** 0**;**

count**[**3**]** **=** 0**;**

track**=**0**;**

clock\_mode**();**

**}**

//end of clock

//beginning of temp

**if((**10.6 **<=(**ADCC\_GetSingleConversion**(**POT**)>>**5**))&&** **((**ADCC\_GetSingleConversion**(**POT**)>>**5**)** **<** 21.3**))**

**{**

count**[**0**]** **=** 0**;**

count**[**1**]** **=** 0**;**

count**[**2**]** **=** 0**;**

count**[**3**]** **=** 0**;**

track**=**0**;**

temp\_mode**();**

**}**

//end of middle temp

//beginning of stopwatch

**if((**ADCC\_GetSingleConversion**(**POT**)>>**5**)** **>=** 21.3**)**

**{**

stopw\_mode**(**count**);**

**}**

**}**

//end of stopwatch

**}**

void clock\_mode**()**

**{**

decide **=** 0**;** //keeps stopwatch button reset when switched out of mode

**if(**CLC1\_OutputStatusGet**()==**0**)**

**{**

C4\_SetHigh**();**//turns off 4th digit

digitShow**(**time**[**0**]);**

\_\_delay\_ms**(**1**);**

C1\_SetLow**();**//turns on 1st digit

\_\_delay\_ms**(**4**);**

C1\_SetHigh**();**//turns off 1st digit

digitShow**(**time**[**1**]);**

\_\_delay\_ms**(**1**);**

C2\_SetLow**();**//turns on 2nd digit

\_\_delay\_ms**(**4**);**

C2\_SetHigh**();**//turns off second digit

digitShow**(**time**[**2**]);**

\_\_delay\_ms**(**1**);**

C3\_SetLow**();**//turns on 3rd digit

\_\_delay\_ms**(**4**);**

C3\_SetHigh**();**//turns off 3rd digit

digitShow**(**time**[**3**]);**

\_\_delay\_ms**(**1**);**

C4\_SetLow**();**//turns on 4th digit

\_\_delay\_ms**(**4**);**

**}**

**if(**CLC1\_OutputStatusGet**()==**1**)**

**{**

**if(**time**[**0**]>=** 1 **&&** time**[**1**]>=**3**)**

**{**

C4\_SetHigh**();**//turns off 4th digit

digitShow**(**time**[**0**]-**1**);**

\_\_delay\_ms**(**1**);**

C1\_SetLow**();**//turns on 1st digit

\_\_delay\_ms**(**4**);**

C1\_SetHigh**();**//turns off 1st digit

digitShow**(**time**[**1**]-**2**);**

\_\_delay\_ms**(**1**);**

C2\_SetLow**();**//turns on 2nd digit

\_\_delay\_ms**(**4**);**

C2\_SetHigh**();**//turns off second digit

digitShow**(**time**[**2**]);**

\_\_delay\_ms**(**1**);**

C3\_SetLow**();**//turns on 3rd digit

\_\_delay\_ms**(**4**);**

C3\_SetHigh**();**//turns off 3rd digit

digitShow**(**time**[**3**]);**

\_\_delay\_ms**(**1**);**

C4\_SetLow**();**//turns on 4th digit

\_\_delay\_ms**(**4**);**

**}**

**else**

**{**

C4\_SetHigh**();**//turns off 4th digit

digitShow**(**time**[**0**]);**

\_\_delay\_ms**(**1**);**

C1\_SetLow**();**//turns on 1st digit

\_\_delay\_ms**(**4**);**

C1\_SetHigh**();**//turns off 1st digit

digitShow**(**time**[**1**]);**

\_\_delay\_ms**(**1**);**

C2\_SetLow**();**//turns on 2nd digit

\_\_delay\_ms**(**4**);**

C2\_SetHigh**();**//turns off second digit

digitShow**(**time**[**2**]);**

\_\_delay\_ms**(**1**);**

C3\_SetLow**();**//turns on 3rd digit

\_\_delay\_ms**(**4**);**

C3\_SetHigh**();**//turns off 3rd digit

digitShow**(**time**[**3**]);**

\_\_delay\_ms**(**1**);**

C4\_SetLow**();**//turns on 4th digit

\_\_delay\_ms**(**4**);**

**}**

**}**

**}**

void temp\_mode**()**

**{**

uint8\_t temp**;**

uint8\_t tempF**;**

uint8\_t temp\_Arr**[**3**];**

uint8\_t tempF\_Arr**[**3**];**

uint8\_t digit**;**

int i**=**0**;**

decide **=** 0**;** //keeps stopwatch button reset when switched out of its mode

**if(**CLC1\_OutputStatusGet**()==**0**)**

**{**

**if(**EMC1001\_Read**(**0**,** **(**uint8\_t**\*)&**temp**)){}**

tempF **=** **((**temp **\*** 9**/**5**)** **+** 32**);**

**if(**tempF**<**10**)**

**{**

tempF\_Arr**[**2**]=**tempF**;**

tempF\_Arr**[**1**]=**36**;**

tempF\_Arr**[**0**]=**36**;**

**}**

**if(**tempF**>=**10 **&&** tempF**<**100**)**

**{**

digit**=**tempF**/**10**;**

tempF\_Arr**[**1**]=(**int**)**digit**;**

tempF\_Arr**[**2**]=**tempF**%**10**;**

tempF\_Arr**[**0**]=**36**;**

**}**

**if(**tempF**>=**100**)**

**{**

digit**=(**tempF**/**100**);**

tempF\_Arr**[**0**]=(**int**)**digit**;**

digit**=(**tempF**/**10**)-**10**;**

tempF\_Arr**[**1**]=(**int**)**tempF**;**

tempF\_Arr**[**2**]=**tempF**%**10**;**

**}**

// 7 segment display F temp

C4\_SetHigh**();** //turns off 4th digit

digitShow**(**tempF\_Arr**[**0**]);**

\_\_delay\_ms**(**1**);**

C1\_SetLow**();** //turns on 1st digit

\_\_delay\_ms**(**4**);**

C1\_SetHigh**();** //turns off 1st digit

digitShow**(**tempF\_Arr**[**1**]);**

\_\_delay\_ms**(**1**);**

C2\_SetLow**();** //turns on 2nd digit

\_\_delay\_ms**(**4**);**

C2\_SetHigh**();** //turns off second digit

digitShow**(**tempF\_Arr**[**2**]);**

\_\_delay\_ms**(**1**);**

C3\_SetLow**();** //turns on 3rd digit

\_\_delay\_ms**(**4**);**

C3\_SetHigh**();** //turns off 3rd digit

digitShow**(**15**);**

\_\_delay\_ms**(**1**);**

C4\_SetLow**();** //turns on 4th digit

\_\_delay\_ms**(**4**);**

**}**

**if(**CLC1\_OutputStatusGet**()==**1**)**

**{**

**if(**EMC1001\_Read**(**0**,** **(**uint8\_t**\*)&**temp**)){}**

**if(**temp**<**10**)**

**{**

temp\_Arr**[**2**]=**temp**;**

temp\_Arr**[**1**]=**36**;**

temp\_Arr**[**0**]=**36**;**

**}**

**if(**temp**>=**10 **&&** temp**<**100**)**

**{**

digit**=**temp**/**10**;**

temp\_Arr**[**1**]=(**int**)**digit**;**

temp\_Arr**[**2**]=**temp**%**10**;**

temp\_Arr**[**0**]=**36**;**

**}**

**if(**temp**>=**100**)**

**{**

digit**=(**temp**/**100**);**

temp\_Arr**[**0**]=(**int**)**digit**;**

digit**=(**temp**/**10**)-**10**;**

temp\_Arr**[**1**]=(**int**)**temp**;**

temp\_Arr**[**2**]=**temp**%**10**;**

**}**

C4\_SetHigh**();** //turns off 4th digit

digitShow**(**temp\_Arr**[**0**]);**

\_\_delay\_ms**(**1**);**

C1\_SetLow**();** //turns on 1st digit

\_\_delay\_ms**(**4**);**

C1\_SetHigh**();** //turns off 1st digit

digitShow**(**temp\_Arr**[**1**]);**

\_\_delay\_ms**(**1**);**

C2\_SetLow**();** //turns on 2nd digit

\_\_delay\_ms**(**4**);**

C2\_SetHigh**();** //turns off second digit

digitShow**(**temp\_Arr**[**2**]);**

\_\_delay\_ms**(**1**);**

C3\_SetLow**();** //turns on 3rd digit

\_\_delay\_ms**(**4**);**

C3\_SetHigh**();** //turns off 3rd digit

digitShow**(**12**);**

\_\_delay\_ms**(**1**);**

C4\_SetLow**();** //turns on 4th digit

\_\_delay\_ms**(**4**);**

// 7 segment display C temp

**}**

**}**

void stopw\_mode**(**int count**[])**

**{**

int i**,** time**;**

**if** **(**decide **==** 0**)**

**{**

TMR0\_StopTimer**();**

C4\_SetHigh**();** //turns off 4th digit

digitShow**(**count**[**0**]);**

\_\_delay\_ms**(**1**);**

C1\_SetLow**();** //turns on 1st digit

\_\_delay\_ms**(**4**);**

C1\_SetHigh**();** //turns off 1st digit

digitShow**(**count**[**1**]);**

\_\_delay\_ms**(**1**);**

C2\_SetLow**();** //turns on 2nd digit

\_\_delay\_ms**(**4**);**

C2\_SetHigh**();** //turns off second digit

digitShow**(**count**[**2**]);**

\_\_delay\_ms**(**1**);**

C3\_SetLow**();** //turns on 3rd digit

\_\_delay\_ms**(**4**);**

C3\_SetHigh**();** //turns off 3rd digit

digitShow**(**count**[**3**]);**

\_\_delay\_ms**(**1**);**

C4\_SetLow**();** //turns on 4th digit

\_\_delay\_ms**(**4**);**

//display count array on 7-segment

**}**

**if** **(**decide **==** 1**)**

**{**

TMR0\_StartTimer**();**

C4\_SetHigh**();** //turns off 4th digit

digitShow**(**count**[**0**]);**

\_\_delay\_ms**(**1**);**

C1\_SetLow**();** //turns on 1st digit

\_\_delay\_ms**(**4**);**

C1\_SetHigh**();** //turns off 1st digit

digitShow**(**count**[**1**]);**

\_\_delay\_ms**(**1**);**

C2\_SetLow**();** //turns on 2nd digit

\_\_delay\_ms**(**4**);**

C2\_SetHigh**();** //turns off second digit

digitShow**(**count**[**2**]);**

\_\_delay\_ms**(**1**);**

C3\_SetLow**();** //turns on 3rd digit

\_\_delay\_ms**(**4**);**

C3\_SetHigh**();** //turns off 3rd digit

digitShow**(**count**[**3**]);**

\_\_delay\_ms**(**1**);**

C4\_SetLow**();** //turns on 4th digit

\_\_delay\_ms**(**4**);**

//display count array on 7-segment

**}**

**}**

bool EMC1001\_Read**(**uint8\_t reg**,** uint8\_t **\***pData**)**

**{**

I2C2\_MESSAGE\_STATUS status **=** I2C2\_MESSAGE\_PENDING**;**

static I2C2\_TRANSACTION\_REQUEST\_BLOCK trb**[**2**];**

I2C2\_MasterWriteTRBBuild**(&**trb**[**0**],** **&**reg**,** 1**,** EMC1001\_ADDRESS**);**

I2C2\_MasterReadTRBBuild**(&**trb**[**1**],** pData**,** 1**,** EMC1001\_ADDRESS**);**

I2C2\_MasterTRBInsert**(**2**,** **&**trb**[**0**],** **&**status**);**

**while(**status **==** I2C2\_MESSAGE\_PENDING**);**

**return** **(**status **==** I2C2\_MESSAGE\_COMPLETE**);**

**}**

/\*\*

End of File

\*/