Physics 319 Final Report: Alarm Clock

Drew Spencer

April 8, 2019

Abstract

This project will be using an MSP430G2553 microprocessor to create an alarm clock. The output will be an Optrex 20434-cwm LCD screen which utilizes a HD44780 Driver for communication from the MSP430. The HD44780 is the main communication piece for the LCD screen. The communication between the microprocessor and HD44780 is done through 14 different pins. The clock itself was successful in keeping track of seconds, minutes, hours, days, months, and years and outputting updated numbers to the LCD display. The alarm used a piezoelectric "beeper" to emit a sound at a specific time. The calibration of the clock was very close to an actual second, being off by about being withing less than half a millisecond. This is done through the internal peripherals of the microprocessor which will be explained more in depth later in the paper and how the calibration could be improved.

1 Introduction

The main idea behind the choice for this project was getting to understand the some of the peripherals in the microprocessor MSP430G2553. Through many of our labs we used different components in the microprocessor called peripherals which have specific functions that we can utilize to our advantage. One of these internal functions is the internal clock system which can be initialized through certain commands. This gave the idea to try and create an alarm clock. An alarm clock would need to utilize the clock systems to be able to keep track of time and thus give an output. An LCD screen, Optrex 20434, was connected to the MSP430 to display the time. Using an LCD screen and figuring out the pin structure and communication of another device rather than just a seven segment display which we already used in labs 1 and 2 was an added challenge as no other students used an LCD display. Once these were linked together, a clock with an alarm would be the result. Clocks are used everyday by almost every one in the world in one way or another. Making something that we use in our everyday lives would be interesting and practical to show others.

2 Theory

2.1 MSP430 Clock Peripherals

The main peripheral being used is the clock system in the MSP430 device. The Basic clock System Control Register (BCSCTL) activates the basic clock system inside the microprocessor. Setting this register to 1 mega hertz (1Mhz) which stores the value of 1000000 hz in the clock system . BCSCTL1 = CALBC1_1MHZ. Once this is set, the timer A clocks (TACCTL) were used to keep track of the count using the sub main clock (SMCLK) through (TASSEL_2) command, counting in up mode with MC_1 and then dividing the 1 Mhz term by 8 with the ID_3 command. These are shown below and all information was gathered from the MSP430 data sheets [5]:

TACCTL=TASSEL_2 + MC_1 + ID_3

TACCR0 = 2500 this is the number the Timer A clock will count to.

TACCTL0 = CCIE this enables the interrupt routine.

Here the Timer A is initialized to count to the specified number in TACCR0. Since the clock counts in microseconds, it made sense to initialize at 1 Mhz as 1000000 microseconds = 1 second and 1 Mhz is 1000000 hertz.These would balance each other out. For the division by 8 through ID_3, the clock value was now 125000 hz to count to one second. Now the 125000 number will be divided by the value set to CCR0 is set to, say 2500.

TACCR0 = 2500

What happens now is $\frac{125000}{2500} = 50$

Seeing that the result is 50, the clock will increment to 2500 and reset to zero 50 times for one second to pass.

2.2 Optrex-20434 LCD Display

The Optrex-20434 LCD with HD44780 controller is the device that will take the given output of time and display it to the user. It communicates and writes the updated information through 14 pins with the HD44780 controller. Out of the 14 pins, 8 of them are specific to what characters are written to the specified part of the screen. This can be done in 8 bit mode, which uses all 8 pins or 4 bit mode which uses 4 of the 8 pins. In this experiment, 4 bit mode is used and thus 4 of the 8 pins are left open. The diagram below will show the pin structure for the display.

No.	Symbol	Level	Function
1	V_{SS}	–	Power Supply (0V, GND)
2	V_{CC}	–	Power Supply for Logic
3	V_{EE}	–	Power Supply for LCD Drive
4	RS	H / L	Register Select Signal
5	R/W	H / L	Read/Write Select Signal H : Read L : Write
6	E	H,H→L	Enable Signal (No pull-up Resister)
7	DB0	H / L	Data Bus Line / Non-connection at 4-bit operation
8	DB1	H / L	Data Bus Line / Non-connection at 4-bit operation
9	DB2	H / L	Data Bus Line / Non-connection at 4-bit operation
10	DB3	H / L	Data Bus Line / Non-connection at 4-bit operation
11	DB4	H / L	Data Bus Line
12	DB5	H / L	Data Bus Line
13	DB6	H / L	Data Bus Line
14	DB7	H / L	Data Bus Line

Figure 1: Pin format of LCD Display. 14 pins with one starting the leftmost pin 14 being the rightmost pin when looking directly at screen. Picture is taken directly from the Optrex 20434 Data sheet [1]

3 Apparatus

There are 3 main components to the build of this project, one being the MSP430 another being the Optrex 20343 LCD display, and the last being the breadboard which everything is connected to. The breadboard power the device and allows easy connections through its various pin layouts. The breadboard also has the potentiometer built in, which is used for the contrast of the LCD screen and connected to pin 3 of the LCD display. The potentiometer is essentially a variable resistor that will allow the user to control the voltage/current supply to the screen by turning a small knob to change the resistance. The supply voltage and ground are also directly linked through the breadboard which allows for easier distribution through the circuit as there is much more space to work with. A smaller component connected to the breadboard for the project is the peizoelectric beeper. This is the device that will emit a sound at a given frequency established in the code. It is set through the CCR0 register for the period. The smaller/shorter the period the highest the pitch of the beeper. The duty cycle is set with the CCR1 register and needs to be a smaller value than is stored in the CCR0 register for it to emit a sound.

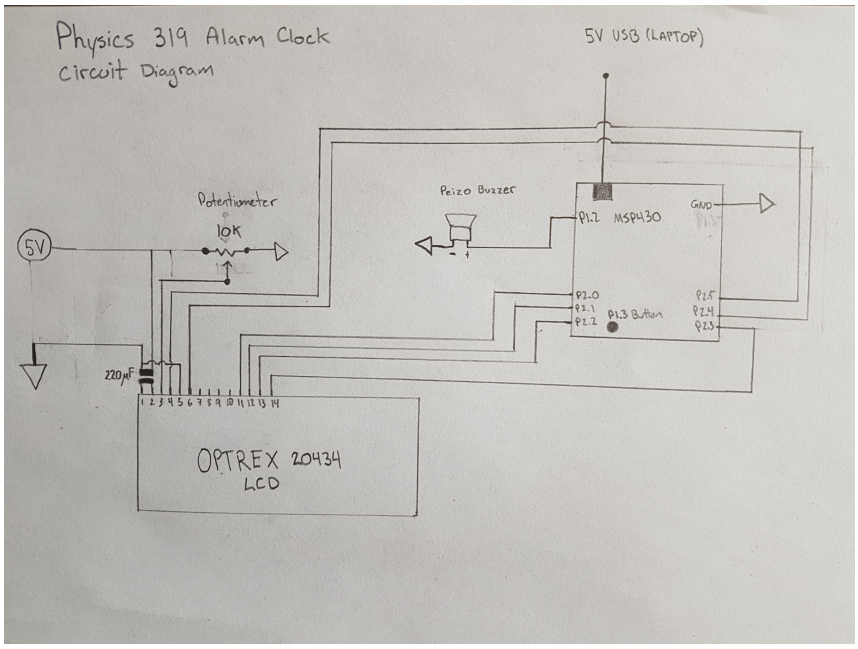

Figure 2: Above is the Circuit diagram for my project.

3.1 MSP430 Functions

For the MSP430 there are a few integral functions used including the aforementioned clock controls. Along with the clocks, there is a couple interrupt routines that, if not defined, would cause the project to not work at all. The first interrupt routine runs the entire update portion for the time. With the clock system stated earlier in the Theory section 2.1 this variable increments every time the TACCR0 register counts up to 2500. This is done automatically by initiating a function which again was written with:

TACCTL=TASSEL_2 + MC_1 + ID_3

TACCR0 = 2500 this is the number the Timer A clock will count to.

TACCTL0 = CCIE this enables the interrupt routine.

This was the initialization sequence for the counter. Now the variable that increments every time the interrupt is triggered (when TACCR0 counts to 2500 and resets) will count up to 50, once this happens another variable will increment by 1 which denotes 1 second has passed. This continues until seconds increment 60 times which then a variable for minutes will increment by one. This continues on for hours, days, months, years hcih all increment at their own respective rates. These values are all output to the screen which is connected to the MSP430 with a certain pin format which will be explained further down. A second interrupt routine was created to stop the alarm (peizoelectric beeper) from emitting a sound. When a certain time hits, which is defined in the code by the user, P1.2 from the MSP430 is set to output a PWM routine which we utilized in lab 3. The code is taken directly from there

with slight modifications to allot for consistent clock counting. Once this happens, the P1.3 button on the MSP device is activated as an interrupt. What this button is programmed to do is when pushed by the user, the P1.2 output is flipped off and the beeper stops making noise, shutting off the alarm portion. When this happens, the interrupt flag is flipped and the routine is exited.

3.2 Pin layout for MSP430 and LCD Display

Now for the pin-out of the MSP430 and the LCD display. On the MSP430 the port 2 pins were used to free up the port 1 pins for the interrupt routines and PWM sequence. The structure of the pins were taken from a University of Texas el Paso website. This website also supplied an easy to interpret library which is noted in the reference section for initializing and writing commands for the LCD. There was a total of 6 pins connected directly to the LCD display with the LCD using 10 pins in total which are all listed at the top of the next page:

LCD PIN	SYMBOL	CONNECTION	FUNCTION
1	GND	Ground	Ground Connection
2	Vcc	V_{cc}	Voltage Source to Power Display
3	Vee	Potentiometer	Screen/Character Contrast
4	RS	MSP P2.5	Register Select (write command or data)
5	R/W	Ground	Always to ground in read mode
6	E	MSP P2.4	Falling edge to trigger operation
7	DB0	N/A	No need to connect in 4 bit mode
8	DB1	N/A	No need to connect in 4 bit mode
9	DB2	N/A	No need to connect in 4 bit mode
10	DB3	N/A	No need to connect in 4 bit mode
11	DB4	MSP P2.0	Data Bus Line
12	DB5	MSP P2.1	Data Bus Line
13	DB6	MSP P2.2	Data Bus Line
14	DB7	MSP P2.3	Data Bus Line

Table 1: This shows the pin connections and the function of each one. Pin 11-14 are the controls for the characters to be written. The sequence starts with the most significant bit. The four "bits" put together create a nibble, which corresponds to a specific character to be written. [2]

3.3 How to Use the Clock

The alarm clock itself it very quick and simple to use. Everything is controlled through the code such as setting the time and date and when the alarm goes off. Improvements to this system are mentioned further in the paper. Once the code is uploaded, the alarm clock will run on its own as long as power is supplied to the devices that need it, such as the breadboard LCD and MSP430. There is no manual control outside of the code for the clock to be updated. As for the writing to the screen, there is an initialization sequence that is

written so that the LCD can be used. If this sequence isn't done properly, the screen will not display anything. This was taken from The University of Texas el Paso EE3376 website [2] with the lcd.Lib.h [3] and lcd.Lib.c [4] files. Some of these were changed around as their screen was a 16x2 (16 columns by 2 rows) and the one used for this project was a 20x4 screen. Some of the functions were changed around as well to allot for this and functions were added into the library.

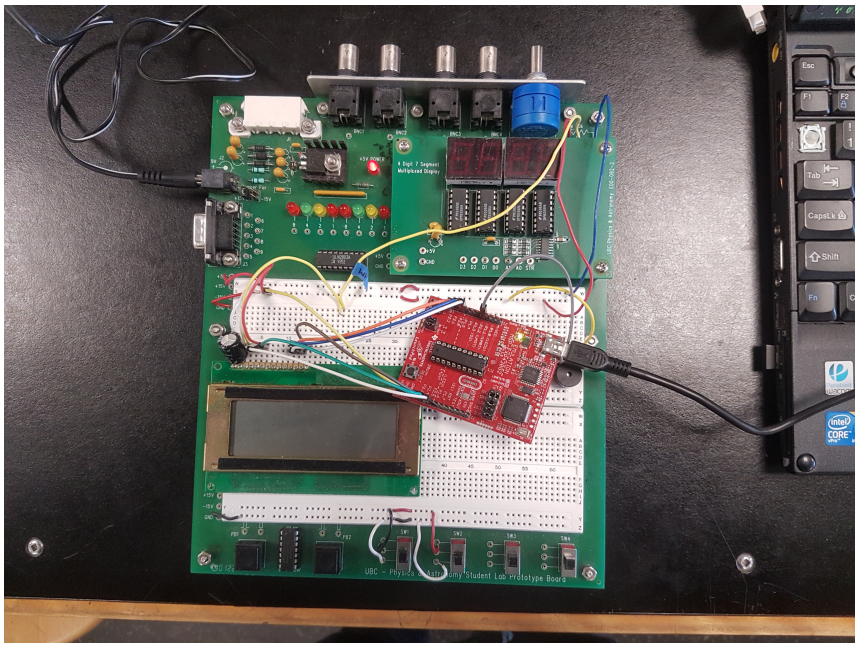

Figure 3: The build of the project including all devices. More figures are shown in the appendix. Both are shown included the display showing the date and time. One if after the rollover of midnight and the change of the date and day along with the time updateing as well.

4 Results

Overall the alarm clock ended up working quite well. The time updated correctly and was able to update multiple values/elements such as seconds, minutes, hours, days and months all at the same time. This took quite a while to program correctly as there were many different delays, display clears, and specific sequences to write that needed to be taken into account. The first few times going through, the understanding of the cursor needed time to shift around the LCD screen to be able to write properly was overlooked. If the time between writing functions is too short, it will overwrite the characters wherever the cursor is at when trying to shift to its new position. To counter this issue, the sequence of writing was changed to start at the first column and first row, and proceed to write that whole row before going to the next one limit how much the cursor moved. This coupled with delays in between each writing command and clearing the screen after each minute increment allowed to LCD to update cleanly without problems.

As for the alarm portion, it worked as expected. A time was set for the alarm to go off and it went off every time without problems. The push button was a bit finicky at first but with some slight modifications it was not a hard fix as all that needed to be done was stop P1.2 from being an output. Once this was done there we no further issues with the P1.3 button press to stop the beeper from emitting a sound.

A couple problems were encountered at the very beginning of the project. These problems were resolved in a very simple way as they were due to a lack of awareness. One of these issues was just simply through having the contrast of the screen fairly low, thus making the characters faint. As there is a certain viewing angle mentioned in the Optrex data sheet, which was clearly overlooked, I couldn't see the characters on the screen. As even without the LCD initialization there should be some black squares on the screen without any text. As embarrassing as it is to say, it took me about 4 hours overall to realize what was going wrong, as once I sat down in my chair and changed my viewing angle, the black boxes were visible. Another issue that arose was through an improper interrupt vector definition. In one of our previous labs, we used a TI definition for an interrupt vector. This was carried over into this project for the purpose of the timer interrupt vector to keep track of time. As much of the documentation for interrupts were defined like this online, it took quite a figure out what the problem was. Peter Gysbers, one of our teaching assistants helped troubleshoot this problem and all that was needed was to define the interrupt service routine differently. Once this was done everything was taken care of and the program ran smoothly.

5 Discussion

There are a few improvements that stand out to make the alarm clock more efficient and accurate. Originally my plan was create a snooze button that would work with 2 or 3 button presses. Over the course of writing the code for this, it became more and more difficult to implement more routines. Optimizing the code may have helped this by getting rid of redundancies and allowing more routines to be written overall. The motivation behind having a snooze button that could be used multiple times was the fact that I use the snooze button on my clock quite often. This is something that can easily be played around with in the future, but in the time frame of this project was not something that was achieved.

As for the actual time calibration of the clock, it was hard to get the clock to count at an exact second pace. The peripherals in the MSP430 can only be changed so much. Coupling this with the need for different delays throughout the program to allow for correct updates, the time can be skewed even more. While the difference was very small, maybe about 10 microseconds, over time this number adds up to give a difference in the coded time, to the actual time. To improve this a real time clock module/sensor can be integrated into the circuit which would eliminate the peripheral differences. This would work to keep perfect time, but the main idea behind this project was to use and understand the peripherals better so this was not considered a viable option.

The last change I would have made was the structure of the system. Using the breadboard made the project very bulky and impractical for use as an alarm clock. I would have liked to have a smaller board that can be powered with a battery as opposed to being plugged into a wall outlet. Though many components were direct components of the breadboard, it would be easy to implement these externally and create a more compact device. Along with this I would like to try and add some push buttons to enable the user to change the times and alarm setting without having to open up a coding platform and change the code explicitly. This would be much more efficient as alarms are typically changed all the time and having to hook up the a computer everyday would be quite inefficient.

6 Conclusion

Overall the project was a success as it was able to do all the processes I wanted to include. The time keeping was done through the clock peripherals in the MSP430 and successfully output to the LCD display. Though there is minor calibration issues with the time being exact, it still keeps track of time consistently and the alarm works consistently without any issues. As for the screen, after a couple of troubleshooting sessions, it displayed and updated everything well without any distortions or errors. The project ended up running cleanly and was a fun project to help understand how the clock peripherals of the MS430 are working. It has peaked my interest in micro-controllers quite a bit as I was very unfamiliar with them beforehand. They are an incredibly versatile piece of equipment that have a wide variety of applications from making an alarm clock, to powering a remote control car. It was especially nice to finally get a stronger understanding of how software and hardware work together, and how to analyze issues that arise from the cross over.

7 Acknowledgements

In my code I use a two libraries from the University of Texas el Paso EE3376 Lab 5 website. Though I do use the general idea of these functions, mine are slightly modified to allot for the fact that they use a 16x2 LCD screen and I use a 20x4 LCD screen. I also had to add some code into their lcdSetText command as again they only implemented a 16x2 screen, I also created my own function to take an updating integer variable which I called lcdPrintSec. The bulk of my code has no overlap with theirs as they only supply a brief outline to initialize the LCD and some of the commands. The main reason I used their libraries was because the initialization sequence from the data sheet did not work, therefore I didn't want to try and use the rest of data sheet information in case it was wrong. There are certain functions that were not changed from the original content, and as there wasn't really any other way for me to write them in accordance with some of the definitions. I give them a lot of credit for sourcing their code online to be used by others, and am very grateful that they did. While I do use some of their functions explicitly, I did take the time to understand them and thus be able to use them properly.

The unchanged functions are listed below:

- lcdInit
- lcdClear
- lcdTriggerEN
- lcdWriteCmd
- lcdWriteData

The rest of the functions, interrupts and code were either written by myself or taken from our previous lab work in the course.

Another huge thank you to Colby DeLisle, Peter Gysbers, and Dr. Andrezj Kotlicki as without their help and guidance, much of my project would not have been completed. All three were a major help on my ability to work on this project and complete to my satisfaction. I highly appreciate the time and effort they put in to helping me understand the concepts and materials for this course.

8 Appendix

```
#include <msp430.h>
#include "lcdLib.h"

#define LOWNIB(x) P2OUT = (P2OUT & 0xF0) + (x & 0x0F)

//set arrays as char* pointers, this was due to int not being able to be 00,
//01, 02 etc.  Strings could take what was wanted

char* time_val[60] ={"00","01","02","03","04","05","06","07","08","09","10","11","12",
"13","14","15","16","17","18","19","20","21","22","23","24","25","26","27","28","30","31","32","33","34","35","36","37","38","39",
"40","41","42","43","44","45","46","47","48","49","50","51","52","53","54","55","56","57","58","59"};
char* months[12] ={"Jan/","Feb/","Mar/","Apr/","May/","Jun/","Jul/","Aug/","Sep/","Oct/","Nov/","Dec/"};

char* days[30] ={"01/","02/","03/","04/","05/","06/","07/","08/","09/","10/","11/","12/",
"13/","14/","15/","16/","17/","18/","19/","20/","21/","22/","23/","24/","25/","26/","27/","28/","30/","31/"};

//global variable to keep track of time, day, month etc.
volatile int count=0, snooze=0, twice=0, check=0, compare=0, timercount=0;
//setting this keeps track of the seconds on the clock
volatile int seconds = 51;
//initial times of clock n=seconds, l=minutes, m=hours
volatile int n = 50, l = 58, m = 23;
//day and month set
volatile int day = 28, month = 8;
void main(void)

{
//these enable the LED of P1.0 to be on when clock starts
//sets interrupt to P1.3 to run ISR
P1DIR = BIT0 + BIT2;
P1OUT = BIT0;
P1REN = BIT3;
P1IE = BIT3;
P2DIR = EN + RS + DATA;
WDTCTL = WDTPW + WDTHOLD;// Stop Watchdog
BCSCTL1 = CALBC1_1MHZ; //sets basic clock register to 1 Mhz
DCOCTL = CALDCO_1MHZ; //

lcdInit();// Initialize LCD

counterInit(); // initialize clock routines

lcdClear(); // clear screen before writing anything, just to be safe

//sets up the initial readings of the screen, can set these to any
//value from the array, just make sure to set the stuff in the interrupt
//these all take the values from the global initialization of each
//variable name and correspond the the array element

```

```
50 delay_ms(10);
51 lcdSetText("", 0,0);
52 delay_us(1000);
53 lcdSetText(days[day], 4,1);
54 delay_us(1000);
55 lcdSetText(months[month], 0,1);
56 lcdSetText("2019", 7, 1);
57 delay_us(1000);
58 lcdSetText("H M     S    ", 0,2);
59 delay_us(1000);
60 lcdSetText(time_val[m], 0,3);
61 delay_us(1000);
62 lcdSetText(time_val[l], 6,3);
63 delay_us(1000);
64 lcdPrintsec(seconds, 8,3);
65 delay_us(1000);
66
67 __bis_SR_register(GIE); // Enter interrupts but no low power mode, stops
       clocks
68
69 //this loops is what updates the screen, sequence of writing is necessary
70 //follow left most to right most, and top most to bottom most
71 //need delays to lalow time for cursor to shift cleanly
72
73   while (1){
74       delay_us(5000);
75       lcdSetText("PHYS 319", 0,0);
76       delay_us(5000);
77       lcdSetText(months[month], 0,1);
78       delay_us(5000);
79       lcdSetText(days[day], 4,1);
80       delay_us(5000);
81       lcdSetText("2019", 7, 1);
82       delay_us(5000);
83       lcdSetText("H M     S       ", 0,2);
84       delay_us(5000);
85       lcdSetText(time_val[m], 0,3);
86       delay_us(5000);
87       lcdSetText(time_val[l], 6,3);
88       delay_us(5000);
89       lcdPrintsec(seconds, 8,3);
90       delay_us(5000);
91 }
92 }
93
94 //initialization sequence to get the LCD to turn on properly and take command
95 // sets certin pins as outputs which allow the ability to write to the LCD
96 //The read write register is connected to ground which means always read mode.
97 //starts the program in 4 bit mode which only uses 4 pins to write
98 void lcdInit() {
99   delay_ms(100);
100  // Wait for 100ms after power is applied.
101
102  P2DIR = EN + RS + DATA;  // Make pins outputs
103  P2OUT = 0x03;  // Start LCD (send 0x03)
104
105  lcdTriggerEN(); // Send 0x03 3 times at 5ms then 100 us
```

```
106   delay_ms(10);
107   lcdTriggerEN();
108   delay_ms(10);
109   lcdTriggerEN();
110   delay_ms(10);
111
112   P2OUT = 0x02; // Switch to 4-bit mode
113   lcdTriggerEN();
114   delay_ms(10);
115
116   lcdWriteCmd(0x48); // 4-bit, 2 line, 5x8
117   lcdWriteCmd(0x08); // Instruction Flow
118   lcdWriteCmd(0x01); // Clear LCD
119   lcdWriteCmd(0x06); // Auto-Increment
120   lcdWriteCmd(0x0C); // Display On, No blink
121 }
122
123 //enables signal high or low
124 void lcdTriggerEN() {
125   P2OUT |= EN;
126   P2OUT &= ~EN;
127 }
128
129 //flicks cursor to and shifts nibbles right 4, nibbles are four bit
130 // since we are in 4 bit mode nibbles, lownib refer to xxxx0000 the last 4
      bits
131 void lcdWriteData(unsigned char data) {
132   P2OUT |= RS; // Set RS to Data
133   LOWNIB(data >> 4); // Upper nibble
134   lcdTriggerEN();
135   LOWNIB(data); // Lower nibble
136   lcdTriggerEN();
137   delay_us(50); // Delay > 47 us
138 }
139
140 //this function follows same principle as above
141 void lcdWriteCmd(unsigned char cmd) {
142   P2OUT &= ~RS; // Set RS to Data
143   LOWNIB(cmd >> 4); // Upper nibble
144   lcdTriggerEN();
145   LOWNIB(cmd); // Lower nibble
146   lcdTriggerEN();
147   delay_ms(5); // Delay > 1.5ms
148 }
149 //this function sets what row and column to write message to
150 //x indicates row, y indicates column
151 //eg   x = 0, y = 0, sets to upper left corner of screen (1st row, 1st column)
152 void lcdSetText(char* text, int x, int y) {
153   int i;
154   if (x < 16) {
155     x |= 0x80; // Set LCD for first line write
156     switch (y){
157     case 1:
158       x |= 0x40; // Set LCD for second line write
159       break;
160     case 2:
161       x |= 0x14; // Set LCD for third line write
```

```
      break;
    case 3:
      x |= 0x54; // Set LCD for fourth line write
      break;
    case 4:
      x |= 0x60; // Set LCD for first line write reverse
      break;
    case 5:
      x |= 0x20; // Set LCD for second line write reverse
      break;
    case 6:
      x |= 0x34; // Set LCD for third line write reverse
      break;
    case 7:
      x |= 0x74; // Set LCD for fourth line write reverse
      break;
    }
    lcdWriteCmd(x);
  }
  i = 0;

  while (text[i] != '\0') {
    lcdWriteData(text[i]);
    i++;
  }
}
//fully clears/wipes LCD screen
void lcdClear() {
  lcdWriteCmd(CLEAR);
}

//function made to take a string value and output seconds.
void lcdPrintsec(int sec, int x, int y){
   char number_string[16];
   sprintf(number_string, "%d", sec);
   lcdSetText(number_string, x, y);
}

void counterInit(void){
    // Enable Timer Interrupts
  TACCTL0 = CCIE;
    // SMCLK, Count to CCR0 Value, Divide by 8
    TACTL = TASSEL_2 + MC_1 + ID_3;
// bases timer off SMCLK, at 1MHz, MC_1 indicates timer will count up reset to
    0
//after interrupt ID_3 will count up once every 8th clock count
  TACCR0 = 2500;
// 1,000,000 / 8 / 50 = 2500, this is the stored value that MC_1 counts to
}
//interrupt function to occur when SMCLK counts to specified value of CCR0
//as 1Mhz is divided by 8 giving 125000, divide this number by CCR0
//value and set a variable to count that many times to increment 1 second
//time flows as expected after that. Clock is in 24 hour mode
//blink function set inside to ensure interrupt is working

void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) Timer_A(void){
  P1OUT ^= BIT0 + BIT6;
```

```
218
219 timercount++;
220   if (timercount >= 51){
221       timercount = 0;
222   delay_us(10);
223       seconds++;
224   delay_us(10);
225   n++;
226       if (n >= 59){
227           seconds = 0;
228     n=0;
229     l++;
230     delay_ms(5);
231     lcdSetText("PHYS 319", 0,0);
232     delay_us(1000);
233     lcdSetText(months[month], 0,1);
234     delay_us(1000);
235     lcdSetText(days[day], 4,1);
236     delay_us(1000);
237     lcdSetText("2019", 7, 1);
238     delay_us(1000);
239     lcdSetText("H M   S        ", 0,2);
240     delay_us(1000);
241     lcdSetText(time_val[m], 0,3);
242     delay_us(1000);
243     lcdSetText(time_val[l], 6,3);
244     delay_us(1000);
245     lcdPrintsec(seconds, 6,3);
246     delay_us(1000);
247     lcdClear();
248
249     if (l >= 59){
250       l=0;
251       m++;
252       lcdClear();
253     if (m >= 23) {
254       m=0;
255       delay_us(100);
256       day++;
257       delay_us(100);
258       lcdClear();
259
260       if (day == 28 && month == 2){
261         delay_us(1000);
262         month++;
263         delay_us(1000);
264         day = 0;
265         lcdClear();
266       }
267       else if (day == 31 && month == 0 || 2 || 3 || 6 || 7 || 9 || 11){
268           month++;
269           delay_us(1000);
270           day = 0;
271           lcdClear();
272       }
273       else {
274           day = 0;
```

```
275             delay_us(1000);
276             month ++;
277             lcdClear();
278 
279         }
280 //sets alarm to go off at specified time in statement
281 //CCR0 set to same time frame as counterinit function so clock counts
282 //at same rate when interrupt happens and doesn't change timing
283           if(m == 0 && l == 0){
284             CCR0 = 2501-1; //period set for PWM
285             CCR1 = 250;   //duty cycle for PWM
286             CCTL1 = OUTMOD_7; //CCR1 reset/set
287             P1DIR |= BIT2;
288             P1SEL |= BIT2;
289             P1REN = BIT3;                   //enable resistor
290             P1IE = BIT3;
291             P1OUT = BIT3;
292       //checks conditions that increment from interrupt routine
293 
294             }
295         }
296       }
297     }
298     }
299 }
300 //interrupt routine to shut off beeper/alarm
301 void  __attribute__ ((interrupt(PORT1_VECTOR))) button(void)
302 
303 {
304 snooze++;
305 twice++;
306 check++;
307 count = m; // count set to minutes when Int. activated
308 counterInit(); // sets timer to be at same levels as clock system to not
        change count rate
309 while(1){
310       if(compare != count + 1){
311       //delay_us(5);
312       compare = (count + 1) ; // as minutes keep counting,
313       }
314       else{
315         break; // breaks loop
316       }
317   P1DIR ^= BIT2; //switches p1.2 to input to shut off output to alarm
318     P1IFG &= ~BIT3; // clears interrupt flag to stop function
319   }
320 
321 }
```

Listing 1: code

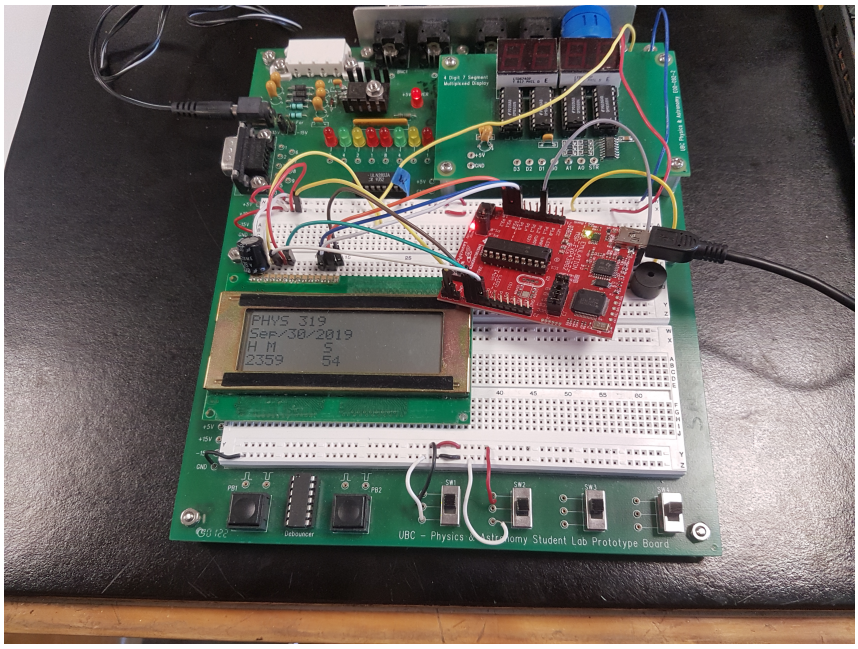

Figure 4: This is an image of the clock right before a change over at midnight from September 30, 2019 to October 1. 2019

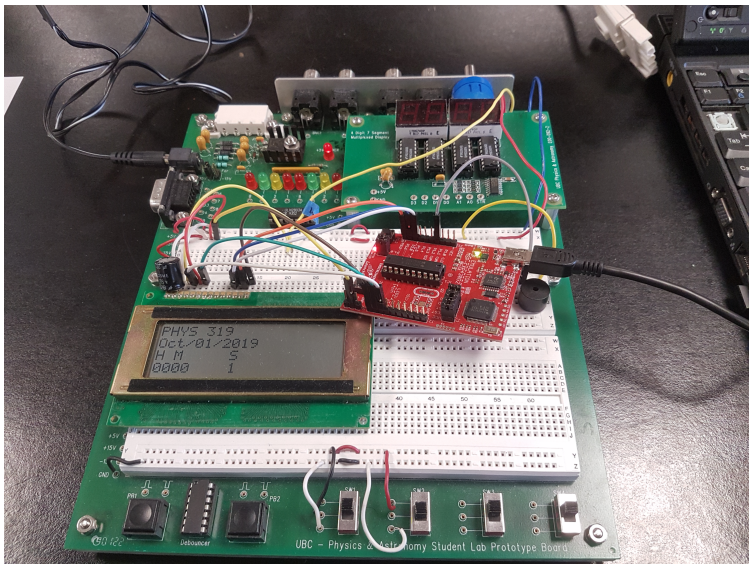

Figure 5: Here is the clock right after the change over showing the updated time in 24 hour mode with the month and day also updated cleanly.

References

[1] Optrex Data Sheet. (1999). *LCD Module Specification : DMC 20434.* pp. 7, Accessed Mar. 16 2019. Retrieved from: https://datasheet.octopart.com/DMC-20434-Optrex-datasheet-91238.pdf

[2] University of Texas el Paso: EE3376 Lab 5. Retrieved from: `http://www.ece.utep.edu/courses/web3376/Lab_5_-_LCD.html` Accessed Mar. 19 2019

[3] University of Texas el Paso: EE3376 Lab 5. Retrieved from: `http://www.ece.utep.edu/courses/web3376/Lab_5_-_LCD_files/lcdLib.h` Accessed Mar. 19 2019

[4] University of Texas el Paso: EE3376 Lab 5. Retrieved from: `http://www.ece.utep.edu/courses/web3376/Lab_5_-_LCD_files/lcdLib.c` Accessed Mar. 19 2019

[5] Texas Instruments. (2004). *MSP430x2xx Family: User Guide.* Retrieved from: `http://www.ti.com/lit/ug/slau144j/slau144j.pdf.`