|  |
| --- |
|  |
| Lab 2: Checksum and LCD Demo |
| ETEC 454 |
|  |
| **Mark Moerdyk** |
| **2/1/2013** |

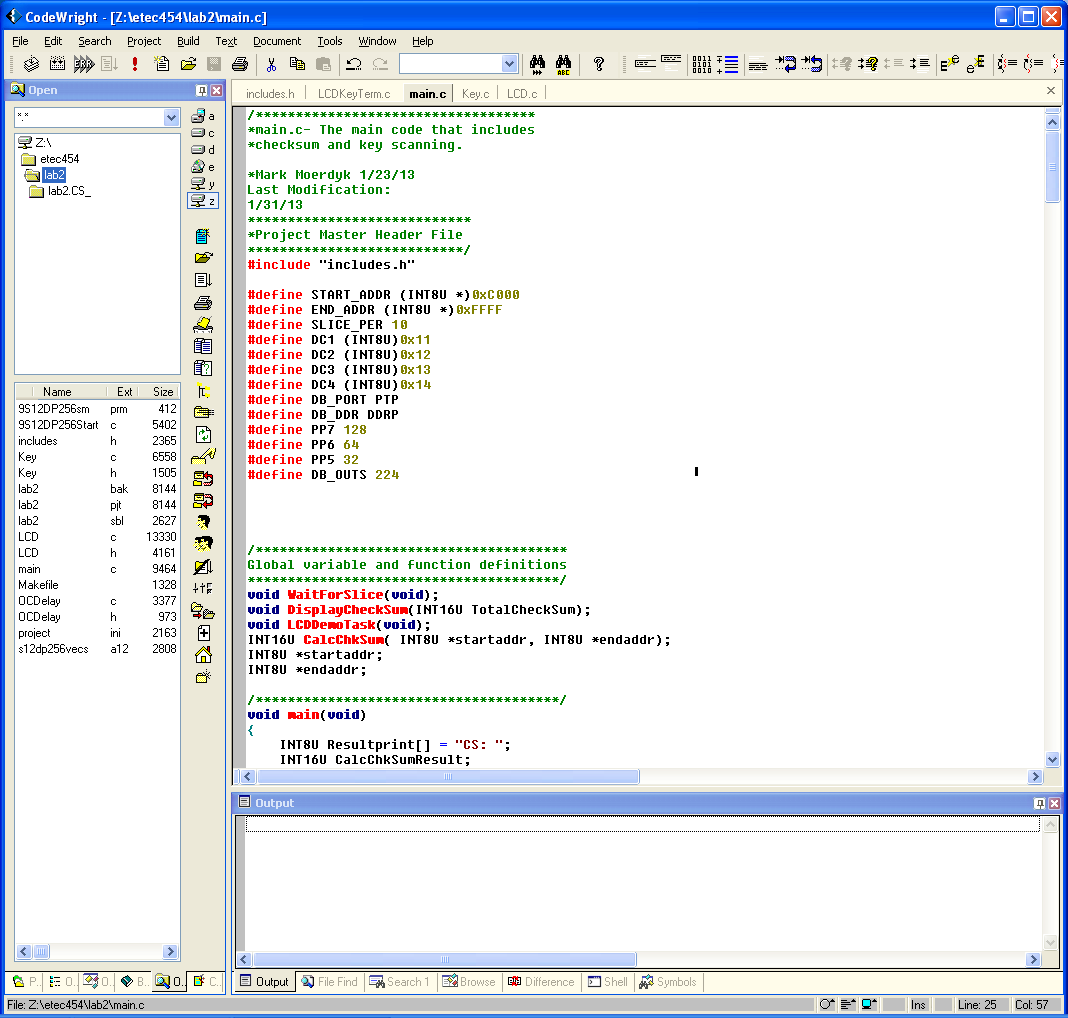
|  |
| --- |
|  |

**Introduction:**

The goal for lab 2 was to start writing our own code that performed a checksum routine, as well and display an LCD demonstration in C. The code was put onto the 9S12DP, where it had a keypad and LCD attached for the code to work. We were given the startup files, but we had to make our own main.c file in order to demonstrate the tasks needed.

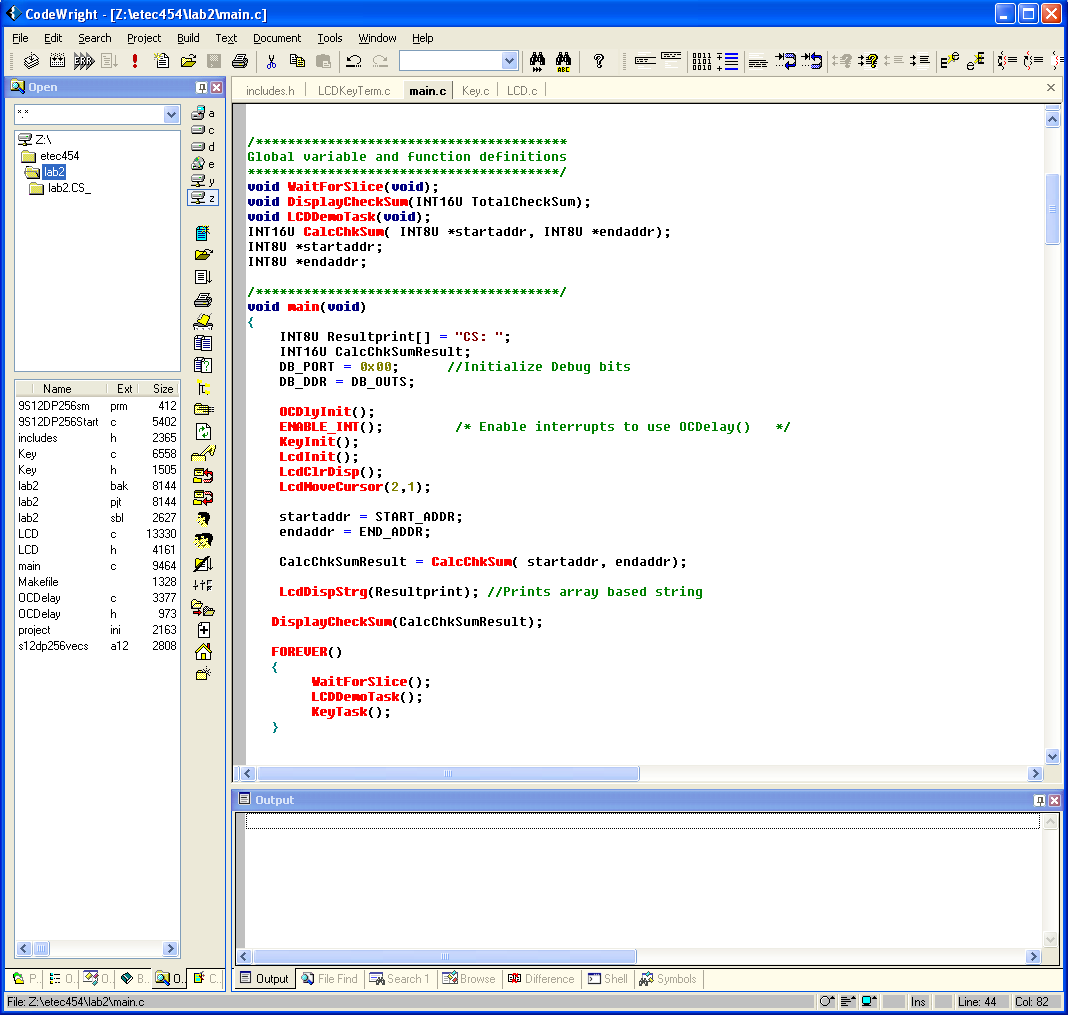
**Program Description:**

The first thing that I did was make a new project folder and project for all of the files to go into. I then inserted the given header file, linker command file, startup file, and library files needed for this lab. I then created a new main.c within the folder to write the main portion of the code. Below the description of the file and my name, I included the includes.h file and defined variables that were needed for certain functions and tasks, as you can see in figure 1.



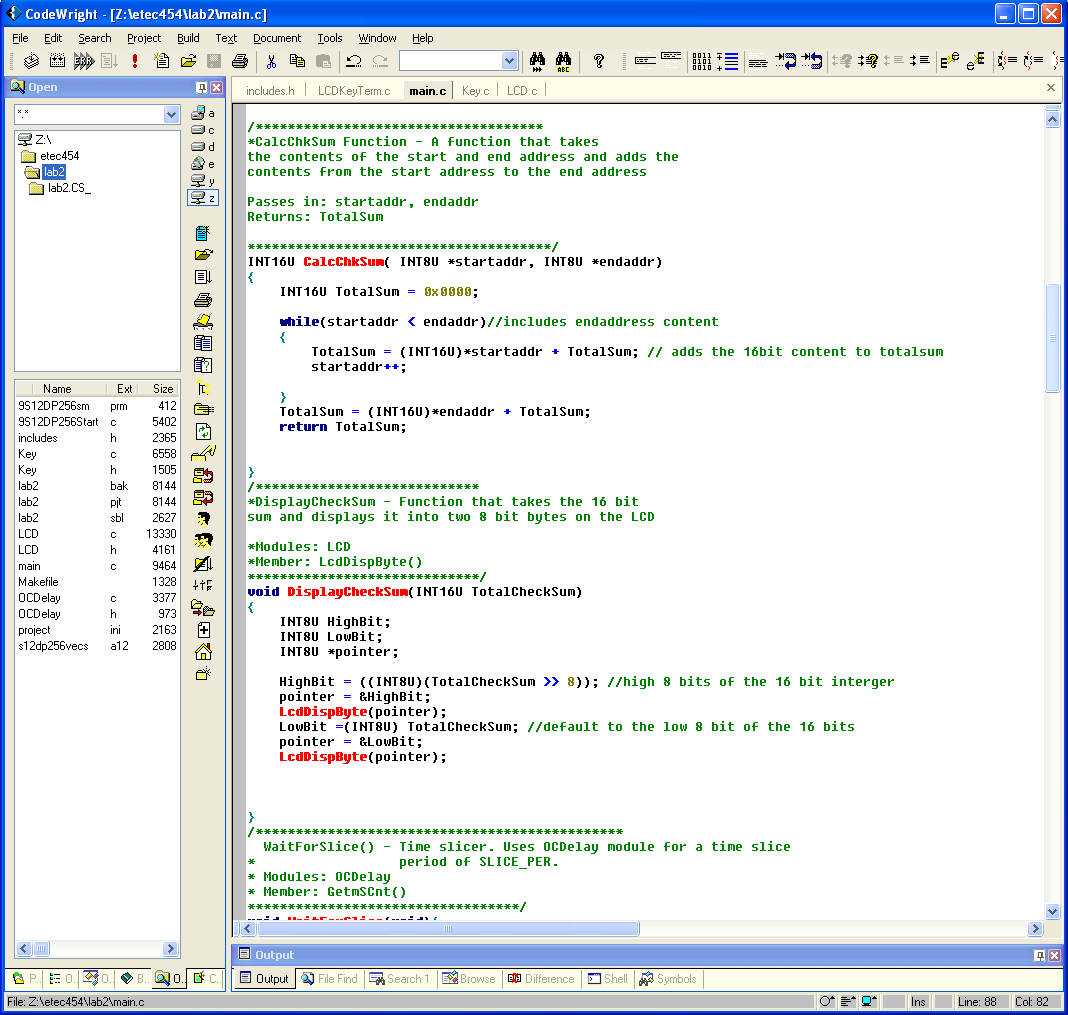
Figure

From Figure 1, I defined the START\_ADDR and END\_ADDR for the checksum routine, which is later in the code. The SLICE\_PER is used in the WaitForSlice function, and helps set up the wait for the next slice. DC1 to DC4 refers to the a,b,c, and d buttons that are used in the LCDDemoTask routine. The other four are ports for each of the three looping tasks, and set up the ports to see when each of the tasks are used, and not used.



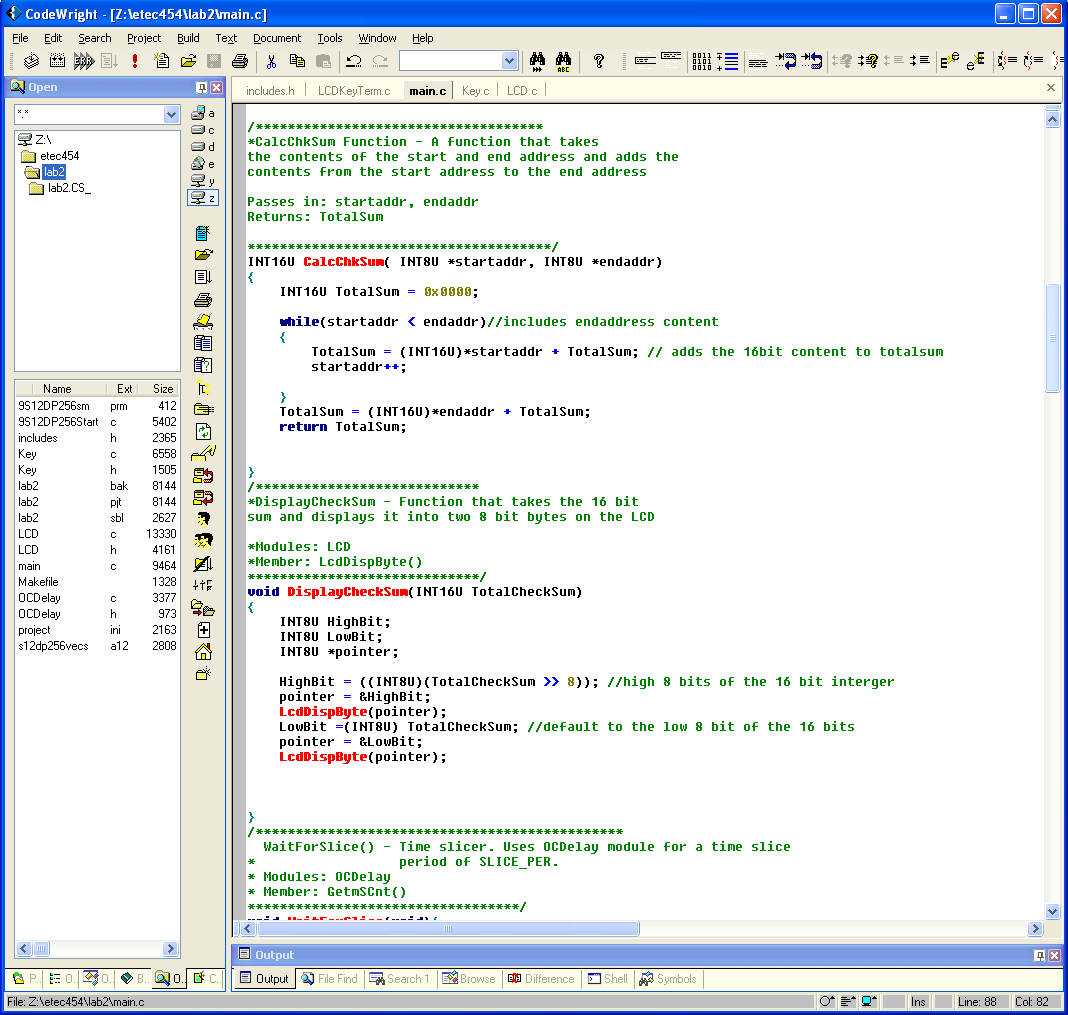
Figure

Shown in Figure 2 above, is the declaration of the global variables and functions, along with the main function. There are four different functions that were written for this lab, that include WaitForSlice, DisplayCheckSum, LCDDemoTask, and CalcChkSum. There are also two global variables that are pointers called startaddr and endaddr. Within the main function, you can see that both of the global variables are set to equal the values of the START\_ADDR and END\_ADDR that were defined back in the code on Figure 1.The first thing that the main function does is initialize the board, which includes the keypad, LCD screen, and the OCDealy for the counter. This allows all the other tasks to work on the board. Also in the main code we have two variables, Resultprint and CalcChkSumResult. The ResultPrint is an 8 bit unsigned integer, which is used to display a string with the given function LCDDispStrg. The CalcChkSumResult is used to store the value of the check sum result, and display it on the LCD. It has to be a 16 bit integer because of the space required by the addition of the checksum routine. The main also has a forever loop, in which it runs the three other functions: WaitForSlice, LCDDemoTask, and KeyTask. WaitForSlice and LCDDemoTask are written later in the code, while the KeyTask is referenced on Key.C



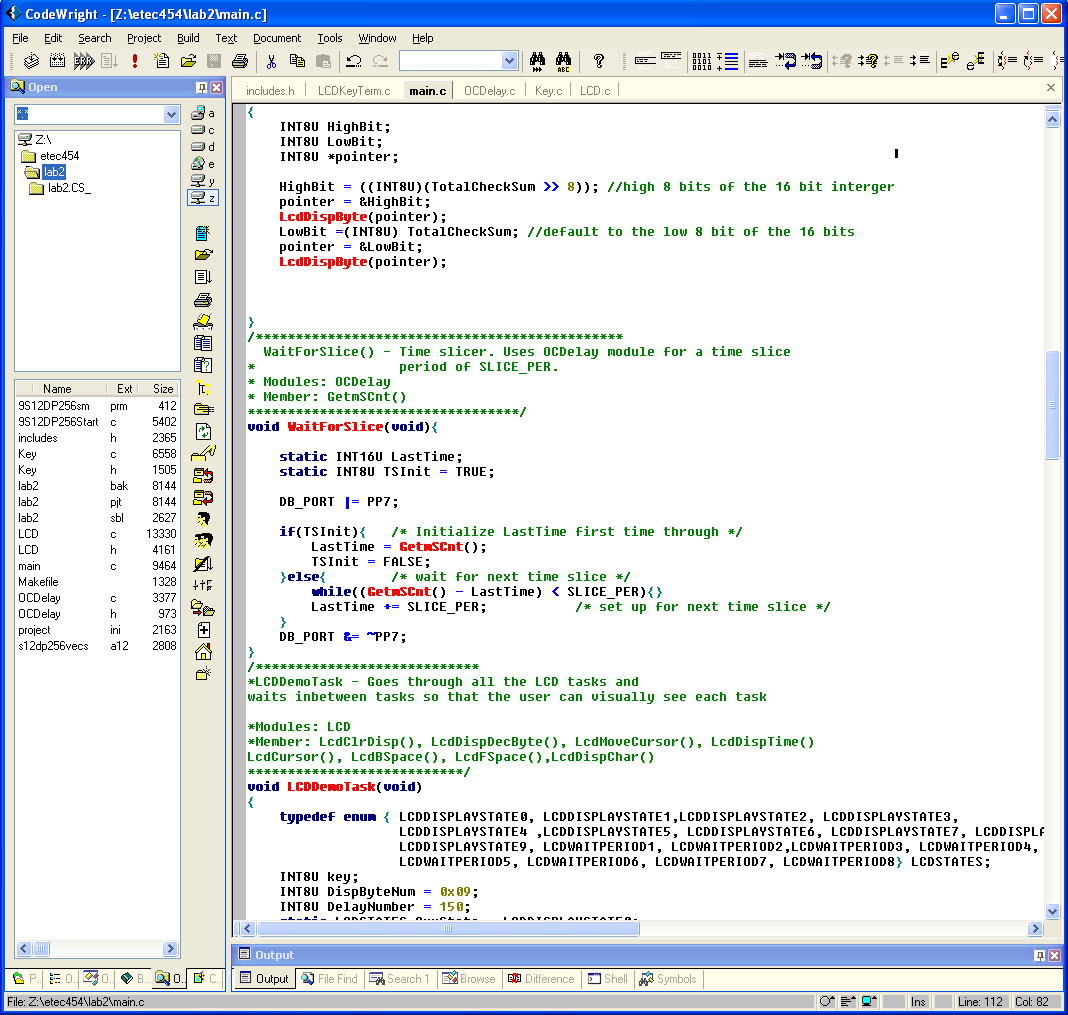
Figure

The first routine that was written was the CalcChkSum routine, shown in Figure 3. This routine takes in two 8 bit pointers, and returns a 16 bit integer. I first initialize a 16 bit integer where the sum the start address to end address will be stored. Then there is a while loop, which goes from the start address to the end address and adds the contents of each of those addresses, and stores them in TotalSum. In order for the addition to take place, you have to define startaddr as a 16 bit integer, otherwise you will get an error because you can’t add two different size integers together. After the while loop, you add the contents of the end address because the while loop did not add the contents of the endaddr within the loop. Then it returns the TotalSum to the variable that we defined back in Figure 2, CalcChkSumResult.



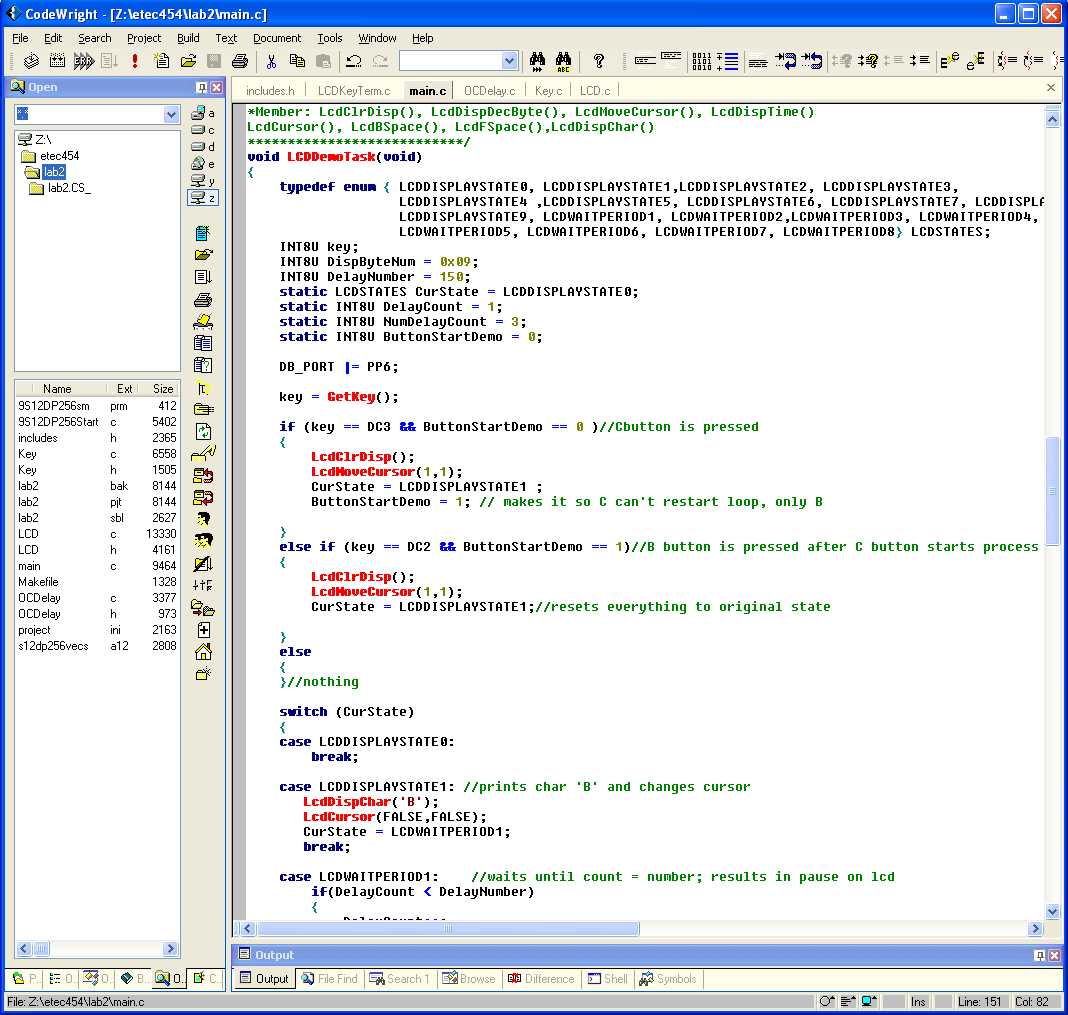
Figure

Shown in Figure 4 is the next function DisplayCheckSum. It takes the 16 bit integer of the TotalCheckSum, which I got from the CalcChkSum routine in Figure 3, and displays it onto the LCD. Since the LCD only has tasks that can display 8 bits, the function breaks down the 16 bit into two 8 bit integers, so they can be displayed on the LCD. When you take the 8 bits from a 16 bit integer, it defaults to taking the low 8 bits of the 16 bit integer. So in order for Highbit to have the high 8 bits of TotalCheckSum, I used the >>8, which tells the code to shift 8 bits. Then the code displays the upper and lower bytes onto the LCD screen.



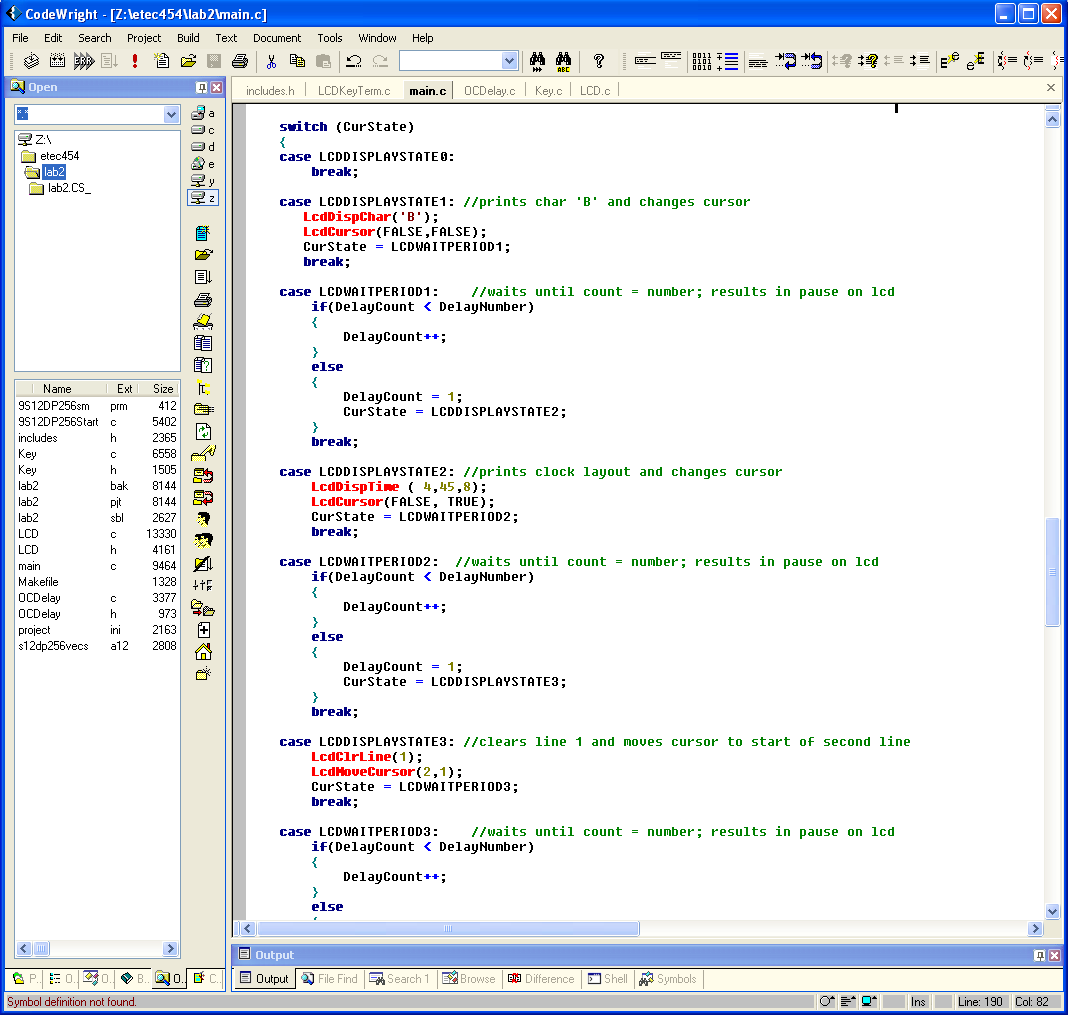
Figure

Figure 5 is the function WaitForSlice. The main purpose of this function is to make sure that each cycle of the task loop is done every 10ms. It calls the function GetmSCnut, which is the OCDelay.c file, and checks if the ms count is less than 10ms. If it is less than 10ms, then it stays in that loop. When it does reach 10ms, it starts over with a new 10ms clock cycle.



Figure

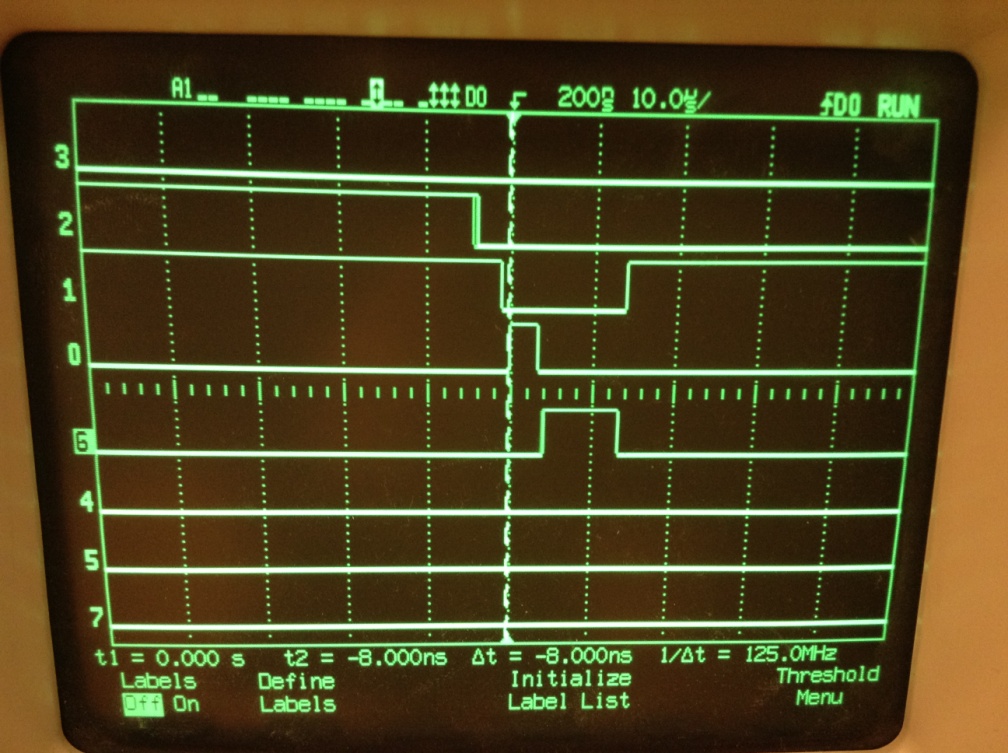
The main function that was written is shown on Figure 6, LCDDemoTask. The first thing that I did was define an enumerator type for the state machine that is later in the code. I had to have some static variables in this function because the function jumps in and out every 10ms. Having the static variables allows me to change their values, without them changing back when they reenter the function. The variable key gets the input from the key pad function GetKey(), which is stored in the Key.c file. It takes the value of the key press, and compares it to the two conditions that are given. If the key press does not equal any of the two keys defined, then it goes straight to the state machine. The variable ButtonStartDemo acts as a switch. It makes sure that when the C button is pressed and only the C button does the State of the state machine change. Then when the C button is pressed, it changes the value of ButtonStartDemo to 1 so that only the B button can be pressed to reset the state machine. The state machine itself has 17 different cases which are all defined by the enumerator type. The state is initially set to LCDDISPLAYSTATE0 so that it does not run anything until the C button is pressed. When the C button is pressed, it changes the State to LCDDISPLAYSTATE1. When the B button is pressed, it clears the LCD screen, and sets the state from its current state, back to LCDDISPLAYSTATE1.



Figure

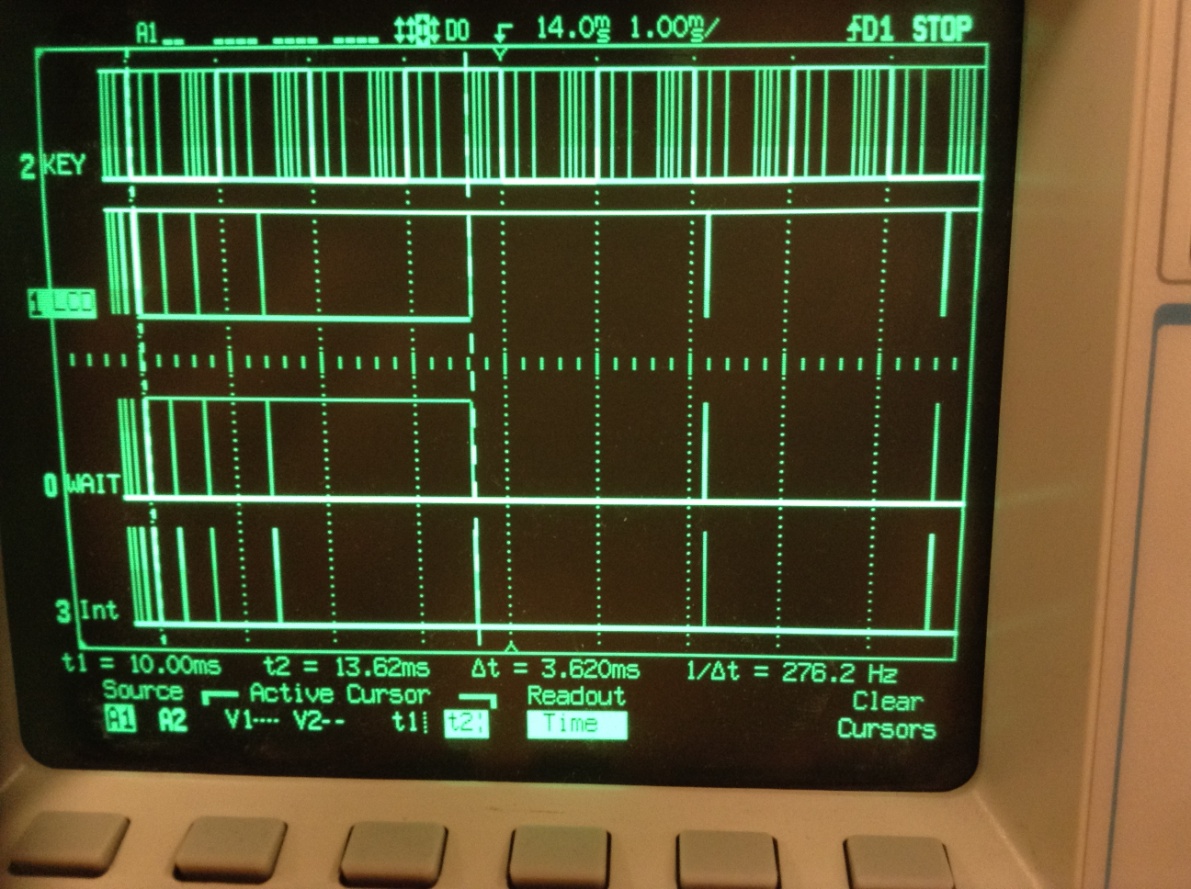
Figure 7 shows a part of the state machine that runs through all the different states of the LCD display demo. As you can see after each case, there is a break which jumps down to the bottom of the code, and jumps out of the function. In between each of the display states, there is a wait period state. This allows the user to see each of the states clearly. Without the paused state, the user would see all of the tasks with one second, and would not be able to differentiate between any of the tasks. After each of the case statements, they change the value of the CurState, so that the next state can run. The very last case statement has the next state return to the very first state, which makes the LCD demo a never ending loop.

The way that we had to check that all of our functions were within a 10ms clock cycle, was to use the port pins that we defined in Figure 1, and use them to set each function on and off during the beginning and end of each function. An example of this is shown in Figure 5. We set the bit at the beginning of the function, and turn it off when it reaches the end of the function. This creates something like Figure 8 below.



Figure

Figure 8 shows what the average cycle looks like when we graph the forever loop functions. Line 2 is the clock cycle. Line 1 is the WaitForSlice function. Line 0 is the LCDDemoTask. Line 6 is the KeyTask() function. When the clock goes to the falling edge and starts a new cycle, it makes the WaitForSlice go low. This then allows the LCDDemoTask to run. In Figure 8, it is running one of the LCDWAITPERIOD states, which is why this is so small. Then when the LCDDemoTask is done, it goes low and the KeyTask can start. When the KeyTask is done and goes low, then the WaitForSlice goes high again, and stays that way until the 10ms count is done. Capturing all of the tasks being done to see which state takes the most time is shown in Figure 9. The greatest task measurement is 3.62ms, which is less than 10ms, so that means that our code will not throw the clock off and create problems.



Figure

**Comments and Conclusion:**

This lab took a lot of time in figuring out how things were done. This is probably due to the fact that I have not done a lot of coding since 374. I did like that we had references in Lab 1, as well as the .pdf files to look at if I needed help on how to program something. The biggest issue that I had with this lab understood what was required in the coding of the lab. The lab document seemed at least for me, hard to read and figure out what was needed to be done in the lab. I know that this will be a difficult class that will need lots of extra time in order to succeed, and I have one of these labs under my belt. This lab however, was a good introduction into how these 9S12 boards work, and teaching us the basics of the boards LCD monitor and keypad.