Course Title: CEE-345 Microprocessor System Design

Project Title: Countdown Timer and Stopwatch

Student Names: Jacob Hillebrand and Alison Zimmerman

Instructor Name: Dr. Cheng Liu

University of Wisconsin-Stout

Menomonie, WI 54751

May 09, 2019

Table of Contents

[1.0. PROJECT DESCRIPTION 3](#_Toc7964912)

[1.1. Introduction 3](#_Toc7964913)

[2.0. BILL OF MATERIALS 3](#_Toc7964914)

[3.0. DEIGN PROCESS 4](#_Toc7964915)

[3.1. HARDWARE DEVELOPMENT 4](#_Toc7964916)

[3.2. WIRING DIAGRAM 4](#_Toc7964917)

[3.3. SOFTWARE DEVELOPMENT 5](#_Toc7964918)

[4.0. PROJECT ILLUSTRATION 13](#_Toc7964919)

[4.1 CONTROL FLOW SCHEMATIC 14](#_Toc7964920)

[5.0. STUBLING BLOCKS AND CONCLUSION 14](#_Toc7964921)

# PROJECT DESCRIPTION

This project utilized a KL25Z Freedom Development Board, a Digilent Keypad, and a 2x16 LCD to realize a design for a combination timer and stopwatch device. The keypad can be used to not only control start, stop, pause, resume, and quit functionalities for both the timer and stopwatch, but can also be used to set precise times for the timer. The time limit for both the timer and the stopwatch is 60 hours.

## 1.1. Introduction

For this project we knew we were going to need to implement both a keypad and a display in order to make a functioning timer and stopwatch. This meant we would need to find a way to connect both the keypad and LCD display to the Freedomboard at once and figure out how to get both of them to work concurrently. The keypad needed 10 pins (4 rows, 4 columns, 2 GND) and the LCD also needed 10 pins (4 data, 3 R,W, and R/W Select, 1 3.3V, 1 GND, and 1 Brightness selection pin). Additionally, we were going to need a Potentiometer to control the brightness of the LCD.

Luckily, we had done labs for Microprocessor System Design that covered both of these components. Even more lucky for us was the fact that the labs had been designed such that the pin configurations on the Freedomboard for each component would not interfere with one another. This meant that we were successfully able to wire both components to the Freedomboard and use the keypad to change LCD output before we had even begun to design our project. As such, we were able to set about achieving our goal of creating a stopwatch/timer with relative ease.

# BILL OF MATERIALS

|  |  |
| --- | --- |
| Part Name | Cost |
| 2x16 LCD screen | $6 |
| Diligent Keypad | $10 |
| KL25z Freedom Board | $31.11 |
| Potentiometer | $1 |
| Total | $48.11 |

# 3.0. DEIGN PROCESS

## 3.1. HARDWARE DEVELOPMENT

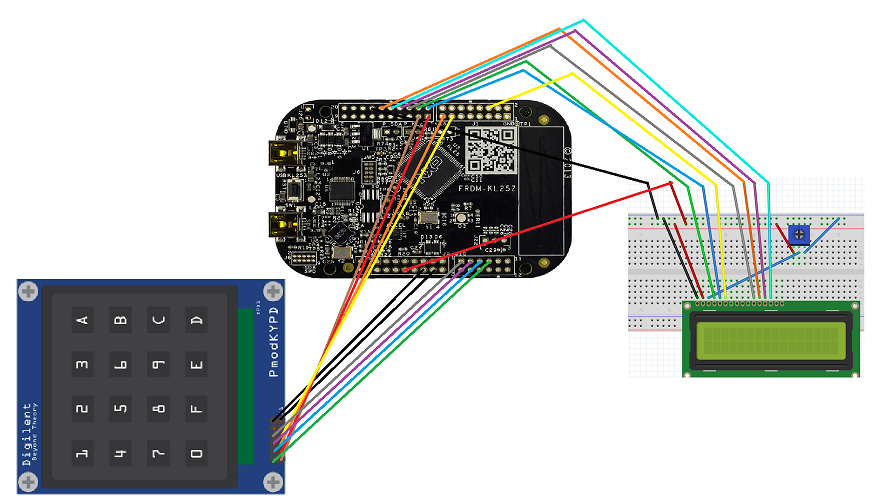
**Keypad Input**

As mentioned above, keypad input was a critical part of this project. Based on work done in previous labs, we were able to successfully wire up the Digilent Keypad to the Freedomboard with the pin configuration shown below. Additionally, we were successfully able to read in user input by utilizing functions we had previously written to scan the keyboard in a row-by-column fashion to determine which key had been pressed. This would then return a position that had been detected as being “on”, which was then converted to a character representation of the keypress. This character representation was then returned to the program and could be easily used to implement features.

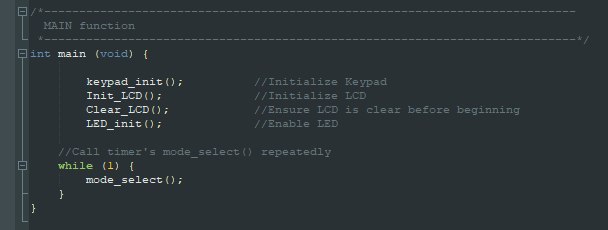
**LCD Output**

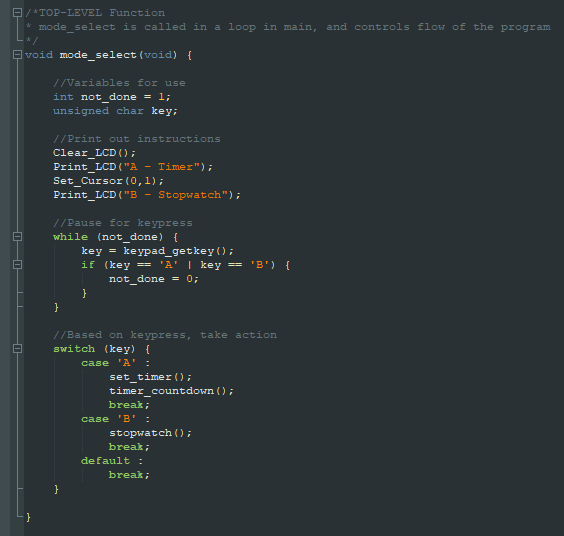
The next piece of hardware to implement was the LCD. Again, we had previously done labs utilizing this component, so we were easily able to wire it up as shown in the feature below. The particular LCD we had chosen to use was a 4-bit LCD, which meant that data could only be sent to the display 4 bits at a time. This data could be sent, then triggered with the register select pin, then converted in the LCD registers and printed to the display in character form. Every time a new set of characters was to be displayed, the display had to be cleared, and then characters could be reprinted. Once this component was successfully implemented, we could begin software development of the timer and stopwatch.

## 3.2. WIRING DIAGRAM

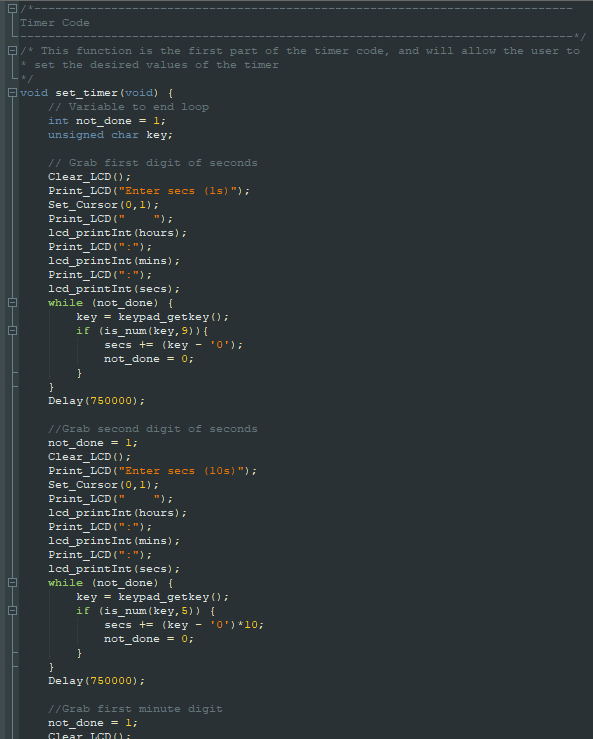


## 3.3. SOFTWARE DEVELOPMENT

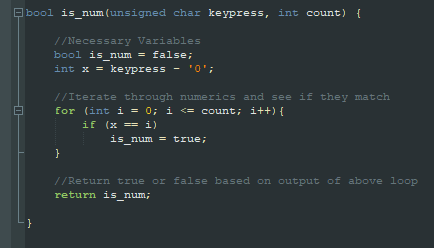
 With this project, we wanted to take a bit of a top-down approach to the design. Essentially, we wanted to make sure that the functionality of our program was modular and that sections of it could easily be added and removed when necessary. As such, we began by making sure we had a top-level control module that could be repeatedly run in main, which would control all aspects of the project. Our main and top-level functions are shown below.



In the top-level function, it is clear to see how the design operates. Information is printed to the LCD regarding how to select either the timer or the stopwatch. Then, a function from the Lab 7 keypad lab is employed in a “while” loop to wait for a keypress. When the proper keypress is detected, the function then moves to the case-switch at the bottom of the function. Based on the key that was pressed, the case-switch calls either the functions for the timer (set\_timer() and timer\_countdown()) or the function for the stopwatch. When those functions finish running (ie. the timer or stopwatch finish), the mode\_select function ends, and is re-called in the main function. This process is repeated indefinitely to give the project operation.



**Timer**

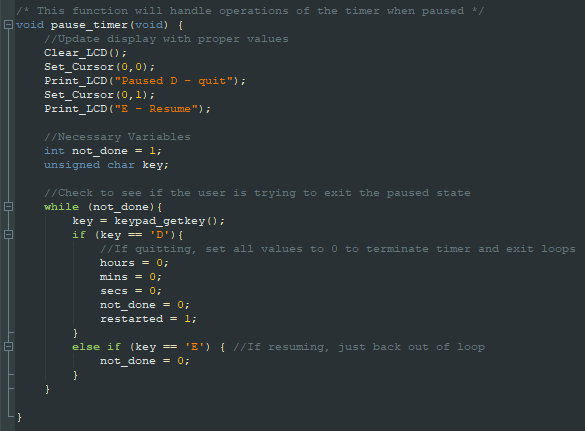
In the event the user selects the timer, the top-level module calls the set\_timer() function shown above. This function’s task is to take user input and allow the user to set a time for the timer. As shown in the code snippet above, the function goes digit by digit taking user input. It clears the display, prints to the display with the digit it is currently requesting, waits for the digit by again employing the keypad\_getkey() function in a while loop. Upon detection of a keypress, the keypress is passed off to a function called is\_num() to determine if the keypress was an acceptable time input. If it is, the time is assigned to the proper time global variable (hour, mins, secs) and the function moves to the next time. It is important to note that when the program requests a time for the 10s place of any value, the value is multiplied by 10 before being added to the timer in order to properly record the time. Once this function has requested and received time for all values, it exits and returns to the top-level module so the program can continue.

As a side note, is\_num() is a function that we had to create particularly for this project. This function takes in 2 parameters – the keypress detected, and the top of a numerical range that is desired. Essentially, this can be used to take in a keypress and determine if the keypress is in a 0-9 range (numerical) or a 0-5 range (time-based, as you cannot exceed 60 seconds, 60 minutes, etc.) The function iterates through the desired numbers, and checks if the keypress is within range. If not, it returns false, if so, it returns true. Though straightforward, this is one of the most useful functions for our project.

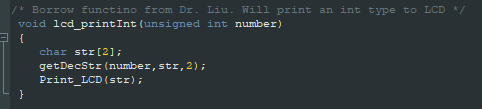
Going back to the timer module, we can see that after set\_timer() has concluded, it returns to the top-level module, where the timer\_countdown() function is called. This function is the primary function of the timer, as it is responsible for managing the decrementing of the timer as well as watching for pause operations from the user. As shown below, the module first checks for a pause operation from the user, then clears the display and prints the new time values to the screen, has a cascading set of if statements to manage the time in increments of 60, and delays by 2550000 to simulate a 1 second delay between operations. This loops over and over until timer reaches 0 or the user quits.

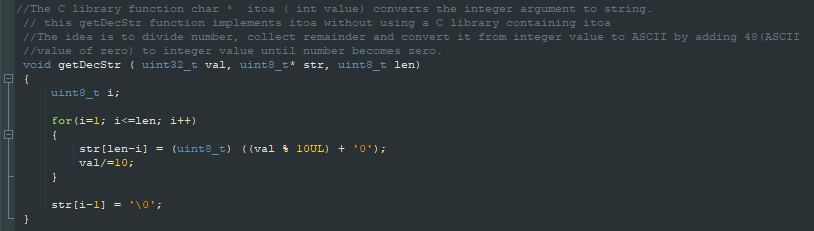


There are a couple of interesting things to note within this function. For example, if it is detected that the user is pressing the “F” key to pause the timer, the function pause\_timer() is called. This function, as the name suggests, manages the program while the timer is paused. As shown below, the function begins by clearing the display and printing with instructions for resuming or quitting the timer. The function then enters another getkey()+while loop code combo, and waits for user input to see what the user would like to do. If the user pressed “D” to quit, the values for all times are set to 0, and the loops are quit to return to the timer\_countdown() function (where the function will quit the timer altogether as it sees that time is 0). If the user presses “E” to resume, the loop is simply broken and the function returns to timer\_countdown() where the countdown continues as normal.

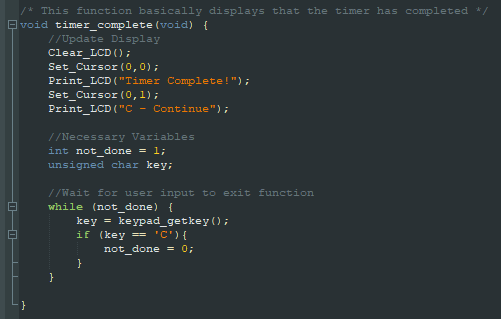


Another interesting thing to note with timer\_countdown() is the function used to print the time to the display, lcd\_printInt(). This function is not exclusively used within timer\_countdown(), and is very critical to the functionality of the LCD. With the standard Print\_LCD() function, a string to be printed must be passed. However, with lcd\_printInt(), a number of type int can be passed, and the function will print it to the display. It does this by passing the Int to a custom function, dec2Str(), which takes the int and uses special bitwise divisions and shifts to convert the int to a string. This string can then be printed to the display normally. This particular function was very difficult to implement, and we ultimately were only able to do so with help from Dr. Liu. Both the lcd\_printInt() and dec2Str() are shown below.



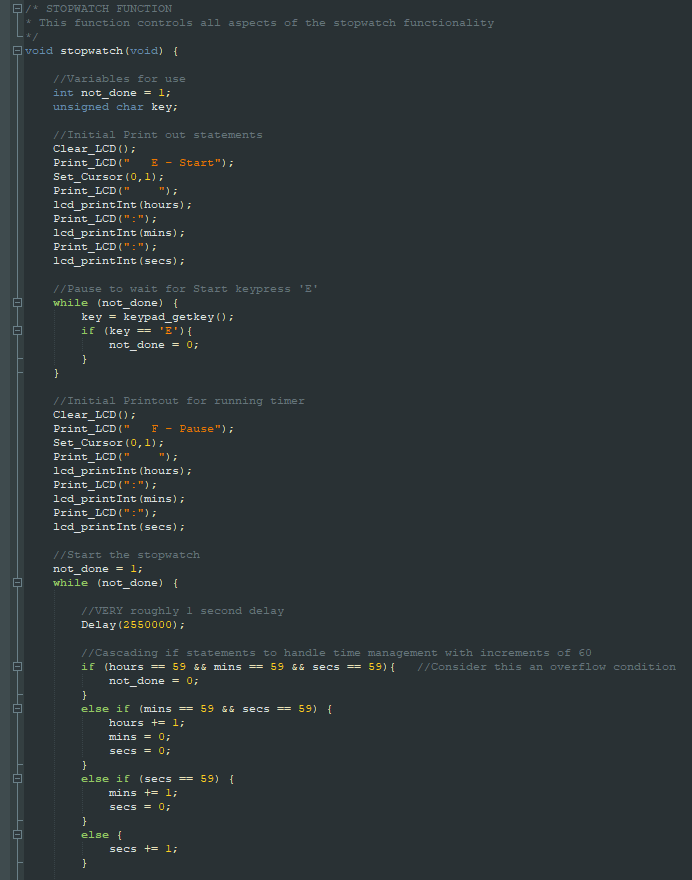


One final interesting part of the timer\_countdown() function is the timer\_complete() function. The conditional in timer\_countdown() ensures that the only way this function is called is if the timer has organically reached 0, not been set to 0 via a pause+quit operation. When called this function simply clears the display, prints information that the timer has completed, then utilizes a keypad\_getkey()+while loop to wait for the user to press “C” to continue. When the key is pressed, the function exits, the timer\_countdown() can exit, and the timer finishes, returning to the top-level module so the user can select which mode to use.

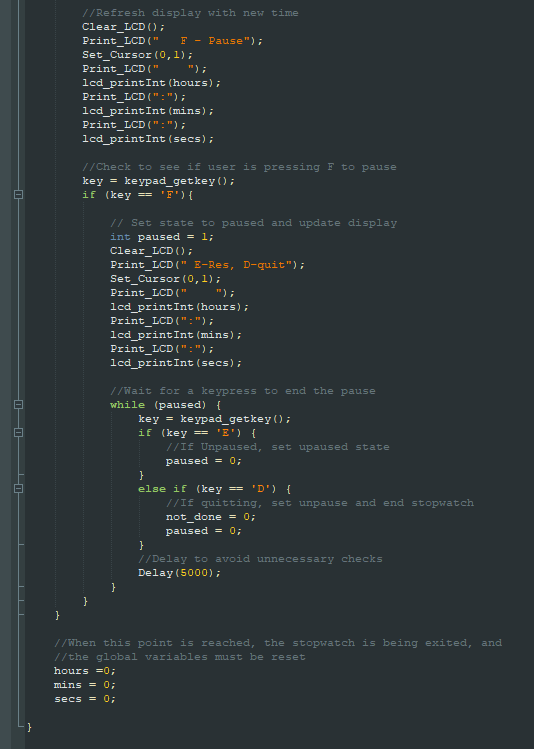


**Stopwatch**

In the event the user selects the stopwatch, the top-level function calls the function stopwatch(). While this operation utilizes many of the same functionalities as the timer does, this operation is much simpler, and is contained mainly within the single stopwatch() function.



As shown on the previous page, the function begins by clearing the display, and printing instructions to start the timer. After gathering input to continue with a getkey()+while loop, the function continues by updating the display with instructions to pause and the current time. At this point, the function enters the main while loop. First, a pseudo-1 second display is run with Delay(2550000), then a cascading set of conditionals is run through to ensure that time will increment in sets of 60, rather than sets of 10 like normal counting.



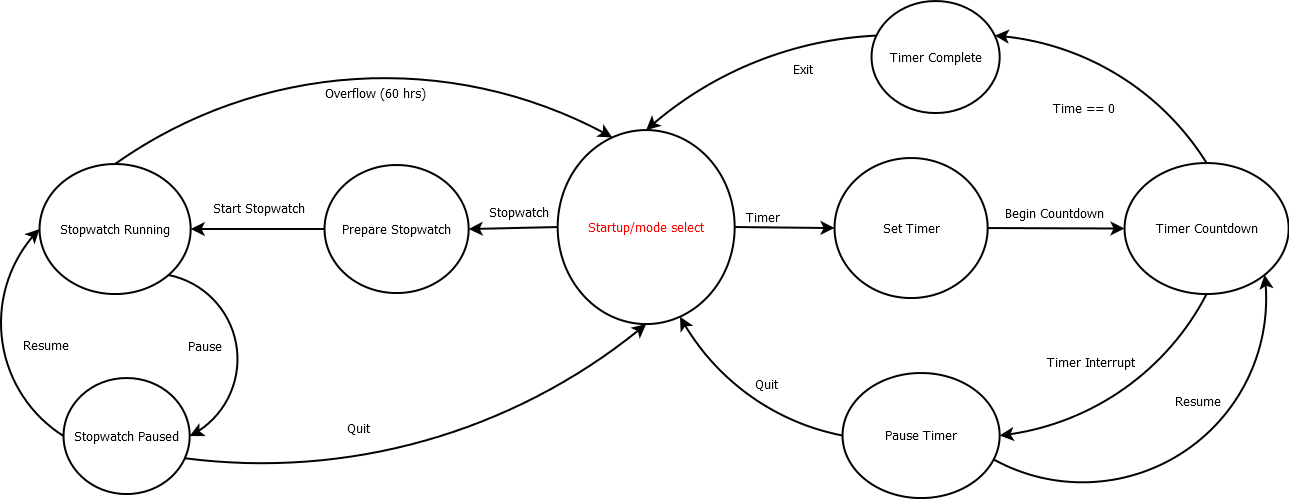
Then, switching to the second code snippet, the display gets updated with the new time, and another getkey()+while loop is used to check if the user is trying to pause. If not, the loop simply restarts here. However, if the user is trying to pause, the display is cleared, refreshed with instructions on how to resume and quit, and another getkey()+while loop is employed to determine whether the user would like to resume or quit. If the user would like to resume, the function simply exits this subloop, and returns to the above loop to continue counting up. However, if the user would like to quit, values are set to signal both the subloop and the above loop to stop (this would only happen organically if the stopwatch hit 60 hours, a decision made on overflow conditions by the designers). When this happens, all time values are reset to 0, and the stopwatch() function returns to the top-module for the user to re-select an operation.

# A close up of a device Description automatically generated4.0. PROJECT ILLUSTRATION

A close up of a device

Description automatically generated

## 4.1 CONTROL FLOW SCHEMATIC



# 5.0. STUBLING BLOCKS AND CONCLUSION

In summary, this project was a success. Not only were we able to successfully interface with both a keypad and an LCD with our KL25Z Freedomboard, but we were also able to create a supporting program that allowed utilization of the system as a stopwatch and timer. In designing our software, we were also able to ensure that our code was modular, and most operations are kept in separate functions for ease of use. As a result, all these functions can easily be modified, added, and removed with relatively low difficulty. All of our code is also on Github and will be preserved for future use.

However, the project was not without difficulty or challenge. Our main challenge was trying to implement a function that converted an int type to a string, as the functions we had utilized in lab only printed strings. After trying to research if the compiler supported a default integer-to-string function (which it did not) we had to turn to possible substitutes and research how to build our own function. After much failure on our own, Dr. Liu’s helped us find a working function to convert the int that was to be displayed to a string, which could then be easily printed on the LCD screen with our existing functions.

One other major challenge with this project was version control management. When making this project we used Github and Gitkraken for version control. One member was new to Gitkraken, and the other member was new to both, so the process required much research on how to use both pieces of software. Several times we ran into issues where one person had pushed changes, and the other person was not up to date, which resulted in merge conflicts. This required research into stashing and branching, and how to resolve merge conflicts. Eventually, though, we figured out the process, and were successfully able to utilize the software to allow for seamless version control. In overcoming this, and other difficulties, we were successfully able to complete our project and realize our vision for a stopwatch and timer design.