Final Project – Math Alarm Clock

CMPE 3815 – Microcontroller Systems

Erik Simkins

Partner: Danny Burger

1. Introduction

This entire project can be found on GitHub at the following link: https://github.com/dburger47/Math-Alarm/tree/main

For the final project, the math alarm clock idea was revised from Lab 7. This time instead of using the Arduino and millis() to keep track of the time, a real time clock module was used. The RTC made this project so much easier as it communicated over I2C, and had a coin battery to save time even without power. Besides the RTC, an LCD is used to display the time, and the keypad was implemented for input rather than the serial monitor like in Lab 7. By pressing "A" on the keypad, the alarm time can be changed. The "B" button is responsible for toggling on and off the alarm, and to reset the current time the "D" button allows the user to do so. When the alarm goes off, the user is prompted with two math problems and one solution, the correct answer needs to be selected to turn off the alarm, an example of this can be seen below in Figure 1.

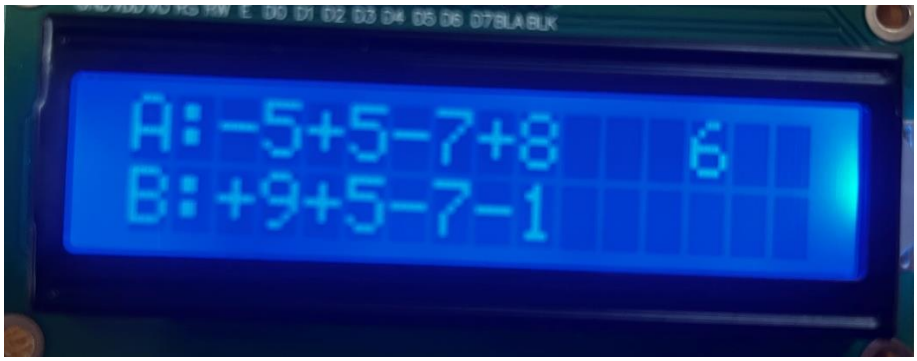

Figure 1 – Math Problems on LCD

2. Results and Analysis

The first step to getting the alarm clock to work was to get it to function like a clock. The get_time() function refreshes the RTC to get the current time, the time is properly formatted and printed to the LCD as well as to a string that will be used to determine if the alarm needs to go off. The hardest part of this function was making the time format to hh:mm:ss properly, with some help from Chat GPT this was obtainable and the time was displayed to the LCD.

To allow user input from the keypad the same function keypadListen() from Lab 2 was used. The keypad matrix had to be rotated 90 degrees and mirrored horizontally as the pins on the new keypad were arranged differently from the

keypad in lab 2. The function sets all the columns high and scans through each, one by one setting them low looking for the corresponding button press to map the key based on the row. The debounce function from Lab 2 was also used to verify that the correct button was pressed and allowed bouncing to pass by rereading the pin again after a 10 ms delay.

The next step was to allow the user to reset the time. The RTC library makes this so easy. RTC.set() can be given: seconds, minutes, hours, day of the week, day of the month, months, and years. For this project only seconds, minutes, and hours were used. When the button “D” is pressed on the keypad the set_time() function is called and prompts the user to enter the hours, and then the minutes. These values are passed to rtc.set() after some error handling to verify that the user only pressed the number keys, enters hours between 00-23, and minutes between 00-59. If the user enters an improper time the function is aborted, and the time remains what it was before. Figure 2 below shows what is prompted to the user when setting the current time.

Figure 2 – Setting the Current Time

The same approach was used to set the time of the alarm. The user is prompted for hours and minutes, error handling verifies the input, and then instead of setting the time, a string is formatted that will be compared to the current time string. Figure 3 below shows what is prompted to the user when setting the alarm time.

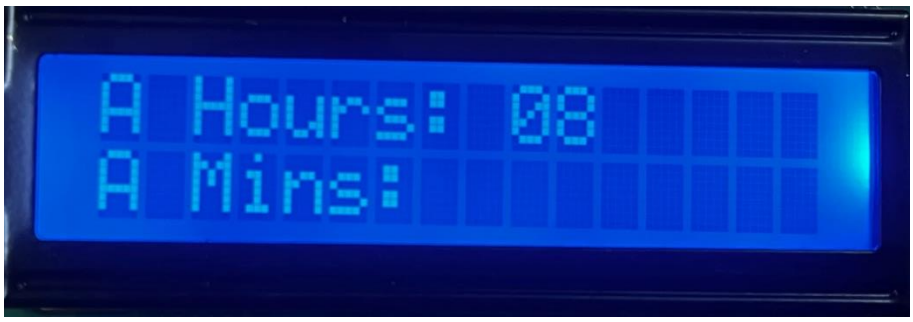

Figure 3 – Setting the Alarm Time

The last step of getting the project to function as a clock was to create a function that toggles on and off the alarm. This was as simple as alarmEN != alarmEN each time the "B" Button is pressed. The disp_alarm() function properly formats the alarm time to be shown on the LCD as well as if the alarm is on or off based on the alarmEN state. Figure 4 below shows the home screen of the clock displaying the time, alarm time, and alarm state.

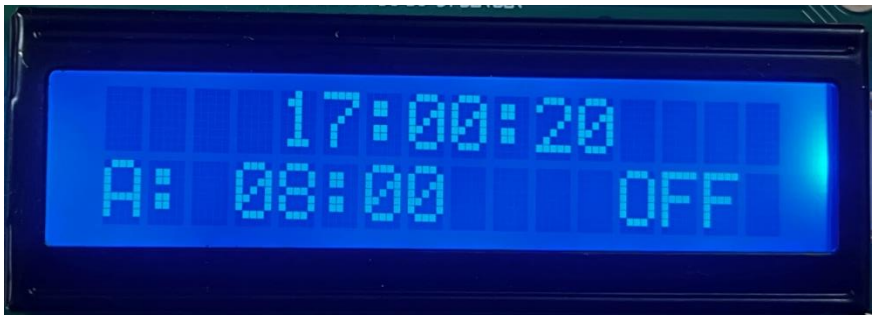

Figure 4 – LCD Display

As mentioned, the strings of the current time and alarm time are returned and used to set off the alarm and enter the math problem's function. While the current time equals the alarm time, the alarm state is on, and the math problems haven't been answered yet, the code plays the buzzer and enters the mathProblem() function.

The mathProblem() function works by first randomly generating a 4 number problem and solves for the correct answer. It then randomly changes the problem either in the operand or the numbers to create a second problem. Both of these problems are displayed to the LCD along with the answer as seen in Figure 1, the keypadListen() function is used to get the input from the user, either an "A" or "B". If the correct key is pressed the function is exited and the buzzer is turned off. If the incorrect key is pressed a different set of problems is created and the user must try again until correct.

Once the code functioned properly all the hardware was condensed and soldered to a proto board. Using SolidWorks a housing was created to give the alarm clock the feel of an actual product. The design for the housing was based off those old calculators where all the buttons are on the bottom and the screen is on the top and slanted at a 45-degree angle. Figure 5 below shows the completed housing, the width of the housing is dictated by the width of the Arduino and LCD which is roughly the same. A little extra room was needed in the back to ensure that the vertically placed RTC would fit. The housing is too parts that are snapped and taped together.

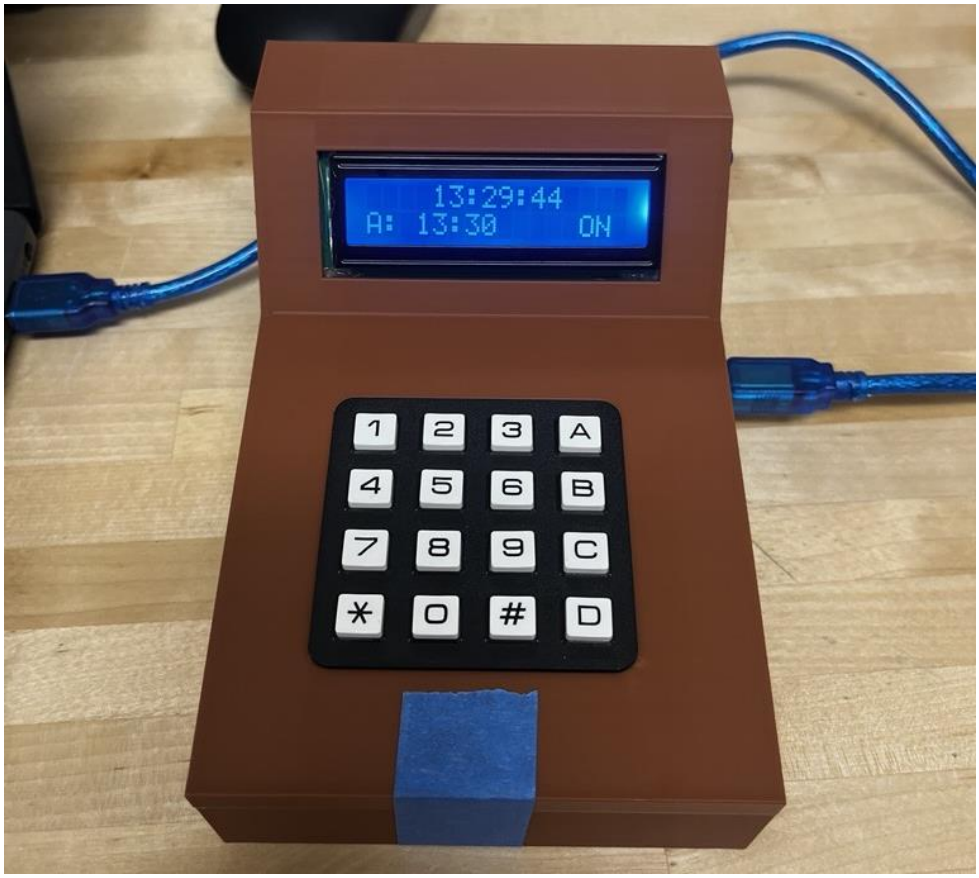

Figure 5 – Final Product

3. Conclusions

The math alarm final project was a success compared to the free project in lab 7. The real time clock module made keeping track of time way easier than using millis() and convert the time to hh:mm:ss. Because the RTC has a coin battery to save the time the time doesn't need to be set upon start up like it did in the earlier design. Using the keypad as an input compared to the serial monitor made the clock usable without the use of a computer. The math problems function in the first design required the serial to be set to NL and CR whereas setting the time required just NL. This required the user to change these settings, but thanks to the keypad all inputs worked through a common keypadListen() function. GitHub made managing this project even easier as Danny and I were able to work on the same code as the same time. Shout out to Dr. Barsic for a great Microcontrollers class, I had a lot of fun.

4. Appendix

Code: The code for this project can be found in the GitHub under "math-alarm-code-RevA.ino".