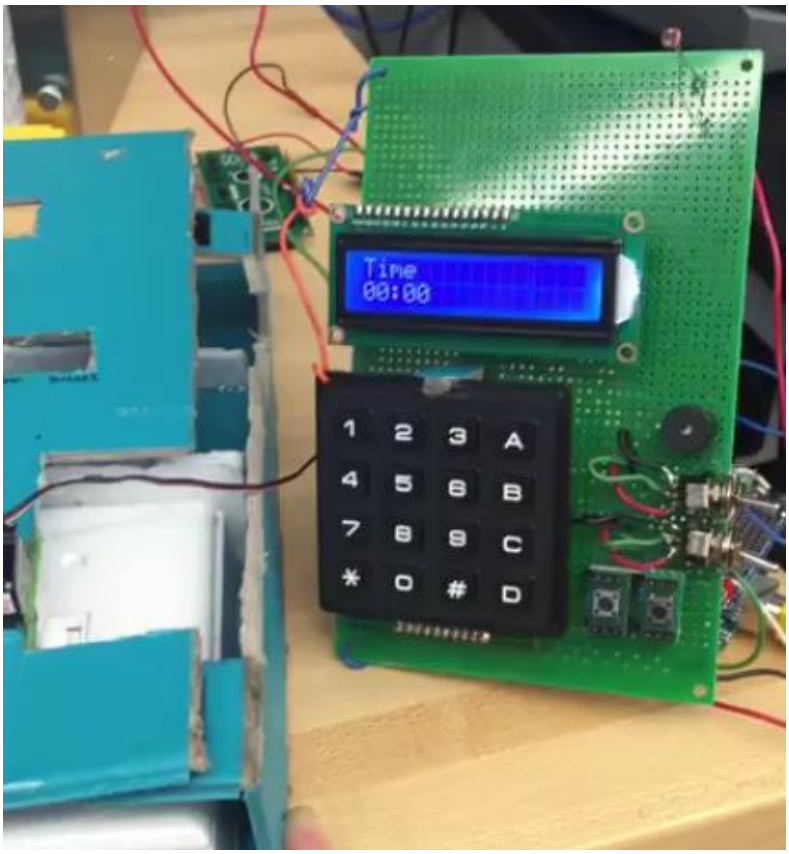
5-2016

Wallarm

Amanda Dinh
Trinity University, adinh1@trinity.edu

Kate Walls
Trinity University, kwalls@trinity.edu

Wallarm



Submitted by:
Group C
Amanda Dinh
Kate Walls

ENGR-4367
Mechatronics
Trinity University
May 2, 2016

Table of Contents

Design Summary…….3
System Details…….4
Design Evaluation……5
Partial Parts List……..5
Lessons Learned……...6
Appendix……...7-11

I. Design Summary

The nature of our project is essentially an upgraded alarm clock that ensures that the user will wake up at the time set. Figure 1 displays the hardware of the project and has colored circles to indicate different components. The alarm functions by initially sounding a loud buzzer (red) at the set alarm time. If the user decides to snooze the alarm by pressing the snooze button (purple), the alarm will wait for a given amount of time before setting off the buzzer again. It will then trigger a Servo motor (pink) that will flip a traditional toggle switch to turn the light on and then return to sounding the buzzer until the user turns off the alarm by pressing the off button (orange). This off button may be used at any time throughout the alarm.

The time is displayed on a liquid crystal display screen. The brightness of the screen is adjusted automatically using a photoresistor (yellow). The photoresistor allows the screen to be brighter in a well-lit area and dims the screen in darker settings in order to conserve power. The user can use the numeric keypad to manually input the time in military time. The purpose of using military time is to avoid the complexities of differentiating between AM and PM. To set the alarm time, a switch (blue) is used to change to "set alarm" mode where the user can use the same numeric pad and instruction set to set the alarm.

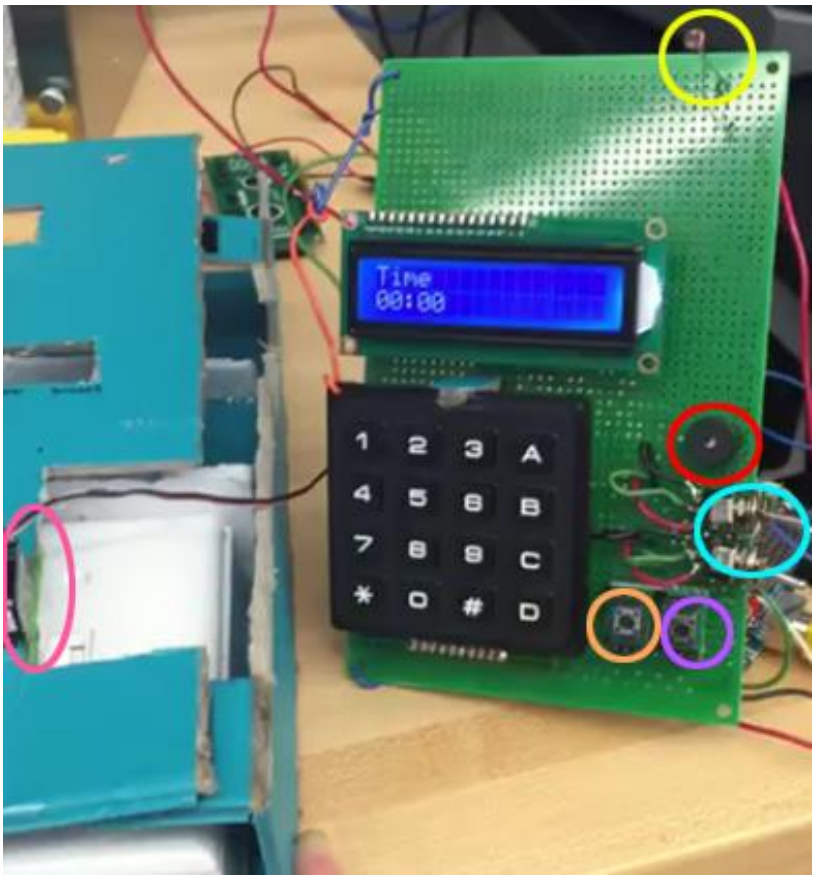


Figure 1. Diagram of the completed Wallarm project.

II. System Details

The clock counter is controlled by the Arduino and displayed on the LCD connected to it. It functions using a one second delay to increment the second constant within the Arduino. The counter is then set to increase the minute constant once the second constant exceeds 59 seconds and the minute constant similarly controls the hour constant. As these constants exceed 59 they are both reset to zero to count the next minute or hour respectively. This process repeats until the hour constant exceeds 23, at which point all constants are reset to zero. As this process loops, the minute and hour constants are displayed on the LCD screen. The photoresistor is connected to the brightness control pin of the LCD screen in order to manipulate the brightness of the screen with the surrounding light intensity.

The alarm can be set by flipping a toggle switch connected to the Arduino. If the switch input is high, the LCD displays ‘set alarm’ with the minute and hour constants automatically set to zero the first time until a user input is given from the keypad. While the switch is high, the keypad controls the alarm hour and minute constants, otherwise the keypad controls the clock hour and minute constants. The keypad control is condensed using a keypad decoder (74C922N) which outputs a unique four-bit binary code to the Arduino so that it can differentiate among all of the inputs. Once the clock and alarm hour and constants match, the Arduino outputs a digital 1 to the PIC. From here, the PIC controls the alarm system.

When the PIC reads the digital 1 from the Arduino, it begins the alarm loop which starts by setting the buzzer to high. The buzzer will continue to sound until either the snooze button is pressed or the off button is engaged. If the off button is pressed at any point during the alarm loop, the loop will end and reset to wait for the next trigger. If the snooze option is chosen instead, the buzzer will pause for 10 minutes before resounding. The buzzer will resound for only a few seconds as a warning before the PIC outputs a binary 1 to the Arduino. At this point, the motor is actuated by the Arduino and turns on the light switch. Figure 2 a) shows the motor before it has been actuated and is set to 0 degrees and b) after it has been turned on and set to 90 degrees. The setup of our program is shown in the flowchart in Figure 3 and 4 for the PIC and Arduino respectively. Likewise the interface circuits are given in Figures 5 and 6 and the wiring diagrams in 7 and 8.

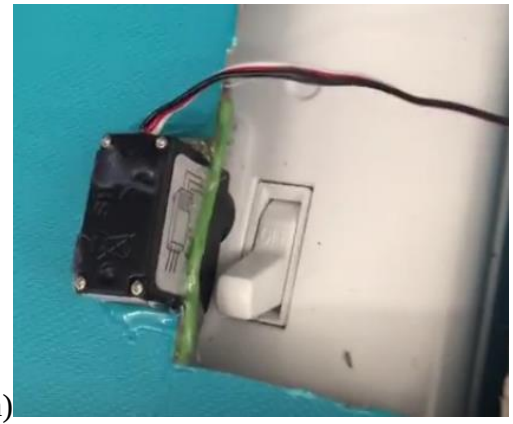
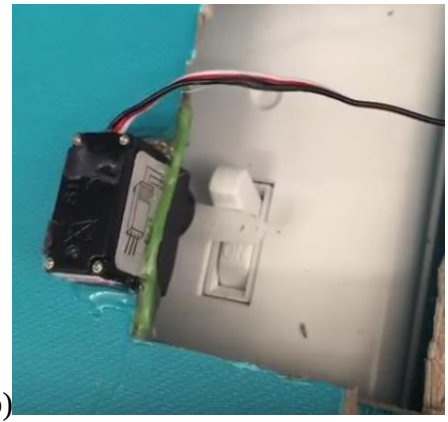
a)  b) 

Figure 2. The servo motor before and after turning on the light switch.

III. Design Evaluation

Overall, we believe that we met the requirements for each of the functional element categories accordingly. First, we had the liquid crystal display for the output display which showed the time or alarm mode depending on the manual inputs. The audio output device was the piezoelectric buzzer, which appropriately functioned as the alarm sound. There was a variety of options for the manual user input, including two buttons, a switch, and numeric keypad. This category, specifically the keypad, required further research in regards to the keypad decoder which we had not been introduced to before. The automatic sensor was the photoresistor, which controlled the brightness of the LCD screen by adjusting its resistance accordingly. The actuator of the system was the Servo motor that turned on the light switch. This too required further knowledge as we had not attempted communication between the PIC and Arduino previously. The logic, processing, and control category was implemented in both the PIC and Arduino. In the case of the PIC, it included programmed logic that included multiple inputs and outputs as well as multiple nested loops. The Arduino's coding was more complex; it managed the time, the keypad input, the alarm time, and motor control. The Arduino was tasked with calculations and data storage/retrieval within its own memory as well as the PIC's memory.

During our initial demonstration, the LCD screen did not work properly as well as the alarm off button due to power issues that were not anticipated on the day of. However, the functionality of both categories were ultimately fulfilled although we do anticipate a grade reduction for not being fully functional until a day after the due date. We feel we have earned full points for the manual inputs, actuator, and logic categories since each of these required further research. The output display, audio output device, and automatic sensor fully fulfill the requirements but do not exceed expectations as they are components we were already familiar with.

IV. Partial Parts List

Name/Description	**Model Number**	**Source**	**Price**
Piezoelectric Buzzer	17855	Electronics Lab	$2.00 (listed)
Keypad Decoder	74C922N	Electronics Lab	$1.70 (listed)
Arduino UNO	Rev 3	Electronics Lab	$24.95 (listed)
PIC	16F88	Electronics Lab	$2.73 (listed)
Photoresistor	CdS	Electronics Lab	$0.95 (listed)
Servo Motor	HS-311	Electronics Lab	$7.89 (listed)
Toggle Light Switch	43009	Home Depot	$6.47
DPDT Toggle Switch	PH-30-10048	Electronics Lab	$2.99 (listed)

V. Lessons Learned

Our initial design was to implement multiple PICs rather than use an Arduino. Our intentions were to use a PIC for different aspects of the alarm. However, we struggled to communicate between multiple PICs. We attempted to learn the slave-master PIC modes. However, the textbook directions were not sufficient for beginners. To rectify this problem, we introduced the Arduino into our system as we were already familiar with its language and programming capacity. With the Arduino, we were able to communicate with the PIC using digital outputs.

The original concept of our alarm clock was to use individually programmable LED strips to display the time. We attempted to use the PIC to control the LED strip but were provided with limited instructions. We tested the LED strips using multiple different inputs, but were provided with no clear response patterns. In fact, the only available information was interfacing it with an Arduino. At this point we had not decided to use the Arduino and so we scrapped the idea of the LED strip altogether. Additionally, the Arduino program required libraries to be installed. For future students, we recommend the individually programmable LED strip only if they intend to begin with an Arduino and have the appropriate informative material.

If given more time, we would have liked to print a custom circuit board. We initially designed one in Eagle and presented it to Ernest. However, the automatic wire mapping was insufficient for the printing technology available and required that we would need to manually draw the mapping of the wires. Considering our time frame, we decided to ultimately use a prototyping solderable perf board and hand solder the connections. This board provided us with a compact and sleek design that was more robust than a protoboard while not having to spend the time designing on an unfamiliar program. Additionally, it allowed us to change circuit designs during testing in the case where debugging was required, whereas we would not be provided with this flexibility with a printed circuit board.

Considering we wanted to make our project as close to a finished product as possible, we attempted to battery power the Wallarm. We could only find 9V batteries even though our PIC and interface circuits required only 5V. The 9V battery was causing issues, mainly to the buzzer and LCD screen. We initially corrected this by applying resistance to the supply to bring down the voltage to 5V. Despite confirming the 5V with a multimeter, the LCD continued to malfunction and the Piezo buzzer did not sound correctly. Because we were unable to find the cause of the malfunctions, we decided to backtrack to the outlet power instead.

In total, we are happy with each of the solutions which we arrived at and still feel that we have made considerable strides in PicBasic Pro and interfacing the PIC to other controllers. Additionally, our success with the prototyping solderable perf board provided us with an alternative to the larger protoboards that we are used to in class. Our failures allowed us to learn the full extent of the components we were using (eg. when it works and when it does not, and why) and encouraged us to come up with creative alternatives that still accomplished the same functions.

VI. Appendix

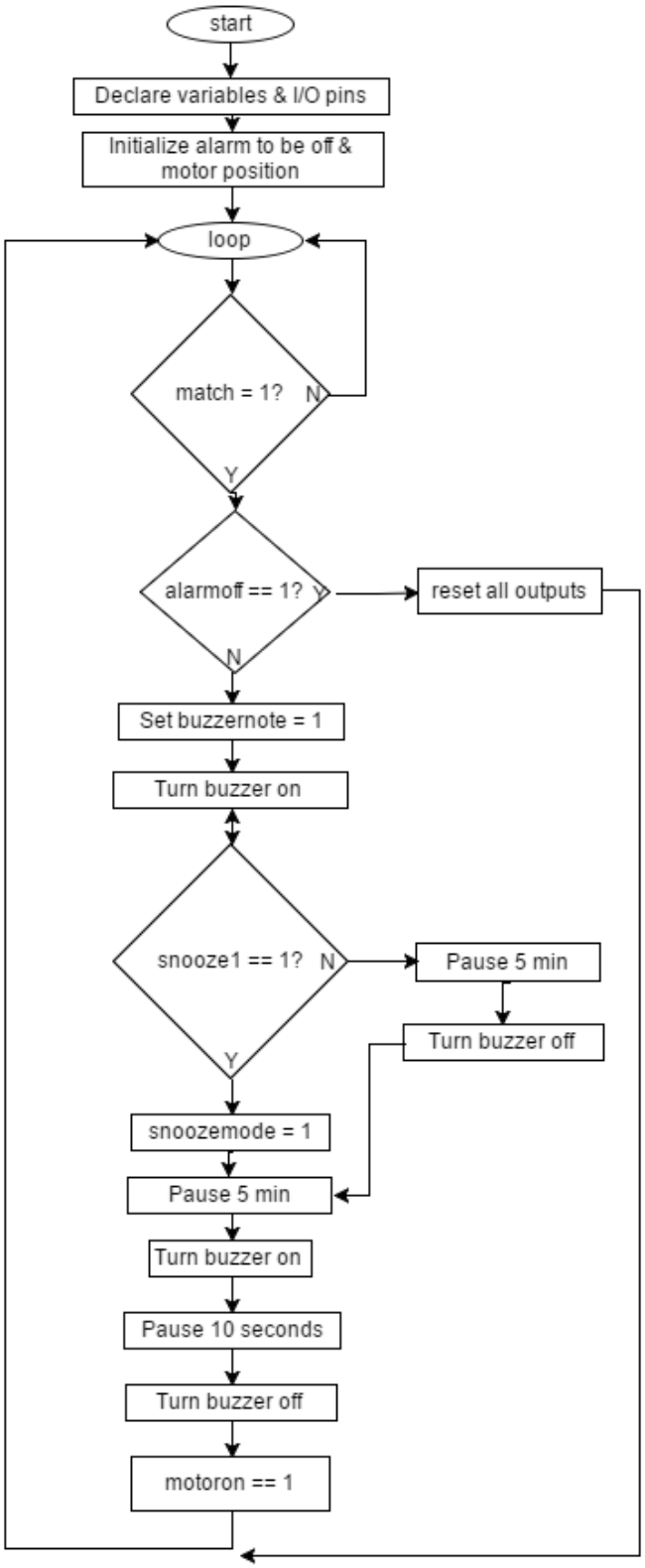


Figure 3. Program flowchart of the PIC

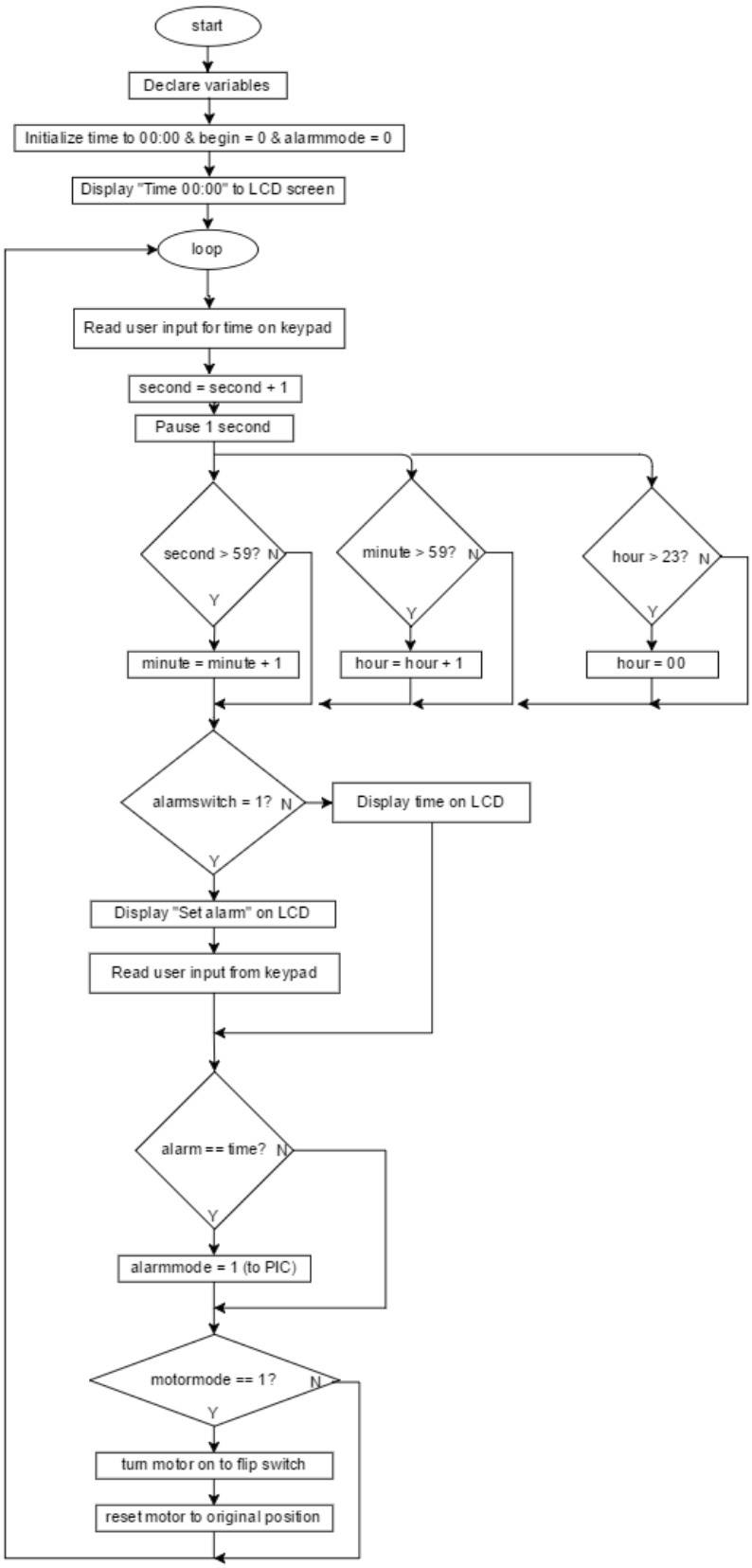


Figure 4. Program flowchart of Arduino

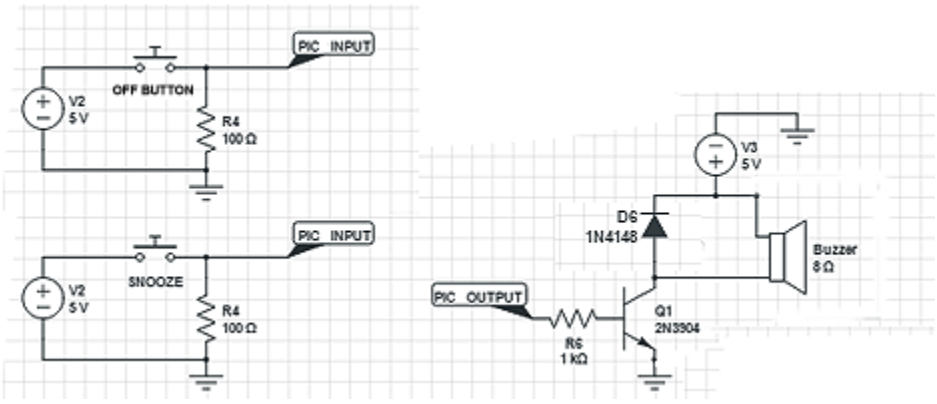


Figure 5. PIC circuit diagram with snooze and off buttons and buzzer

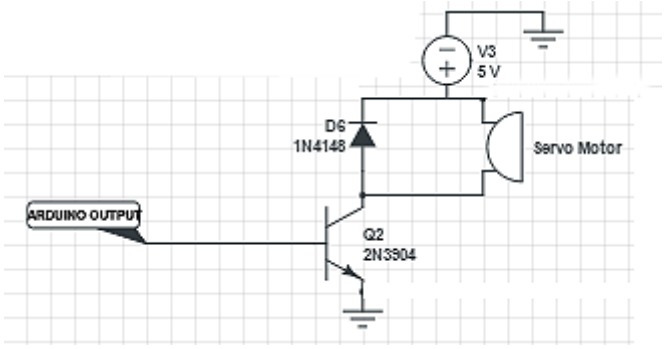


Figure 6. Arduino circuit diagram with the Servo motor.

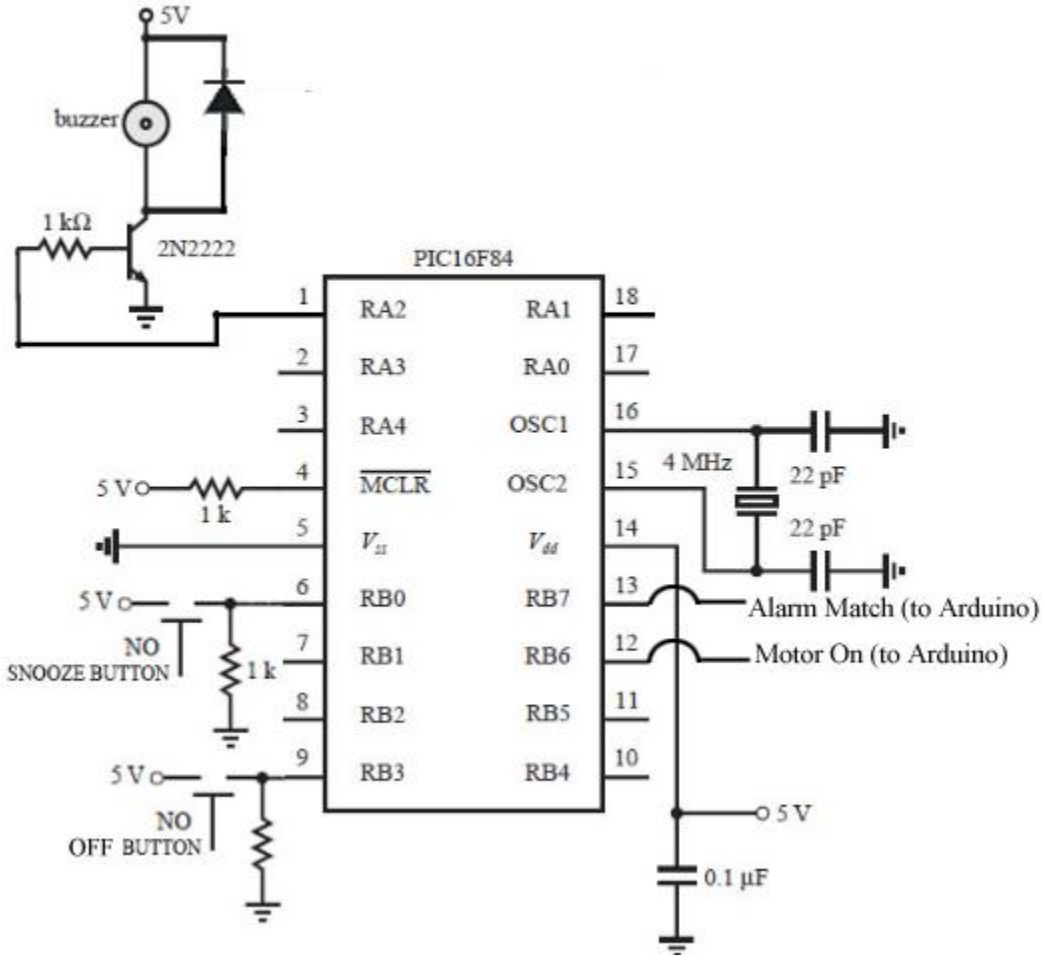


Figure 7. Detailed wiring diagram of the PIC.

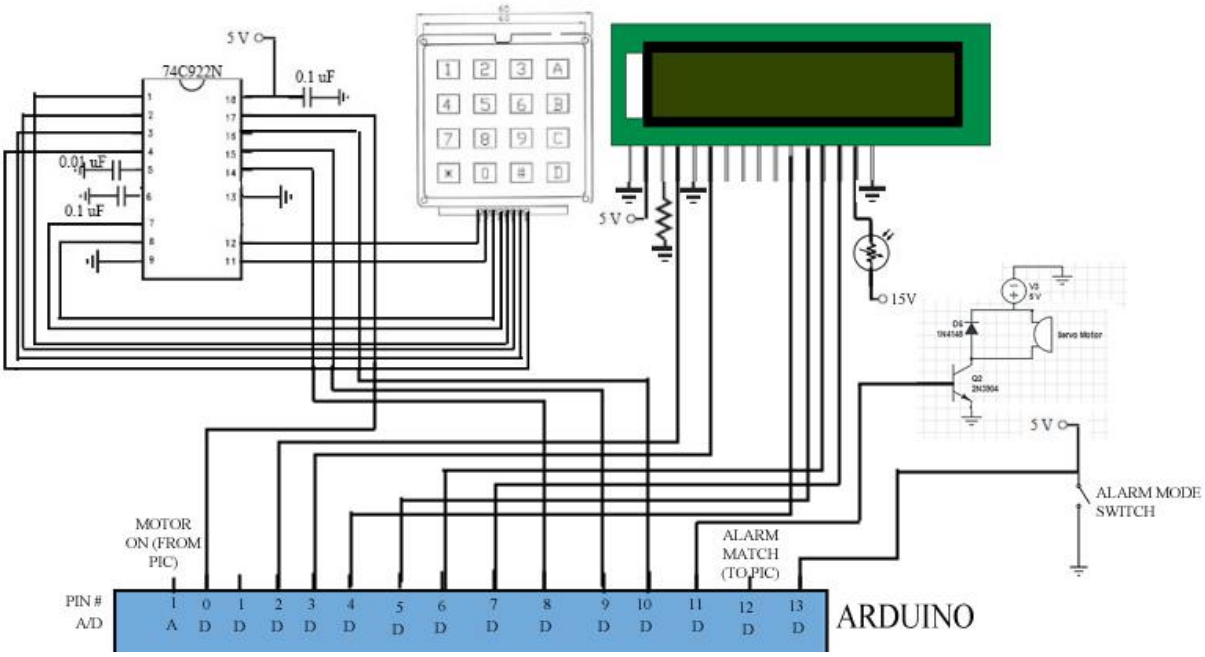


Figure 8. Arduino wiring diagram