Early Bird Alarm Clock

ECE 445, Senior Design

Sasin Gudipati
Siddharth Sharma
Shouri Addepally

Fall 2022
TA: Stasiu C.

Abstract

This paper will outline our senior design project, the Early Bird Alarm Clock. It will go into detail on each aspect of our project, starting with the design process and ending with the testing process. This paper was written by the same three members who completed this project: Sasin Gudipati, Siddharth Sharma and Shouri Addepally under course staff Stasiu C. and Professor Arne Fliflet. The appendix section will provide sources we used, visual documentation of our project and our digital files (as well as a link to its GitHub).

Table of Contents

I. Introduction

The hardest part of starting the day is actually getting out of your bed and waking up. Many people set multiple alarms every morning, only to snooze them all and sleep through them. According to an article written by AmeriSleep, “more than one in three adults press snooze three times before getting up in the morning” [1]. Hitting the snooze button is known to ruin your REM sleep as it will wake you up in the middle of your sleep cycle rather than at the end of it [1]. ScienceDaily provides an article on brain activity, describing that “they found that most people’s brains were most excitable at [early morning times]” [2]. It is also proven by Michael J Breus PhD that “getting up and out” of bed helps you wake up more active than simply setting alarms [6].

Young adults in education have trouble waking up early in the morning and using their brain immediately. So, we wanted to come up with an idea to help get out of bed in time to start the day the right way. Our solution is to create an alarm clock that will help you wake up in the morning. This clock will be an interactive clock as it forces the user to wake up. It will run away from its user once the alarm is triggered, and in order to turn it off, there will be a math equation the user has to solve to turn it off. Once the math equation is solved, the alarm will shut off and the user can go about their day. Section II will describe in detail the physical design of our project as well as the design process.

II. Design

This section will detail the design process from start to finish.

A. Modules

1. Movement Module

For the movement module, the physical components required are three HR-S04 ultrasonic sensors, two 12V DC motors operating at 2000 rpm, and a L298N H-Bridge motor driver. Additional physical components include two wheels and two hubs, manufactured by the machine shop, which allow the 12V DC motors' shafts to fit properly with our alarm clock's wheels.

The movement module also features the main functionality of obstacle avoidance along with quick movement and directional turning based on the ultrasonic sensors' distance readings. The general obstacle avoidance algorithm and enhancements via reinforcement learning will be discussed further under Software Development (Section II.F).

2. User Interface Module

The physical components that relate to this module are the LCD Display and Buttons. On our PCB, we included a shift register (detailed in Section II.D), button connectors and crystal oscillator. This module could be split into two states, the Alarm State and the Clock State. The buttons help manage the input into the clock system, being able to set the clock time in the clock state, change and set the alarm in the alarm state and solve the math equation.

The UI Module displays the time, alarm time and an indicator (for whether or not the alarm is set), shown in Figure 10 (Appendix). The button layout is simple: we allocated three buttons for math functions, one enter button and four buttons for the clock functions. The three buttons for mathematics are to increment the hundreds, tens and ones digits for the number, and the four for the clock functions are increment hour and minutes, set to alarm state and set to clock state.

B. High Level Requirements and Successes

Below are the high level requirements we set to achieve before starting the project.

1. The device can function as an alarm clock. The user should be able to set the current time, alarm time, alarm on and off as well as visibly see an indicator showing that the alarm is ready.

2. The alarm clock can move around the room and avoid hitting an obstacle using ultrasonic sensors.

3. The device will display a math equation for the user to solve, while also accepting an input until the answer is correct. Once the problem is solved, the alarm should turn off and the user can put the car down.

Out of these three high level requirements, we were able to completely meet two of the three of them. We had issues with the second High level Requirement due to the chassis built by the Machine Shop (described in detail in the next section). We were able to generate a completely working User Interface module, where the user could set the alarm, trigger the alarm, change the clock and solve the math equation to turn off the sound. We were able to get the Movement module set up, with basic movement, and also had the sensors tell the motors which direction to go, however, we could not fully test the movement module because of the chassis.

C. Physical Design

Our alarm clock will fit into a chassis that can house all of our components (Appendix, Figure 9). We will have a PCB inside the chassis powering the device, with three batteries connected. We will have motors plugged in (to a H-Bridge) on each side of the car, with a ball-bearing in the back to allow it to be balanced so it can move forward comfortably. Two of the three batteries will be used for the H-bridge (12V power), which will power the motor driver. The third battery is for our PCB.

On the outside of the chassis, we will have an LCD display, a series of buttons and a buzzer all connected to our PCB. We will have three sensors laying in the front of the car, which are giving the distance data to the microcontroller.

D. Parts List

Part	*Purpose*	*Quantity*	*Cost*
16x2 Serial Interface Adapter Module Blue Backlight	LCD Display to display clock, alarm and math systems	1	$15.99
4 PIN Push Buttons Switch	Buttons for User Interface	8	$2.99
MH-FMD Piezo Buzzer	Buzzer for Alarm	1	$1.99
Motors	DC 12V 200RPM Gear Motor High Torque	2	$29.98
HC-SR04	Ultrasonic Sensor	3	$9.99
6V, 2000mAh NiMH Battery	Batteries	3	$29.98

Other parts include the hardware made by the Machine Shop at the ECE Building. They provided a chassis (shown in Figure 10 of the Appendix). A slight issue with the build was that it was made with aluminum, so the bottom of our PCB (which had leads poking through from the solder) would provide a current to the aluminum box. So, we added wrap around our PCB to help fix this issue. But, this shorting issue continued, and we were wary of completely closing the top of the chassis. This is the main reason the movement module would not work. We used the multimeter and we got a reading of roughly 0.42V off the aluminum lid when the chassis was fully closed.

E. Major Design Choices

- **Microcontroller Choice: ATMEGA328-P**

This design choice was by far the most important one we had to make. We originally decided to choose the PIC-16F877A. This choice was made because we liked how easy it was to program and how fast it could execute instructions (200 nanoseconds) [3]. However, after completing more research, we decided to use the ATMEGA328-P. This microcontroller is able to complete instructions also at very quick intervals (in a single clock cycle). The ATMEGA328-P also has a very good approach to balancing power consumption and processing speed. Below are the comparison stats for the microcontroller.

ATMEGA328-P [4]	*Specification*	*PIC-16F877A [3]*
Flash	***Program Memory Type***	Flash
32	***Program Memory Size***	14
20	***CPU Speed***	5
1024	***Data EEPROM***	256
6	***Stand Alone PWM***	0
8	***ADC Channels***	8

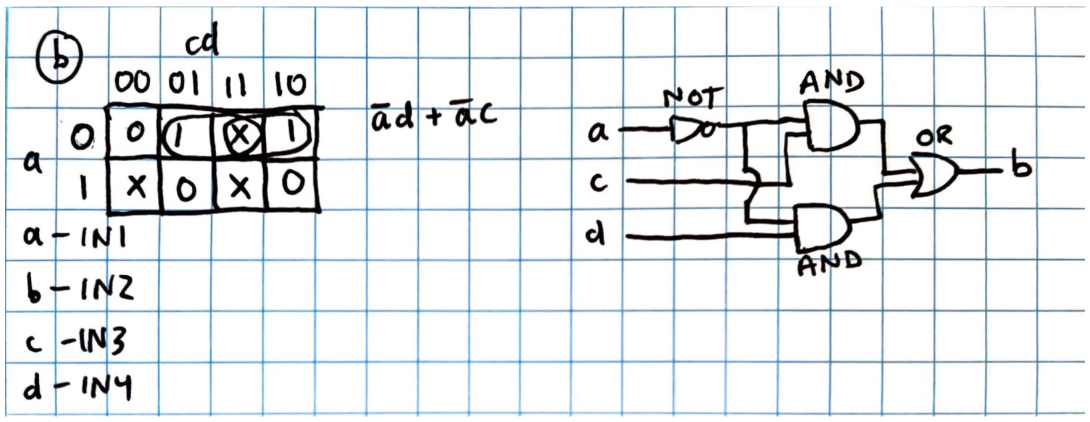
As you can see, the ATMEGA328-P has more Program Memory Size, CPU Speed, PWMs and Data EEPROM (user-modifiable ROM that can be erased and reprogrammed repeatedly through a normal electric voltage) [5].

- **Motor Choice: L298HN (H-Bridge)**

  The motor we decided to purchase was one that could power two motors at once. We decided on the L298HN Dual Full-Bridge Driver to support this. This driver takes up to 50 Volts to power the motors and from 0.3 to 7 V of input voltage to power the driver (which comes from our Circuit Board).

- **Saving of I/O Pins:**
  - **Motor Logic:** When designing our circuit, we realized that there were a limited number of input and output pins. So, to save pins on our circuit, we wanted to be able to create a signal (IN2) using a Karnaugh Map. We figured out the five signals we needed:
    - Forward: 1010
    - Left: 1001
    - Back: 0101
    - Right: 0110
    - Stop: 0000

    We were able to create the following table with three inputs (IN1, IN3, IN4) to calculate IN2. For example, if we provide IN1, IN3, IN4 as 0, 0, 1 (respectively), we should be getting 1 as IN2. Below is the work shown to achieve the output.



- **Shift Register:** Another way of saving pins was to use a shift register for our buttons. We used this to convert eight button input to 4 output pins that go into the microcontroller. The shift register we chose is the 74HC165, which is a Parallel-Load (in), Serial-Out register. The 74HC165 is a 8-bit shift register that takes in the button values as a binary value, so for example if button 7 (enter) was clicked, then it would print 00000001 to the microcontroller, and that value would be taken in as the user input.

- **Ultrasonic vs. Lidar:** This choice was a major design choice for our movement module. The main arguments for ultrasonics included a better price point, less memory usage and availability. We found lidar sensors to be better at environment detection and mapping. However, with how little memory our microcontroller had, we could not allocate enough to mapping the environment, so we decided to use the HCSR04 Ultrasonic Sensors that were readily available. The cost price point helped keep our project cost down while also achieving basic object detection with the vehicle.

F. Software Development

We have also set up our codebase in a module format, where there are multiple .cpp and .h files that complete different functions of the project. Additionally, to make the code more readable, we created a hierarchy for each file. The top of the hierarchy is AlarmClock.ino. The following are the file names: motors.cpp, ultrasonics.cpp, buttons.cpp, obstacleAvoidance.cpp. The text below will describe the codebase purpose of each file and any significant algorithms used.

AlarmClock.ino - Top Level File

Function: The AlarmClock.ino file serves as the top-level module of our software and instantiates objects of all the other submodules in its code. The code then gets the current time from the RTC and displays this on the LCD display, along with an "EarlyBirds Meow" default text. Then, based on which buttons are pressed by the user, the alarm clock module handles that specific scenario. If the user's button input corresponds to the alarm-set function, then the user is able to cycle through the hours and minutes fields and set a time of their choosing. Once the alarm has been set, the speaker will start ringing, and the motors will initiate movement. Once the alarm is set, the user is able to turn it off based on an answer to a randomly generated decimal addition problem. The user is once again able to use the buttons to input and cycle through values at the ones, tens, and hundredths places. If the user answer matches the internal computation to the problem, then the speakers are shut off and the movement will be stopped as well. Otherwise, the speakers will continue to ring, and the motors will continue their movement.

Motors.cpp

Function: The motors.cpp file contains basic motor driver code which enables the alarm clock to move around and perform turns. Specifically, the motors code module contains functions which allow the alarm clock to move forwards, backwards, turn left, turn right, and stop. The motors module is then called in the top-level hierarchy file, where the respective motor functions are called to control the alarm clock's general movement. This module is also instantiated as an object inside of our obstacle avoidance software module.

Ultrasonics.cpp

Function: The ultrasonics.cpp file contains code which obtains readings from the three ultrasonic sensors which are mounted at the front of our alarm clock. Because the ultrasonic sensors return distance readings in float format, we have written individual functions to extract distance readings from each of the three sensors. In each of these functions, we also "filter" out distance readings which are either too small (close to the sensor) or too large (far from the sensors). The filtering is done by simply returning a MAX_DISTANCE constant which is set to an arbitrarily large value, which allows us to identify this reading as improper and not utilize it during our basic obstacle avoidance algorithm. This ultrasonics module is instantiated as an object inside of the obstacle avoidance software module as well.

buttons.cpp

Function: The buttons.cpp file contains code which initializes a button vector and allows it to hold a value based on various button inputs. This button vector holds 8 button inputs which are either a binary value of 1 when pressed, or 0 when not pressed. The buttons module is then called in the top-level file as an object, and its button vector is extracted and used as part of the decision making process in the alarm clock code.

obstacleAvoidance.cpp

Function: The obstacleAvoidance.cpp file contains code which performs a simple obstacle avoidance algorithm using the motor driver, sensor, and buttons code respectively. As part of our basic obstacle avoidance, we repeatedly fill an array of three elements with ultrasonic sensor readings from each of the three sensors (left, middle, right). This module contains a preliminary check using the input buttons' pressed vector which determines whether or not the user has pressed the appropriate buttons before the algorithm begins. Then, based on the distance we acquire from the middle sensor, we immediately move backwards if the distance reading indicates that we are close to any object. Otherwise, we take a max across all of the distance readings and turn in a direction which returns the greatest distance value. This obstacle avoidance code is then called in our top-level file.

III. Verification

Below are our Verification Tables for our Modules and Sub-Modules. These helped us progress in our project as it would help us achieve objectives every meeting.

A. UI Module

Requirement	Verification
● The microcontroller should be able to iterate through the set of values (0-9) and display the values on the LCD Display.	● We can test this by observing the LCD display and checking if the values are iterated through properly.
● The microcontroller should be able to utilize the time subsystem and keep track of the time and display it on the LCD with a margin of error of 1 second	● A test can be done by the microcontroller displaying the time on an LCD and comparing the time to the clock on a computer.
● For a single-digit hour or minute value, the microcontroller should include a 0 in front of the digit as part of the time displayed on the LCD.	● We will increment input via our buttons and verify that a leading 0 is present for any single-digit hours or minutes.
● The microcontroller should be able to take in push button inputs for time-setting and should be able to loop back if the minutes' input exceeds 59 or if the hours' input exceeds 23.	● We will use the buttons to continue incrementing the values in both the hours and minutes fields until they hit their respective thresholds of 23 and 59. Once these thresholds are met, we can verify if an additional button press loops back to the 0 values.
● When the clock-set or alarm-set buttons are pressed, the microcontroller should display clock-set and alarm-set screens. When the enter button is pressed the microcontroller exits the clock/alarm set state and returns to the default screen	● We will connect three buttons to the microcontroller, clock set, alarm set, and enter. We will also connect the LCD to the microcontroller. When the clock set button is pressed, the microcontroller must use the LCD to display the clock set screen. When the alarm-set button is pressed, the microcontroller must display the alarm-set screen on the LCD. If the enter button is pressed in either of these states, the microcontroller must display the default screen on the LCD.
● The microcontroller can control the buzzer to play an alarm at a specific tone and for at least 15 seconds.	● We can observe the length of time that the buzzer outputs a sound.

B. Movement Module

Requirement	Verification
● There are three ultrasonic sensors at the front of the alarm clock. The microcontroller must be able to identify which sensors have objects in front of them	● As a unit test, we will connect the sensor array to the microcontroller, and connect the microcontroller to 5 LEDs, each of which represents one sensor in the array. We will then place objects within the sensors' detection range, and the microcontroller indicates which sensors detected objects by illuminating the corresponding LED.
● The microcontroller can engage the motors to turn the chassis 90 degrees right/left. Each turn should take place in less than 1 second +/- 0.25 seconds	● The unit test will involve writing a program that drives the motors to perform turns in the following order: ○ Right, 90 degrees ○ Left, 90 degrees ● For this test, the motors and wheels need to be mounted on the chassis. We will start the program and verify that the vehicle is turning in the aforementioned order in under 1 second for each individual turn.
● The microcontroller can engage the motors to move the chassis in reverse for 15 +/- 0.5 cm in under 0.5 +/- 0.25 seconds	● Once again for this test, the motors and the wheels need to be mounted on the chassis. Next, using the wheel circumference and the motors rotations per second, we will calculate the amount of time we need to run the motors in reverse to move the chassis 15 cm backwards. Once the program is loaded onto the microcontroller, we will initiate the program and simultaneously start a stopwatch. At 0.75 seconds, we will stop the stopwatch and measure the chassis displacement.
● The microcontroller can engage the motors to move the chassis forward 1 +/- 0.1 meter in less than 3 +/- 0.25 seconds	● This test is identical to the previous test, except the chassis is moving forward for 1 meter, and the stopwatch will be stopped after 3.25 seconds.
● The microcontroller can use the ultrasonic sensor input to drive the motors so that the chassis never makes contact with any obstacles.	● The unit test for this requirement consists of a program that drives the motors and monitors the output of the ultrasonic sensors. The car's default state is to move forward. Depending on the particular combination of ultrasonic sensors that detect an object, the program will choose either to turn the chassis 90/45 degrees in a certain direction, move forward, or reverse. ● The obstacle avoidance program will be tested using the three test cases: 1. freestanding objects (such as a toy lying in the middle of the room) 2. Walls 3. Corners

C. Math Module

Requirement	Verification
● The microcontroller can generate two random two-digit hexadecimal operands and compute their sum	● Create a subroutine that randomly generates two operands and calculates their sum. The microcontroller will output these operands to the serial port. We can view the two operands and the sum by connecting the microcontroller to a computer. We will run the subroutine 50 times to ensure that operands are random and the sums correct.
● The user will use three buttons to input an answer and an 'enter' button to submit. The hundreds place, tens place, and ones place each have a dedicated input button. When an input button is pressed, the value in the corresponding place should increase by 1 until the value reaches 9, at which point the value should reset to 0.	● A unit test for this requirement consists of outputting the value of the answer punched in by the user to the serial port, and connecting the serial port to a computer where we should see the answer value changing in response to button presses.
● The microcontroller can identify when the user has inputted the answer that matches the calculated sum for the math equation	● This requirement will be tested by connecting the microcontroller to the computer using the serial port. We will run the subroutine that generates the math equation, and then use the push buttons to submit answers. The microcontroller will send a 1 to the serial port when an answer that matches the calculated sum is entered.

D. Power Module

Requirement	Verification
● The power subsystem needs to provide 5V +/- 0.3V to the LCD, the ATMEGA328P, the HCSR04 ultrasonics, the motor driver, the RTC, the buzzer, and the push buttons.	● Use an oscilloscope to measure the output voltage of the voltage regulator and verify that it is between 4.7 to 5.3V.
● The power subsystem needs to provide 11-13V to the DC motors.	● Use an oscilloscope to measure the output voltage of the voltage regulator and verify that it is between 11 to 13V.

IV. Operational Efficiency

A. Costs

The cost of this project was $350 for parts and circuit boards. The added cost is an implied cost based on industry standard for the salary of hardware and software engineers, which ended up totaling to roughly $7,600. We spend roughly 40 hours a week for 5 weeks working on this project, at an industry rate of $38 an hour. This makes the total project cost just under $8,000 for 3 engineers (roughly $2,600 a person).

B. Timeline

Task	*Module*	*Timeline*
Project Proposal, Design Document, Preliminary Research	N/A	Early September
Parts Ordering, Start of PCB Design	N/A	Late September
PCB Design I, Sensors	Movement Module	Early October
Motors	Movement Module	Late October
Clock Function	UI Module	Late October
Motors with Sensors	Movement Module	Early November
PCB Design II, Alarm Function	UI Module	Early November
Buttons	UI Module	Early November
Alarm with Motors	Movement, UI Modules	Early November
Math Function	UI Module	Early November
Buttons with Alarm, Math and Clock Functions	UI Module	Late November
Testing	Movement, UI Modules	Late November
Final Demo, Final Project Report, Final Presentation	**N/A**	Late November, Early December

C. Group Responsibilities

Our team split responsibilities of the project based on interest and experience. However, instead of splitting responsibilities and working separately, we wanted everyone to be involved in each part of our project. So, we gave each person a part of the project they would take lead on. Specifically, Sasin is an Electrical Engineering major and is proficient in design softwares, so he took the lead for designing the schematic and the circuit board. Siddharth and Shouri are proficient in programming languages C and C++ so they took the lead on the programming aspect of our project. Specifically, Siddharth took the lead on the UI Module and Shouri took the lead on the Movement Module. This worked really well because each person was able to contribute to each part of the project to a high level, so there is a very good common understanding of what is occurring during the project. Additionally, it made debugging certain parts easier as each student could figure out what each part of the project is doing.

D. Group Organizational Standards

Our group practiced proper documentation techniques: writing down information consistently, organized GitHub practices and sharing files through a Google Drive folder. We were able to meet consistently every week (roughly 4 to 5 times a week), and were able to easily communicate with each other outside of class or lab time. We each contributed evenly and were in attendance at each and every meeting.

V. Conclusions

We started off this project saying that we had two main objectives: learn as much as we can and complete this project. We ended up focusing on the first and realized the second would follow as we completed each objective of the project.

All in all, we were able to nearly finish the project and succeed. The project definitely tested our ability to stay on deadlines, knowledge of intricate electrical components and systems, programming as well as our creativity. We worked well as a group as we each tackled an equal part of the project and used our expertise to our advantage. We had a good balance of work and life, making sure that we were working at reasonable hours, rather than sleeping late and getting up early. This helped us be more productive as we were working very efficiently because we were well rested. We also made sure to study outside of the class, learning about the project in new ways and making sure we could apply what we learned to the objectives we completed. With our emphasis on research, we were able to get our PCB working on the first order, making it easy to finish the project in time.

However, with all of our successes, there were some troubles when working on the device. We had trouble working with the machine shop because we had very little experience with mechanical components and were not sure which parts would work. They were very helpful and gave us tips that helped us finish this project.

Outside of what we completed for the demonstration, we had a few ideas on how we could get the project to be more efficient. With more time, we could focus on our movement algorithm and get it to decide quickly and move faster. We could also work on the chassis a bit and make it look a little less industrial. A major change we could make was to decrease the size of the PCB, use less batteries and make the final product a little lighter so we could house all these components in a smaller body.

Overall, this project was a success as we learned a lot about the design process of a product from start to finish and were able to complete objectives along the way.

VI. Appendix

A. Works Cited

[1] M. Taylor, “The Negative Impact of Hitting the Snooze Button,” *Amerisleep*, 14-Nov-2022. [Online]. Available: https://amerisleep.com/blog/negative-impact-snooze-button/. [Accessed: 07-Dec-2022].

[2] U. of Alberta, “Morning People and Night Owls Show Different Brain Function,” *ScienceDaily*, 24-Jun-2009. [Online]. Available: https://www.sciencedaily.com/releases/2009/06/090623150621.htm. [Accessed: 07-Dec-2022].

[3] Microchip. (n.d.). *PIC16F877A - connected | secure | microchip technology*. PIC16F877A. Retrieved December 7, 2022, from https://www.microchip.com/en-us/product/PIC16F877A

[4] Microchip. (n.d.). *ATMEGA328P - connected | secure | microchip technology*. ATmega328P. Retrieved December 7, 2022, from https://www.microchip.com/en-us/product/ATMEGA328P

[5] Lithmee, “What is the difference between PROM EPROM and EEPROM?,” *Pediaa.Com*, 07-Feb-2019. [Online]. Available: https://pediaa.com/what-is-the-difference-between-prom-eprom-and-eeprom/. [Accessed: 07-Dec-2022].

[6] J. Guerra, “12 Quick Ways to Wake Yourself Up in the Morning, from Sleep Specialists,” *mindbodygreen*, 12-Mar-2012. [Online]. Available: https://www.mindbodygreen.com/articles/how-to-wake-yourself-up. [Accessed: 07-Dec-2022].

B. Diagrams

Figure 1: Original Block Diagram

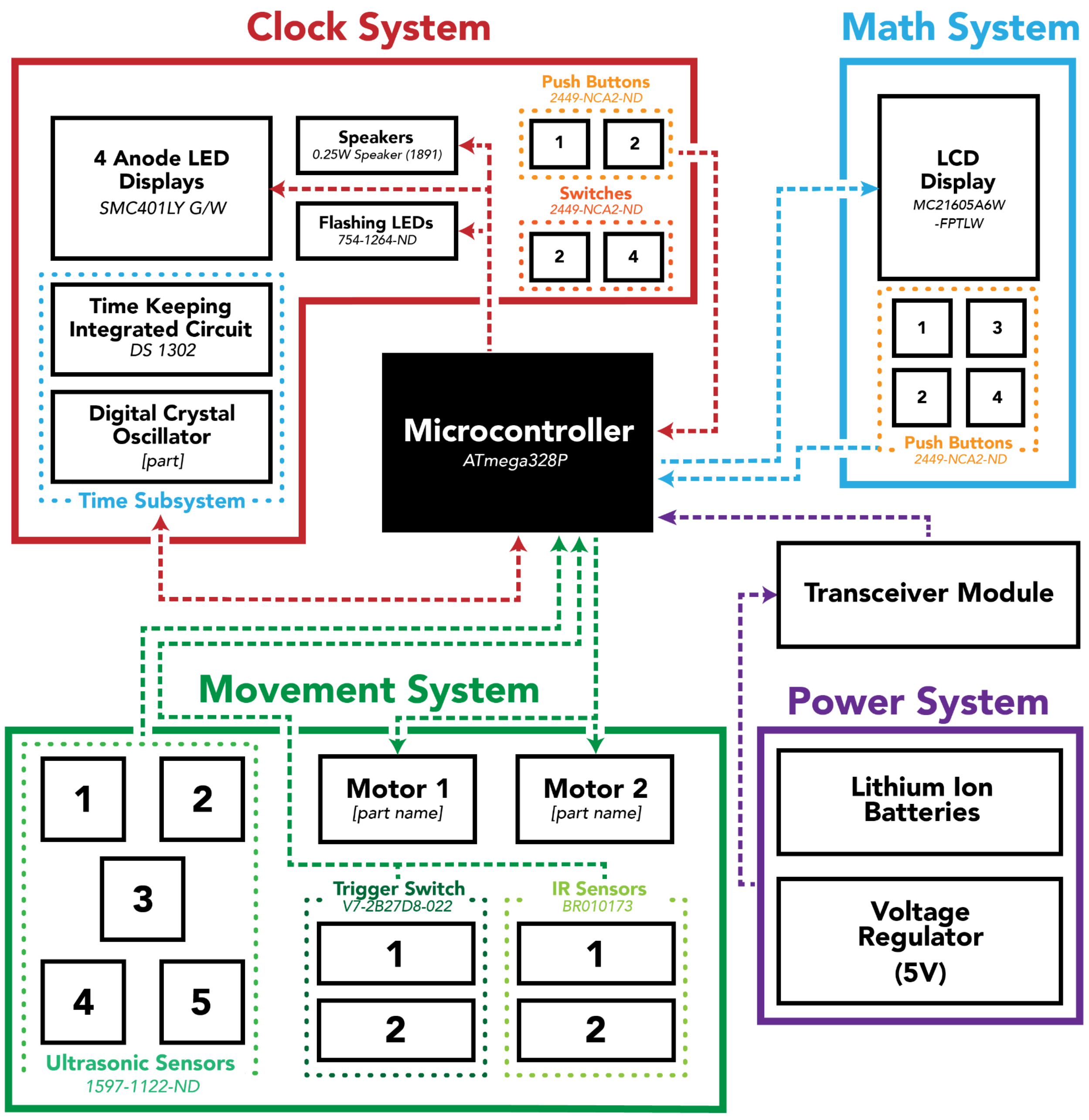


Figure 2: Final Block Diagram

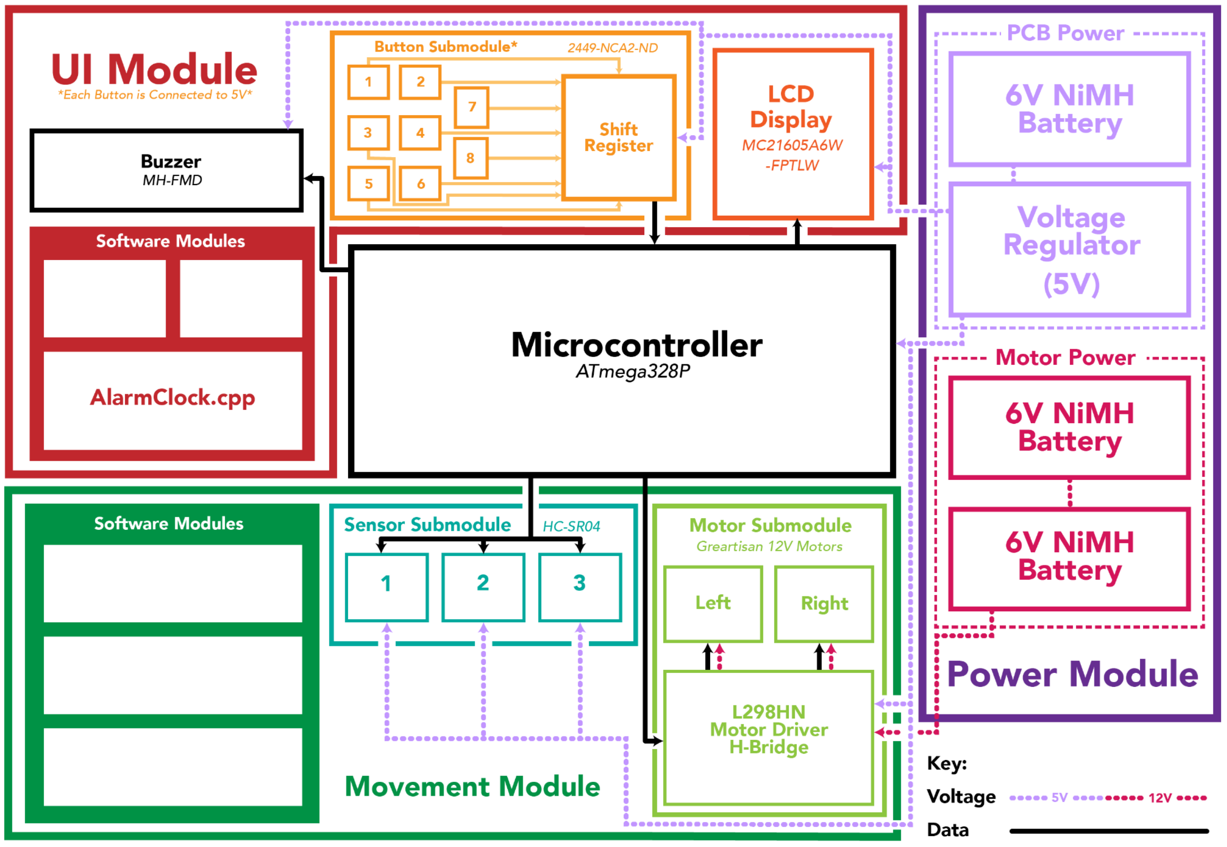


Figure 3: Original Printed Circuit Board

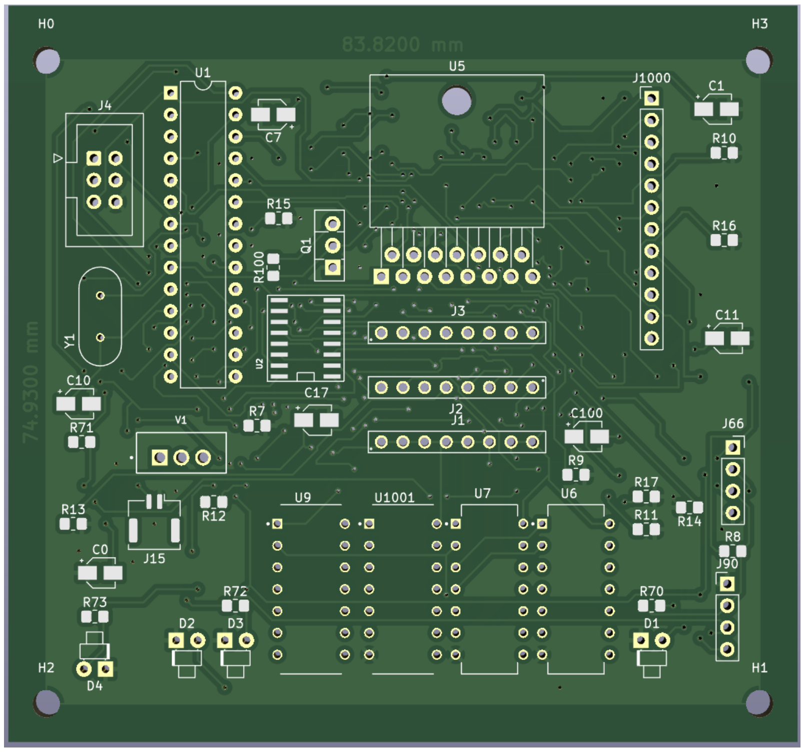


Figure 4: Final Printed Circuit Board

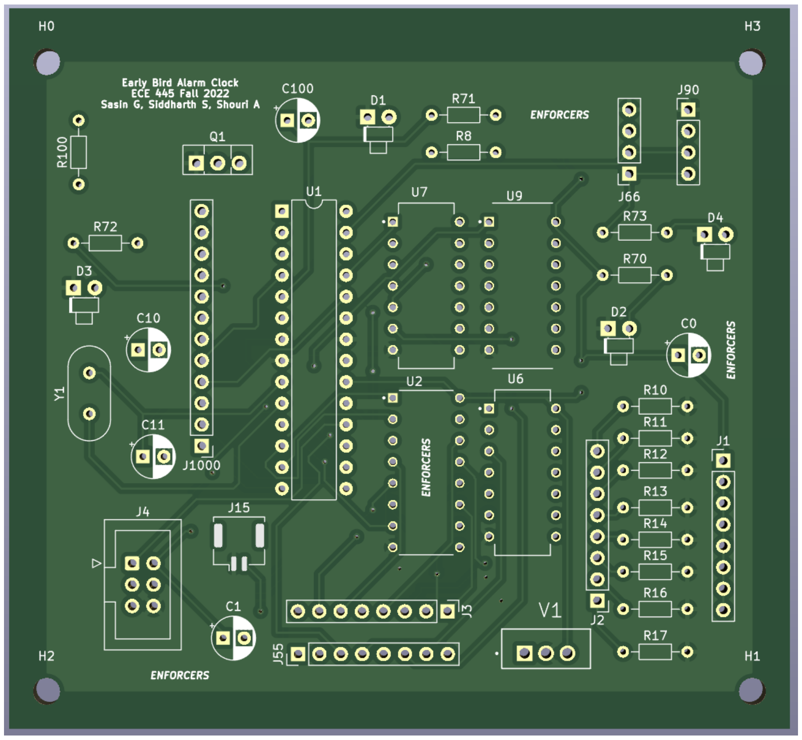


Figure 5: Original Circuit Schematic

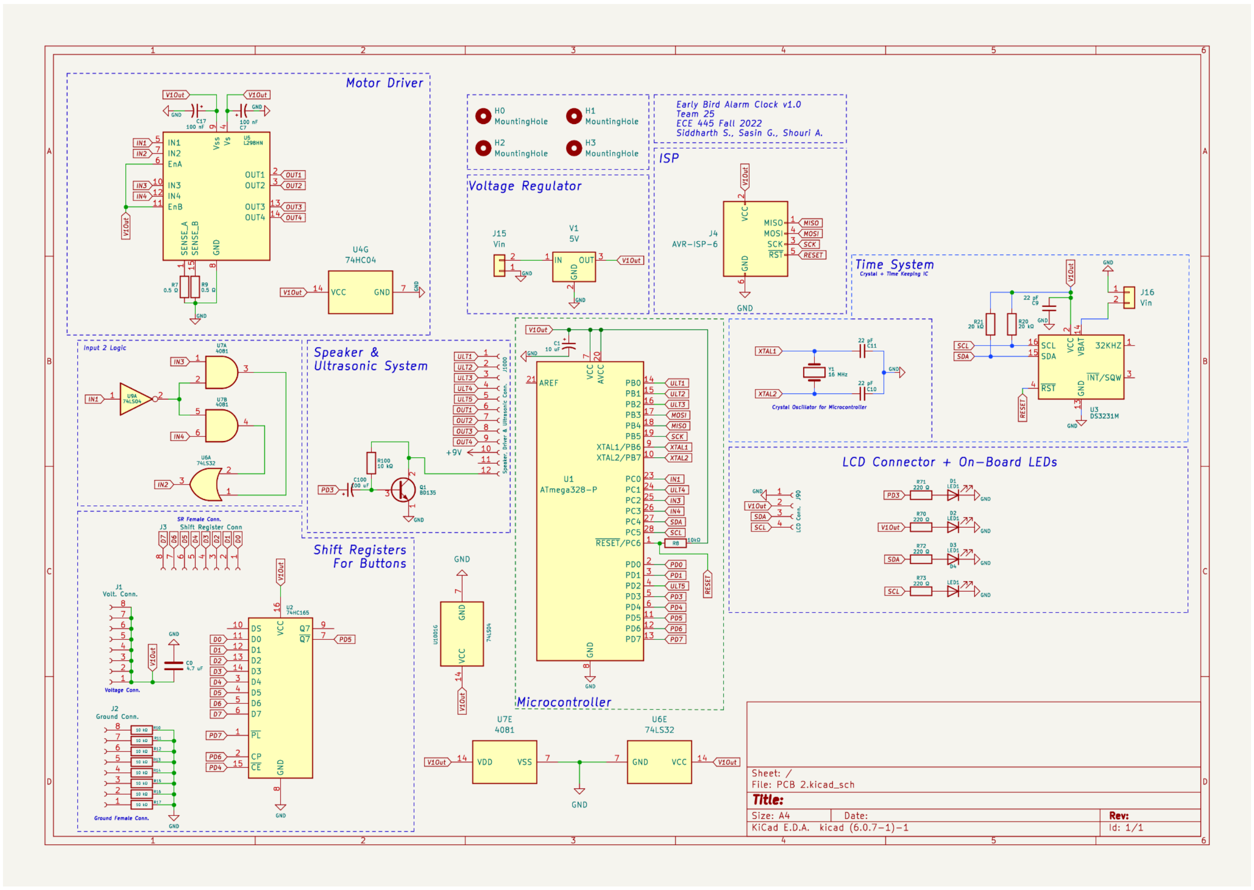


Figure 6: Final Circuit Schematic

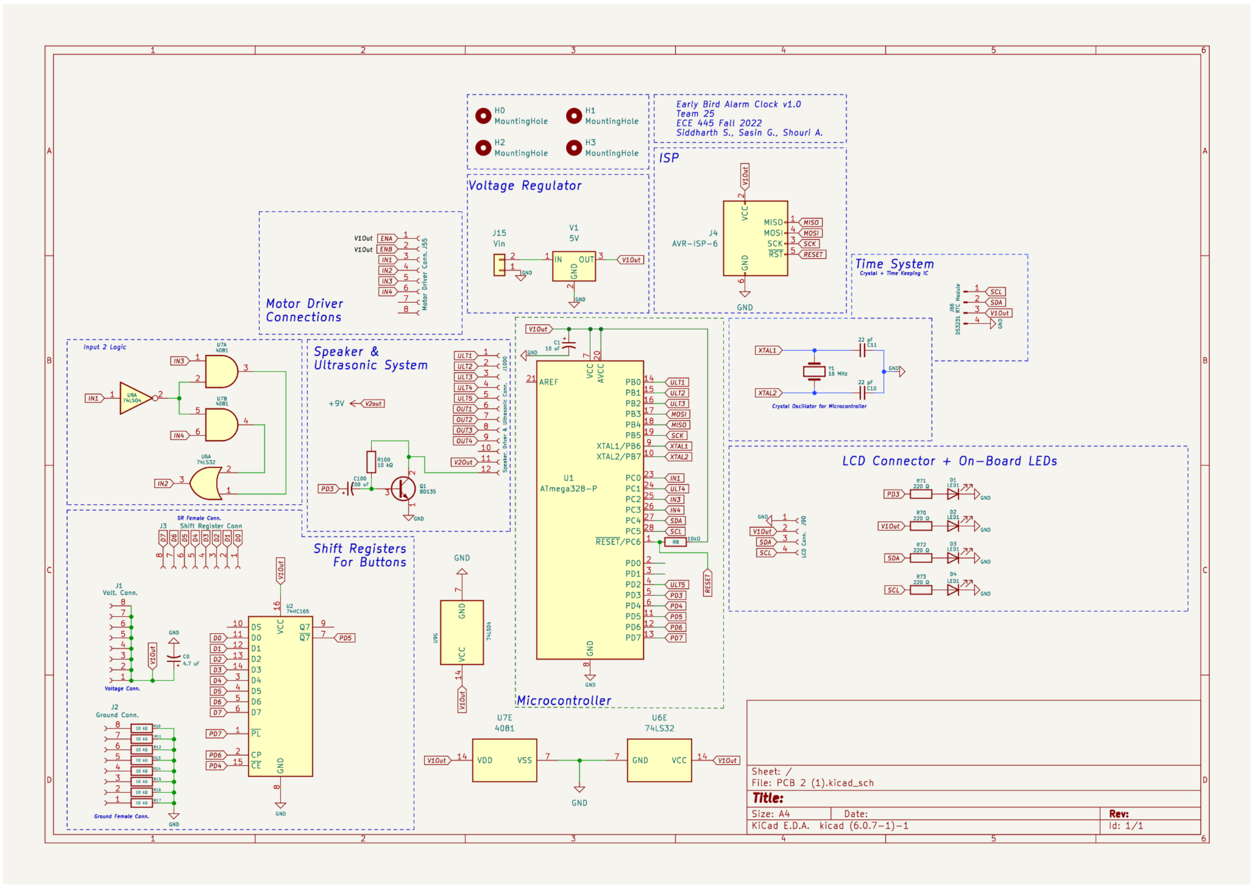

Early Bird Alarm Clock v1.0
Team 25
ECE 445 Fall 2022
Siddharth S., Sasin G., Shouri A.
Voltage Regulator
ISP
Motor Driver Connections
Time System
Input 2 Logic
Speaker & Ultrasonic System
Shift Registers For Buttons
Microcontroller
LCD Connector + On-Board LEDs
ATmega328-P
Sheet: /
File: PCB 2 (1).kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (6.0.7-1)-1
Id: 1/1

Figure 7: Original Visual Aid

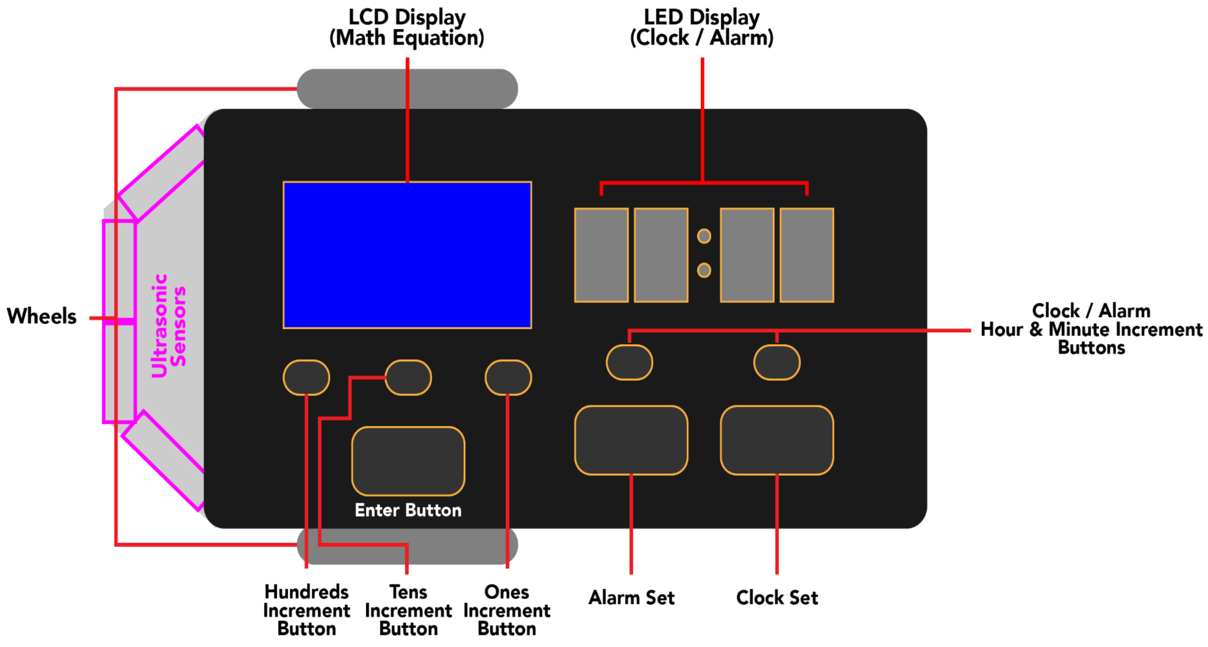

LCD Display (Math Equation)
LED Display (Clock / Alarm)
Wheels
Ultrasonic Sensors
Clock / Alarm Hour & Minute Increment Buttons
Enter Button
Hundreds Increment Button
Tens Increment Button
Ones Increment Button
Alarm Set
Clock Set

Figure 8: Final Visual Aid

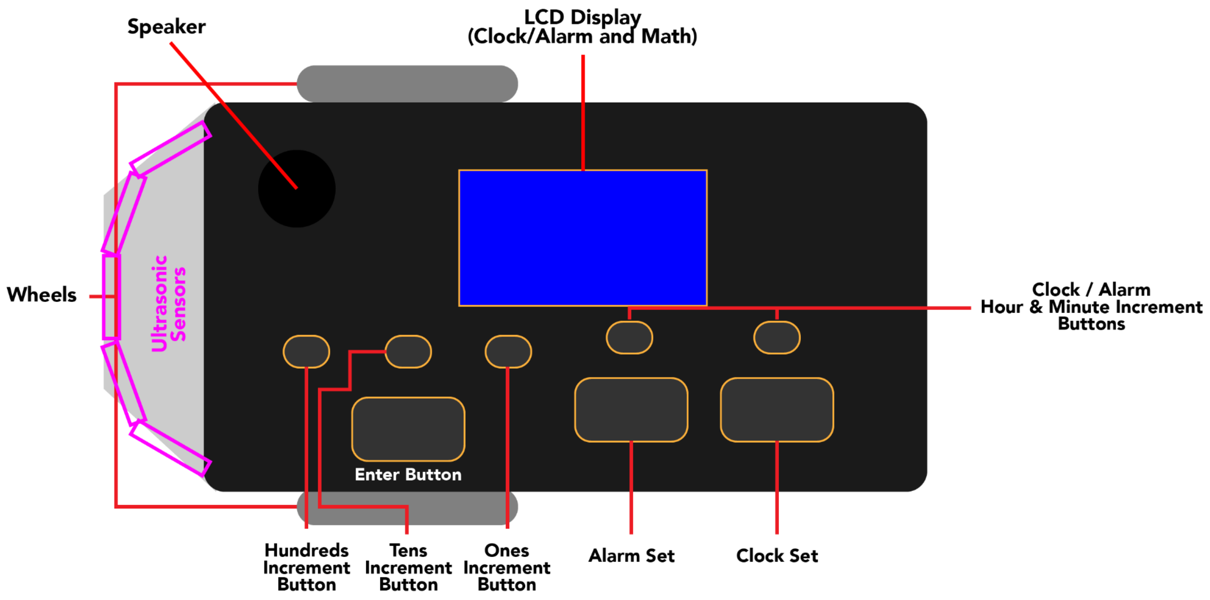

Speaker
LCD Display (Clock/Alarm and Math)
Wheels
Ultrasonic Sensors
Clock / Alarm Hour & Minute Increment Buttons
Enter Button
Hundreds Increment Button
Tens Increment Button
Ones Increment Button
Alarm Set
Clock Set

Figure 9: Chassis

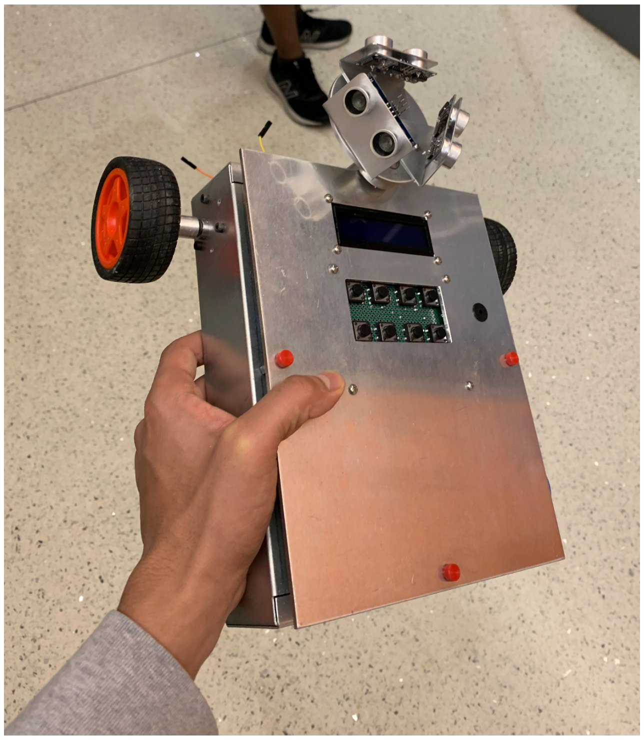
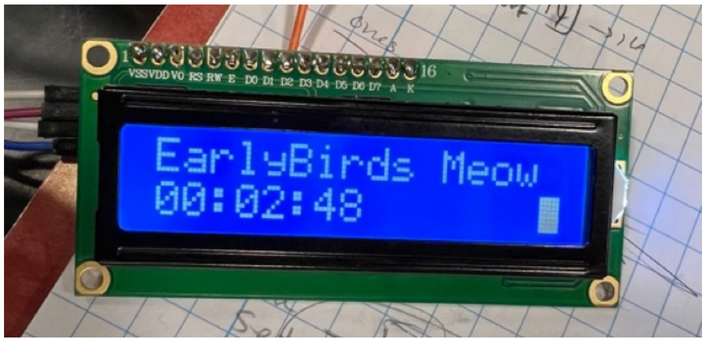


Figure 10: LCD Display



C. External Links

Codebase and KiCad Files (Located along with Codebase): Codebase Link
Presentation: Presentation Link