Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN 1093 – Laboratory Practice I

DIGITAL ALARM CLOCK

Shyamal M.A.L.	- 180601T
Aakkash G.	- 190001M
Abeywansa C.N.	- 190005E
Abeyratne M.D.A.J.	- 190009U

Supervisor: Kithmin Wickremasinghe

This is submitted as a partial fulfillment for the module
EN1093: Laboratory Practice I
Department of Electronic and Telecommunication Engineering
University of Moratuwa.

16th of August 2021

ABSTRACT

In this project, our aim was to design a digital alarm clock, which is capable of setting multiple alarms and playing musical melodies while alarming. First, we divided our main product into sub products. As an experimental design, we used the aid of Arduino boards to develop these sub products. Then we developed them using AVR firmware. We also examined the issues of developing the solution with simulation software. As the final product, we were able to design a digital alarm clock which can set date, time and 5 alarm times, which could also change the alarm tone.

TABLE OF CONTENT

INTRODUCTION

Starting from the pendulum clock the world has come a long way up to the atomic clock with the help of science and technology. However an automated digital clock is the most popular time keeping device as it is more convenient in day-to-day use. Today, digital clocks are mainly designed using microcontrollers as the microcontroller consists of almost all the logical devices and memory elements. It makes alarm clocks even more user friendly and accurate.

Our product is designed using an ATmega328P microcontroller for decision making and system controlling, DS 1307 Real Time Clock(RTC) to keep track of time, LCD for displaying the date and time, etc. and a keypad to input digits.

This is the list of objectives that are expected to be achieved from this product.

- Simple user interface
- Ease of access
- Portable size.
- Simple flow and functionality
- Having multiple alarms with multiple tones

ACKNOWLEDGEMENT

First of all, we extend our sincere gratitude for all the Semester 2 lecturers who gave us great insights and knowledge on the field of electronics and telecommunication.

Next, we would like to express our special thanks to our project supervisor, lecturer Mr. Kithmin Wikramasinghe who gave us the golden opportunity to do this wonderful project on the topic of Digital Alarm Clock, which also helped us in doing a lot of research and gaining knowledge about Arduino firmware, AVR firmware, electronic concepts, CAD drawing, and PCB designing. We are really thankful to him.

Furthermore, we are really thankful for the resource persons who dedicated their valuable time and conducted training sessions to give us the knowledge on Solidworks designing and Altium PCB designing.
Finally, we would also like to thank our friends who helped us in many ways to complete this project successfully.

Date : 16/08/2021 Team Locus

METHOD

From here onwards the following abbreviations will be used for the names of the group members.

Shyamal M.A.L.	- LS
Aakkash G.	- AG
Abeywansa C.N.	- CA
Abeyratne M.D.A.J.	- AA

▪ **<u>Selecting Subcomponents</u>**

First of all, our group members got together and had a discussion about the digital alarm clock. We were able to identify four major subcomponents as follows.

1. Keypad Input
2. LCD Display
3. RTC Module
4. Alarm Audio

The main tasks of these four subcomponents were inputting numbers for date and time, displaying times, and going to various sub menus to set date and time, running a real-time clock, and ringing the alarm when the alarm time has arrived respectively. After discussing about the subcomponents, the most preferred and comfortable subcomponent was selected by each of the four members in the group to develop.

AA	Keypad Input
CA	LCD Display
AG	RTC Module
LS	Alarm Audio

▪ **<u>Development in the Arduino Environment</u>**

As we were given instructions to have an ATmega328P microcontroller in our final product, the development of our selected subcomponents was done using Arduino Uno boards, which also has the same microcontroller. For circuit simulations "Proteus 8 Professional" software was used as an offline tool, and www.tinkercad.com website was used as an online tool. For coding, Arduino IDE 1.8.15 was used in the offline method and Thinkercad had its own coding environment for the online method.

1. <u>Keypad</u>

A standard 3x4 matrix keypad was used by AA to give the inputs. The pressed number was represented using 4 LED bulbs, in the form of binary numbers.

Listed below is how the LEDs were lightened when the corresponding button was pressed. "0" and "1" represents LED off and on respectively.

0	0000	6	0110
1	0001	7	0111
2	0010	8	1000
3	0011	9	1001
4	0100	*	1110
5	0101	#	1111

Seven 100 Ω resistors were connected on the 7 wires between the keypad and the Arduino board to avoid having any errors by pressing two or more buttons at the same time, which cannot be shown using the simulation tools. The inbuilt "keypad.h" library was used for coding.

2. <u>LCD</u>

For developing the menu display part, a 16x2 LCD display was used in 4-bit mode in the Arduino environment. "LiquidCrystal.h" and "Wire.h" inbuilt libraries were used for implementing the code. To control the menu and to go to different options, 6 buttons were used. They were used to perform main functions such as “select”, “up”, “down” and “reset”. Two other additional buttons were used to go “left” and “right” on the screen.

Initially, for switching from one screen to another, a variable called “Menu” was used, which has different integer values according to the screen. When the simulation starts, the LCD display shows a welcome screen giving a welcome message as “WELCOME MENU”. Then it displays the main menu options, “Date & Time”, “Set Date & Time”, “Set alarm” and “Alarm ringtone”. Every option has sub menus that can be accessed by using the “select” button. According to the “Menu” values, there are sub menus that were defined as “Actions”. The left and right buttons are only accessible under “Action2” for setting date and time. Additionally, a DS1307 s real time clock module was added for the demonstration purposes.

When considering the LCD display, it contains 16 pins. These names of the pins and their functionality are as follows.

- VSS - Ground pin connected to system ground
- VDD - Supply the power to the LCD (4.7V - 5.3V)
- VEE - Decides the contrast level of the display
- RS (Register select) - Connected to the chip to shift between registers.
- RW (Register write) - Used to write/read data on the registers.
- E - (Enable) Enabling the LCD and this pin toggled between 1 and 0
- D0 to D7 - Data pins

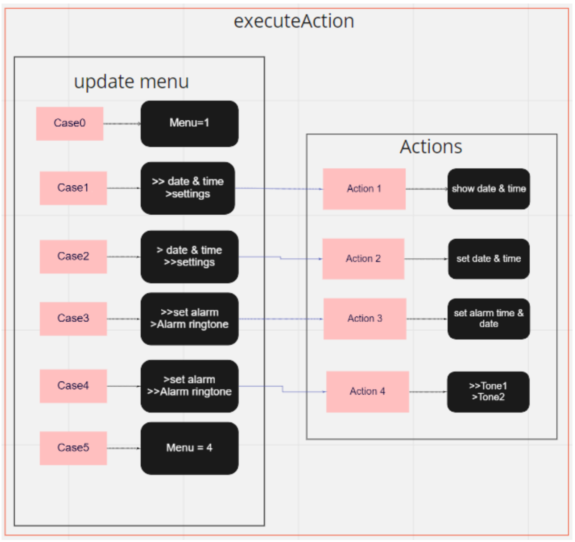

Figure 01 – LCD Menu Structure

3. RTC

In the subcomponent development phase, the Real Time Clock (RTC) part was demonstrated by AG using an Arduino Uno board, DS1307 RTC, LED, switch and a serial monitor via Proteus Simulation. “Wire.h” and “RTClib.h” are the main libraries used in the demonstration process. Since RTC is an I2C communication device, Wire.h library is used to communicate with RTC and RTClib.h is used to set and get date and time from the RTC.

Since the LCD part was being developed as another subcomponent, a serial monitor was used to display the time and date. A simple if condition, (alarm hour==current hour) && (alarm minute==current minute) was used to check the alarm condition and to illuminate the

LED. The glowing LED represents the alarm being active during the simulation process. A resistor of 220Ω was used to protect the LED from high currents. The alarm command is sent to the LED as a continuous signal and a switch is used to represent the alarm being turned on or off for the specific alarm time setting. During the demonstration process only one alarm was set.

4. Alarm

In the beginning there were two methods to ring the alarm. Using frequency notes and sending them to the speaker as electric signals (1 or 0) was one of them. And the other was to use a micro SD to store and play a simple .WAV file. In the Arduino phase both methods were created successfully by LS. In the first method Arduino's inbuilt "Tone()" and "noTone()" functions were used to generate the sound for corresponding frequencies. The musical notations of the melodies were converted into frequency with durations and stored in an array. Also, in this method interrupts were used for play/pause buttons.

In the second method an SD module was used to attach an SD card. For coding, Arduino's inbuilt "TRMpcm.h" library was used to handle the .WAV file reading. For both cases the same speaker (8Ω, 2W) was used with an NPN transistor, even though a simple buzzer circuit was enough for the first method.

▪ Development in the AVR Environment

After successfully developing the subproducts on the Arduino environment, our next aim was to build the same subcomponents using the ATmega328P, without the help of the Arduino board. As the ATmega328P microcontroller was not available in Thinkercad, the simulations were done from Proteus 8 Professional. Programming was done in AVR C++ language. "Microchip Studios for AVR" was used as the offline software tool for programming. The SPDIL28 PCB package of the microcontroller which has the 14x2 pin configuration in Proteus was used for the demonstrations. Moving from Arduino environment to AVR environment was quite challenging for all four of us as we didn't have any prior knowledge on that area. Many resources from the internet (videos, tutorials, datasheets, etc.) were used to gain insights and knowledge. We were instructed to design a simple PCB, as well as an enclosure for the subcomponents we developed individually. Designing of the PCB was done using "Altium Designer 20.0.2" as an offline tool, and www.easyeda.com as an online tool. Enclosure designs were made using hand sketches, as well as "SOLIDWORKS 2020".

1. <u>Keypad</u>

As in the previous stage, AA used the same 3x4 matrix keypad as the input, and the 4 LED lights to show the output. The logic he used was to identify the row and the column using while loops when a number is pressed, so it can be represented using LEDs as in the first development phase. Library creation was not done on this stage as the code was not that complex.

2. <u>LCD</u>

In the AVR environment CA had to implement and go for a more complex code format. When switching from the Arduino environment to the AVR environment, many new partitions had to be added. As the previous inbuilt libraries in the Arduino environment were not possible to use, "<avr/io.h>" and "<util/delay.h>" were used in the AVR environment.

Even though the environments are different, the outcomes of both codes are quite the same. The only difference was that AVR code had more functions. Here, CA used more menus, sub-menus and actions and sub-actions accordingly. CA also used some common hexadecimal value commands for the LCD display.

- 0x38 - Initialization of 16*2 LCD in 8 bit mode having 2 lines
- 0x28 - Initialization of 16*2 LCD in 4 bit mode having 2 lines
- 0x0C/0x0E - Clear the screen
- 0x80 - Cursor on the first row (at home position)
- 0xC0 - Cursor on the second row

The LCD pins work the same as the Arduino environment. Initially CA went with 8-bit mode which uses all the 8 data pins. But the 4-bit mode only uses 4 data pins. Using the I2C (Inter Integrated Circuit) module is much more effective because only 2 pins will be enough for the LCD processing. As the RTC module also uses the I2C communication, for the development of the final product to be less complicated, CA decided to use the 4-bit mode to design the menu display subcomponent. To convert it from 8-bit mode to 4-bit mode, CA had to add a few more codes.

3. <u>RTC</u>

Compared to the previous Arduino development stage, the AVR Environment was a bit more challenging to develop. The circuit and concept used by AG is fairly simple to demonstrate RTC in the AVR environment. Other than the microcontroller, a DS1307 RTC module, a push button and an LED were used in the proteus simulation. Two libraries named TWI.h and rtc.h were created by AG to implement the AVR demonstration of RTC. The 1st problem and the misconception that created the challenge was thinking we can connect the RTC to the chip and the data transfer will be done automatically.

To overcome this challenge, TWI.h library was created to establish the connection and data transfer between RTC and the microcontroller. TWI is a two-way data transfer consisting of 2 data lines SDA and SCL. SDA and SCL stands for Serial Data transfer and Serial clock frequency respectively. The SDA pin of the microcontroller and SDA pin of DS1307 are connected to each other. The SCL pin of the microcontroller and SCL pin of DS1307 are connected to each other.

TWI data transfer consists of two entities: A master (Microcontroller) and a slave (DS1307 RTC). and the command transfer follows a command and acknowledgement format. TWI.h library contains the functions as follows;

- TWI_Initial() - Initializing TWI and setting the bit rate
- TWI_Start() - Master sends the TWI start condition to slave
- TWI_Slave_Write() - Master sends a write signal to slave
- TWI_Slave_Read() - Master sends a read request to slave
- TWI_Write() - Master sends the date that to be written on slave
- TWI_Stop() - Stops the TWI transmission
- TWI_ReadMore() - Master reads one byte data from slave and requests more data
- TWI_ReadStop() - Master reads one byte of data from slave and send a stop condition

Last two functions were defined to make the coding of the RTC library easier. In addition to this the TWI.h had two more functions: BCD to DEC converter and DEC to BCD converter. Since Binary Coded Decimal is used for RTC communication and when displaying time we use decimal values, these functions were used to make the coding process easier. AG created the rtc.h library containing the usual RTC related functions like Set Date and Get Date.

When doing the simulation, the serial monitor was not used like the previous time as it required a different set of initialization and data transmission conditions and libraries. The inbuilt serial monitor of proteus was used to demonstrate the rtc.h part. In addition to the alarm switch of the button from the previous Arduino environment product, a push button was also included for instantaneous stopping of the alarm in the demonstration. A simple if condition checks the alarm condition and illuminates the LED.

During the subcomponent development in the AVR environment, a 16MHz external crystal oscillator was used for the microchip and a 32.768kHZ external crystal oscillator was used for the DS1307.

4. Alarm

Since the final aim was to develop a simple melody player and not a .WAV player LS decided to go with the first method that was used in the Arduino phase. Compared to the Arduino environment, coding was more difficult in the AVR environment since there are no user-friendly functions like Tone() and noTone() as the Arduino environment.

Here the main problem was that there was no inbuilt function to generate sound for a given frequency. So, a function needed to be defined in AVR. Using a simple technique of PWM, a function was created by LS, similar to the Arduino Tone() function which could generate a “Half Duty Cycle” (50% cycle) of electronic signals for a given frequency. As shown in figure 1.0.1.

Later another problem which LS had to solve was that the space of the SRAM of the microcontroller exceeded due to large arrays which were defined for melody notations. That problem was overcome by storing that data in the program memory (Flash Memory) by using “AVR GCC progmem attribute”. With the use of a simple speaker with a proper amplifier circuit, the melody player could be achieved with the feature of switching between melodies.

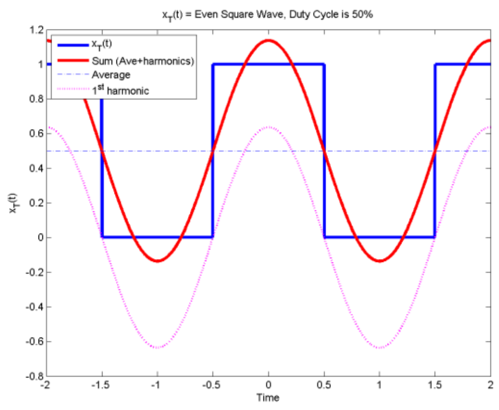

Figure 02 – Half Duty Cycle

Integrated melodies used are as follows.

- He’s a Pirate (from Pirates of the Caribbean)
- Game of Thrones Theme
- Pink Panther Theme
- Für Elise
- Tetris Soundtrack

▪ **Final Product Development**

The final stage of the project was to develop the complete digital alarm clock from the subcomponents we developed individually. Before starting to develop, we discussed how to use the pins of the ATmega328p microcontroller for each subcomponent, as there were limited and special pins which can be used only for special purposes (XTAL1, XTAL2, SDA & SCL pins). After deciding which subcomponent uses which pins, the final phase was divided into 3 sections by us.

1. Combining and completing the code
2. PCB designing
3. Enclosure designing

AG and CA mainly focused on developing and completing the coding while LS and AA focused on designing the PCB and the enclosure. As in the previous demonstrations, Microchip Studios for AVR and Proteus Professional 8 software tools were used for coding and simulation respectively. We were also advised only to use Altium Design for PCB designing and Solidworks for enclosure designing.

1. Combining and completing the code

First Attempt:

At the beginning, we had four distinct codes for each microproduct. We supposed it will be easy to combine all four codes. So, all code files were combined in one file using only 4 or 5 header files. That caused difficulties in debugging and making proper connections with each code part. Therefore, the main code was so long, and every main function was in it. We had to struggle a lot and had to spend more time on debugging.

Since we tried to combine these distinct codes without making many changes, the final full code occupied high memory in the SRAM (with many infinite while loops) that caused issues and errors with some parts of the final code. Then we were unable to achieve our expected outcome of the final product.

Second Attempt:

In the second attempt we planned to go in a way by first connecting input parts together and then moving into the processing and output parts. The order we connected the codes are as follows.

1. Connecting and setting up the LCD conditions
2. Implementing the keypad and getting input from keypad
3. Displaying the keypad input to the LCD display
4. Setting up the main menu
5. Configuring the Start, Back and navigation buttons (here we converted unused keys ”*” and ”#” in the keypad as left and right navigators)
6. Setting up the submenus
7. Implementing RTC libraries and time functions
8. Setting up alarm ring conditions, Alarm LED and Instantaneous Stop Button
9. Integrating the alarm tone functions
10. Final touch-ups

To make the code more readable, understandable and for easy debugging, multiple header files were added containing different parts of code such as LCD initializing condition, RTC initialization condition, menu options, song, and melody functions etc.

Except main.c, 19 other header files were used in our code as the header files mainly get saved in the ROM of the microcontroller.

2. PCB Designing

Since our final design is not a mechanically divided product, we decided to add all the components in one PCB and make it smaller as much as possible. The PCB was designed using the Altium Designer 20.0.2.

First the schematic diagram was drawn. PCB footprints with 3D bodies were downloaded for the ATmega328P and DS1307 RTC module from the internet initially. To have clear schematic diagrams, the pin placements of the schematic library files in both ATmega328P and DS1307 were edited according to our requirement. A 1x7 male header was used to connect the keypad while another two male headers (1x3 and 1x6) were used to connect the LCD display to the PCB. These headers were used as those components are not directly soldered on the PCB. Four test points were also placed on the PCB. Three of them were used to connect push buttons while the other one was used to connect all three buttons to the GND. Initially we connected all subcomponents using wires. After finalizing everything on the schematic diagram, net labels were used to make the schematic clearer.

After the completion of the schematic diagram, designing the PCB was started. First all components were placed in such a way that it minimizes routing distances and maximizes clearness of the routings. Some specific components had to be placed in specific positions to adhere to the basics of PCB manufacturing (placing the voltage regulator as far as possible from the microchip and oscillators) and in order to fulfill our design requirements (placing the LED light closer to the speaker opening of the enclosure so it will be visible well). 20mil trace width was used for the routing of power traces while 15mil trace width was used for the routing of signal traces. After placing the components and finishing the routings, a copper pour was placed on both top and bottom layers to connect all GND terminals. This is helpful to reduce the ground wire impedance, reduce voltage drop and stabilize the magnetic interference. We even considered and kept an empty space on the PCB to add our team logo.

Following are some of the circuits used in the schematic diagram and the PCB.

<u>Power Circuit</u>

- 1.5 AAA battery x4
- LM 7805 regulator (NCP1117)
- 0.33µF & 0.10µF polarized capacitors

Since a voltage of 5V is used by the active elements of the circuit we decided to get a stable 5V voltage using an LM7805 voltage regulator. As VCC should be quite larger than 5V and not too high, (as it affects the thermal equilibrium of the regulator) a voltage of 6V was chosen as the VCC.

<u>Speaker Circuit</u>

- Speaker (8Ω, 2W)
- NPN transistor (BC547)
- Diode-1N4001
- Resistor (220Ω)
- LED

Since a low power of 2W was consumed by the specific speaker, we decided to use it to output the sounds. An NPN transistor with a biasing resistor was added to amplify the output digital signal from ATmega328P to the speaker. Since the speaker is an inductive element, a freewheeling diode was added parallel to the speaker for the protection of the transistor. To make the product more attractive, an LED was used to blink when the alarm is ringing.

<u>RTC Module</u>

- DS 1307
- 32.768KHz crystal oscillator
- 1.5V coin cell battery

The RTC module is a main part of the circuit which keeps the time running. The 1.5V battery has been used in addition to the 5V main power, so that RTC can keep track of time, even when the main power is removed.

Input Method

For inputs we used a 3x4 keypad and three push buttons. Keypad can be used to change numerical things such as set date, time, and alarms. Also we assigned the unused two keys in the keypad (* and #) for “LEFT” and “RIGHT” cases. Other three push buttons are for “SELECT(MENU)”, “BACK” and instantaneous alarm stop purposes. The keypad input array was also changed a little bit so that it would give a good looking and a user-friendly enclosure.

3. Enclosure Designing

Before starting CAD, a rough sketch of how the final enclosure should look, was drawn by LS. We mainly targeted the user-friendliness of our product. The enclosure was designed on SOLIDWORKS 20. The dimensions of the PCB (98.55mm x 64.14mm) was measured first to get an idea on the size of our final product. The following components were used to design and assemble the final enclosure design

1. Main Body
2. PCB
3. LCD Display
4. Speaker
5. 3x4 Keypad
6. 3 Push Buttons
7. 3 Push Button Covers
8. Battery Socket
9. Battery Socket Cover
10. Rear Body

The main body was designed with the appropriate dimensions, extrude cuts, extrude bosses and fillets. Four small half spear shaped bosses were added on the bottom of the main body to improve the stability of the product and to minimize scratching of the bottom side of the main body. A step 3D file of the PCB 3D view was exported from Altium, and then it was converted as a part file using Solidworks. 3D bodies of components such as LCD display, speaker, 3x4 keypad, push button and battery socket were downloaded from www.3dcontentcentral.com and www.grabcad.com and were directly used. The push button covers were designed for the ease of pressing the buttons. The stop button cover was made much larger than the others as the user will be able to identify and switch off the alarm easily. Rear body of the product was to ensure the safety of the items inside the main body. An extrude cut was done at the place where the battery socket is. The battery socket cover was placed on that cut. Hole wizard was used for every place where we used holes for connecting the components to the main body. A semitransparent ring was added around the speaker to make the LED light visible to the outside.

After creating all the components, Solidworks Assembly was used to assemble all of them with appropriate mating of surfaces. After placing all the components in the correct position, textures and colors were added to the components to make the final product eye-catching. Finally, our team logo and the team name was traced on the face of our alarm clock.

RESULTS

The final version of our product is capable of performing the following tasks with a very user-friendly interface and an attractive enclosure.

1. View date and time
2. Set the date and time
3. Set alarms (up to 5)
4. Show existing alarms
5. Change the status or delete the alarm
6. Change alarm tone (5 tones)
7. Factory reset
8. About us

DISCUSSION

- **Pin configuration issue**

As mentioned above, there were some issues with pin configuration when combining all distinct codes of each microcontroller component. Same pins have been used for several components making it a huge problem to continue the process. So, we had to redefine the pin configuration for every component and change the code as well. In the Menu display, we initially used 8bit mode (used 8 pins in the ATmega328P chip). But moving to the final product using 8 pins (one port) for the LCD was not possible unless using the I2C module. Even though I2C needs only 2 pins (SDA, SCL) to connected with the chip, that was also not possible as the RTC module also had to use the same two pins for its communication with the chip. As a solution the 4-bit mode was used for LCD. Although there is a way to connect both components through the same pins, it was too complex, and we were unable to figure it out within the given duration.

- **Memory usage issue**

Mainly, we have used 5 melody songs for alarm ringtones. Those melodies have large files (large nodes arrays & large frequency arrays) that took much memory space. In addition to that, we had time storing, alarm storing and alarm status storing arrays as well. Usually when coding these arrays are stored in main memory. But in this case that made errors such as ‘exceeding data range’ and ‘data is not in the data range’. Instead of the usual way, we stored the larger melody arrays in the flash memory and linked them to main codes.

- **PCB designing**

As Altium’s inbuilt libraries didn't have the 3D bodies of the components such as resistors, capacitors, LEDs and headers, we had to download the PCB footprints and the 3D bodies of those components separately and add those 3D bodies in the PCB library files of the components. All the downloading was done using the www.snapeda.com website.

- **PCB Manufacturing**

Due to the ongoing pandemic situation and the limited time period, we were unable to physically manufacture the PCB.

REFERENCE

ATmega328P data sheet - https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Arduino Uno datasheet - https://datasheet.octopart.com/A000066-Arduino-datasheet-38879526.pdf

LM 7805 data sheet - https://components101.com/ics/7805-voltage-regulator-ic-pinout-datasheet

RTC data Sheet - https://datasheets.maximintegrated.com/en/ds/DS1307.pdf

LCD data sheet - https://circuitdigest.com/article/16x2-lcd-display-module-pinout-datasheet

APENDICES

1. Code - https://github.com/akilaabeyratne/Alarm-Clock

   This is our repository containing the following;

   i. Audino Environment Subproduct codes & Simulations
   ii. AVR Environment Subproducts codes, Simulations, PCB & Enclosures
   iii. Full Digital Alarm Clock Simulation
   iv. Digital Alarm Clock Instructions
   v. Final Product PCB & Schematics
   vi. Final Product Enclosure

2. Flow Chart

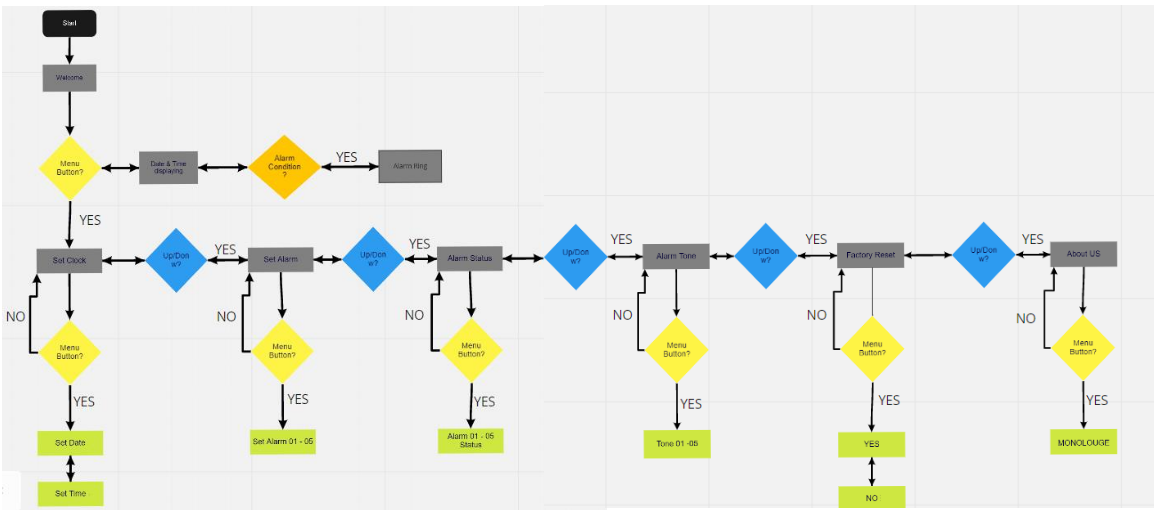

Figure 03 - Algorithm Diagram

3. Simulation

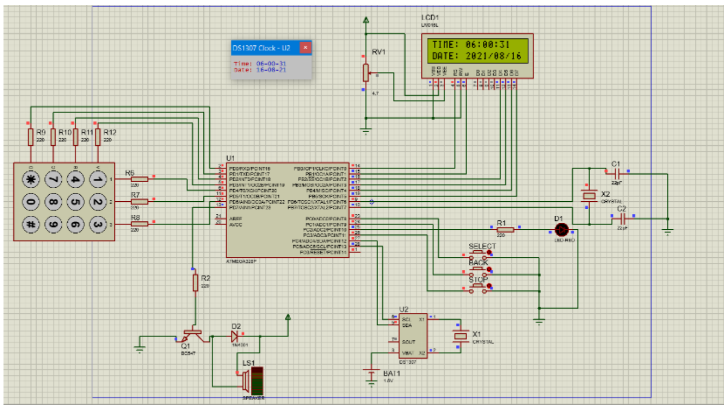

Figure 04 - Simulation when displaying date and time

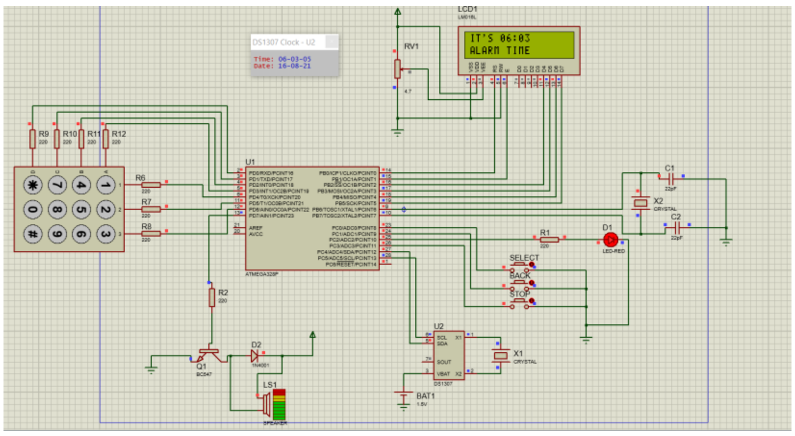

Figure 05 - Simulation when the alarm is ringing

4. Schematic Diagram

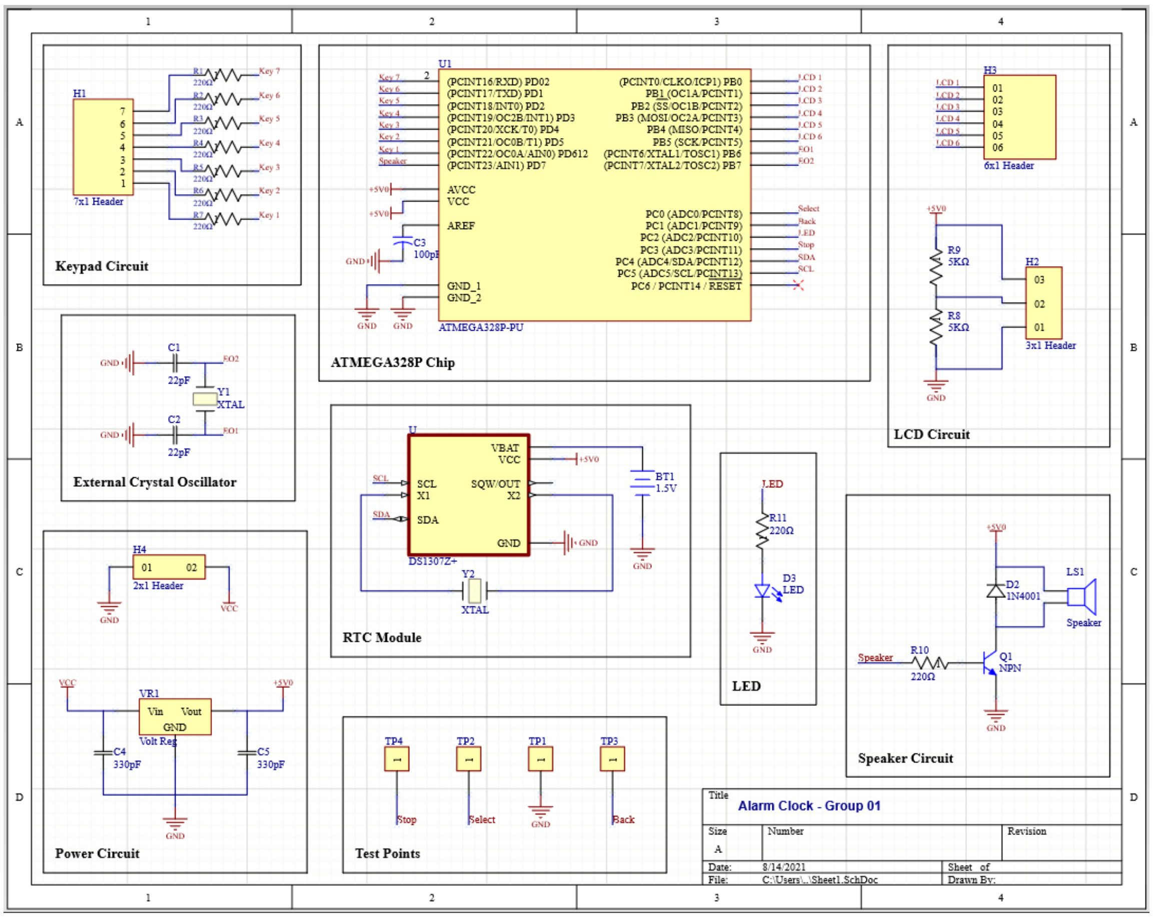

Figure 06 - Schematic Diagram of the PCB

5. PCB

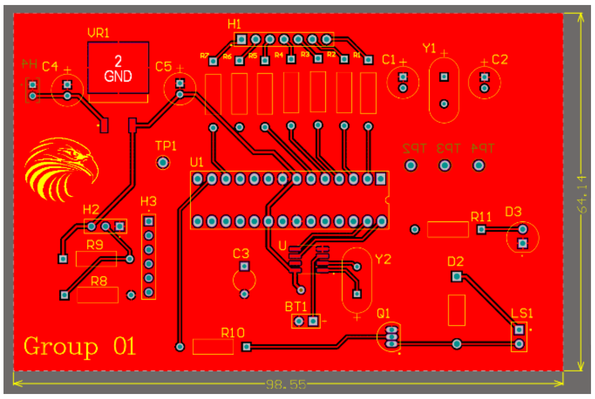
Figure 07 – PCB Top layer

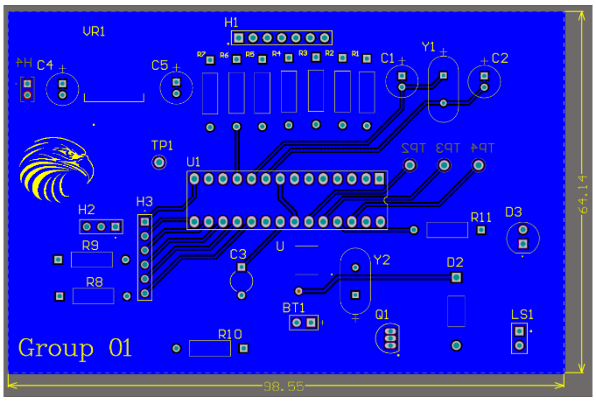
Figure 08 – PCB Bottom Layer

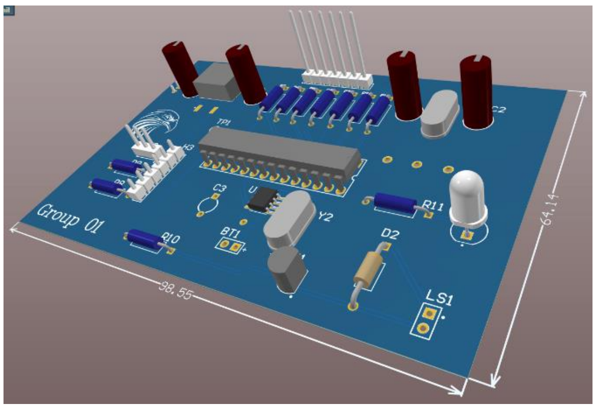
Figure 09 – PCB 3D View (Top)

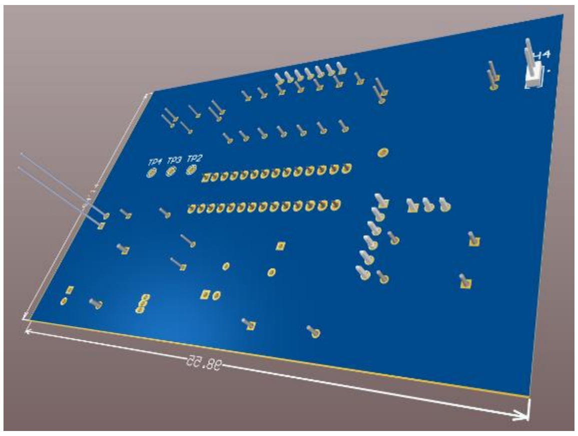
Figure 10 – PCB 3D View (Bottom)

6. Enclosure

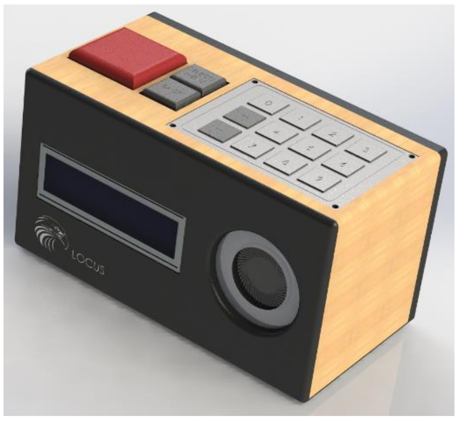
Figure 11 – Top and Front View

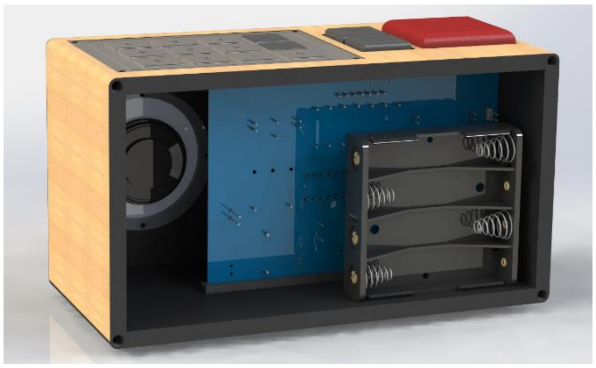
Figure 12 – Back view without the lid

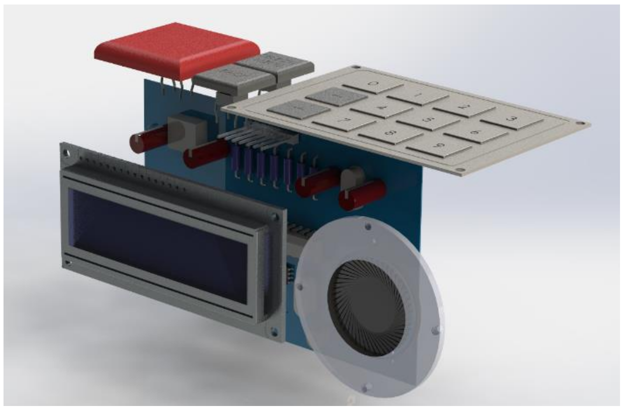

Figure 13 – View without the main body and rear body

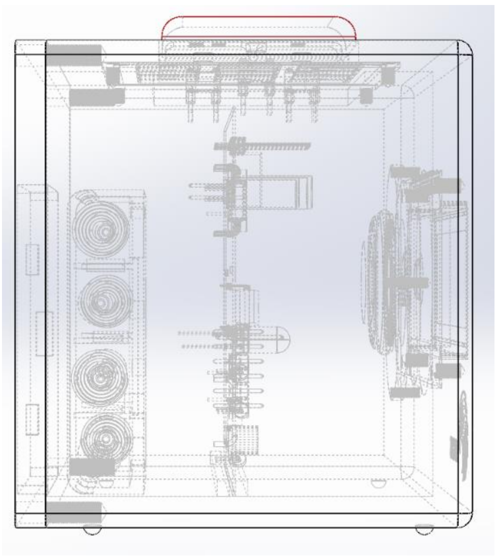

Figure 14 – see through side view

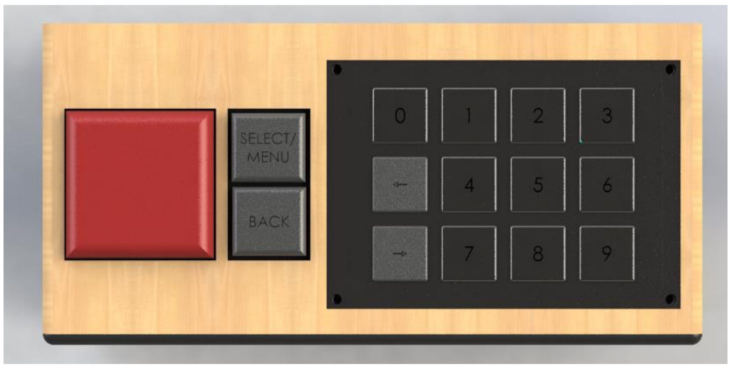

Figure15 – Top View

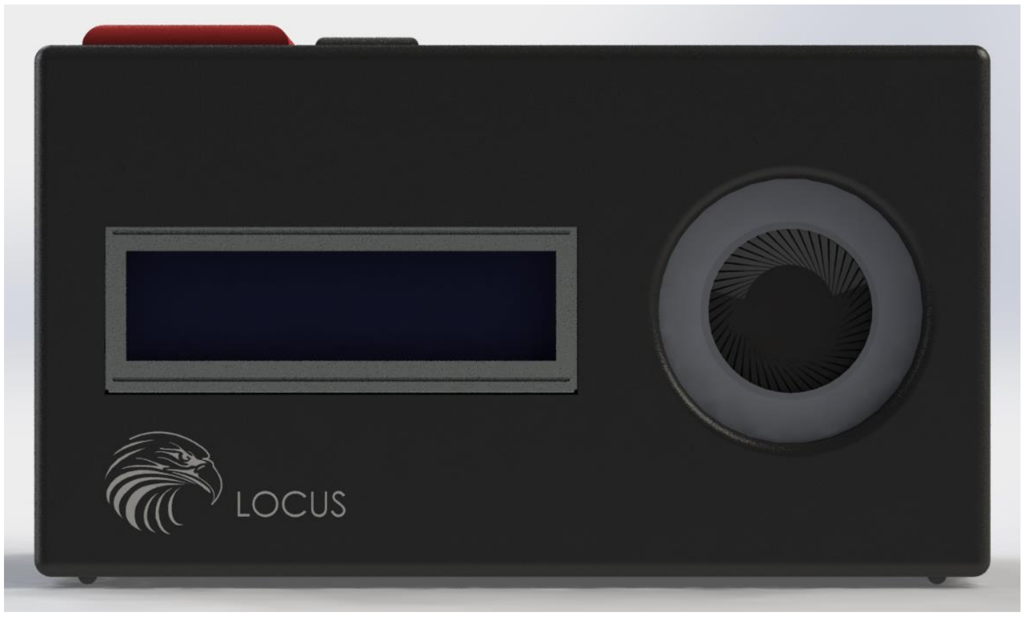

Figure 16 – Front View

7. Instructions for Use - https://bit.ly/Project_Locus