Project Report

Student Name: VIPRA BAGHEL
UID: 23MCI10265
Branch: MCA(AI/ML)
Section/Group: 23MAM3-B
Semester: 4th
Date of Performance: 20 Apr. 25
Subject Name: INTERNET OF THINGS
Subject Code: 23CAH-702

Aim: To design and develop an Arduino-based automatic bell system that rings a bell at pre-defined times using a real-time clock (RTC) and allows easy schedule adjustments through a keypad interface.

Objectives:

1. To create a reliable and programmable bell system using Arduino.
2. To enable users to set and update bell timings via a **4x4 matrix keypad**.
3. To display real-time clock and settings menu using a **16x2 LCD with I2C**.
4. To store the scheduled timings in EEPROM so that data is retained even after power off.

Components Required:

Sno	Name of Component	Qty.
1.	Arduino Uno R3	1
2.	DS3231 RTC Module	1
3.	16×2 LCD with I2C	1
4.	4×4 Matrix Keypad	1
5.	Buzzer	1
6.	Jumper Wires	12

Details of Components:

1. Arduino Uno R3:

The Arduino Uno R3 is the main microcontroller board used in this project. It is based on the ATmega328P chip and is responsible for controlling the entire functioning of the automatic bell system. It processes the input from the keypad, reads the current time from the RTC module, and triggers the buzzer or relay to ring the bell at the scheduled times. It is easy to program using the Arduino IDE and supports communication with other devices through digital and analog I/O pins.

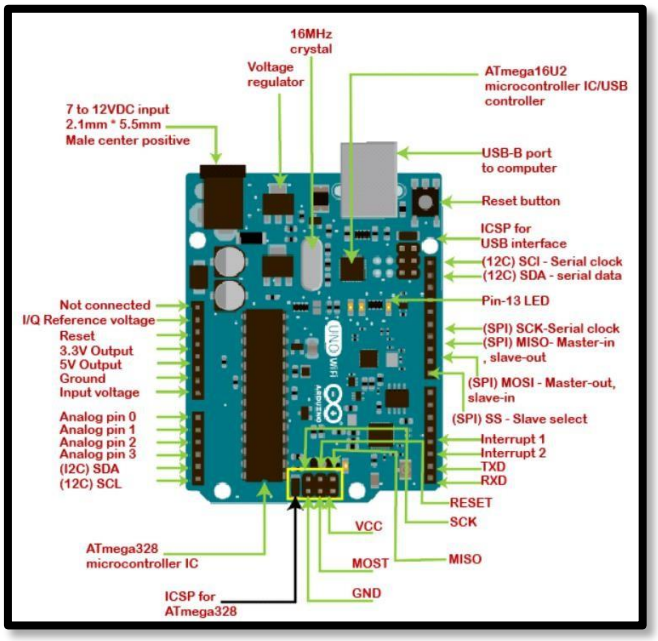

Figure 1 : Arduino Uno R3 Board

2. DS3231 RTC (Real-Time Clock) Module:

The DS3231 is a highly accurate real-time clock module that communicates with the Arduino via the I2C protocol. It contains an integrated temperature-compensated crystal oscillator that ensures precise timekeeping. This module has a battery backup, allowing it to keep track of the time even when the Arduino is turned off. In this project, it is used to maintain the current time and compare it with the scheduled bell timings.

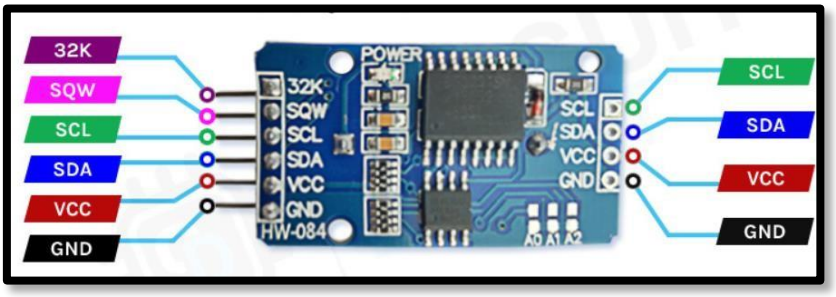

Figure 2: DS3231 RTC (Real-Time Clock) Module

3. 16×2 LCD Display with I2C Interface:

The 16x2 LCD display is used to show the current time, user menu, and confirmation messages. The I2C interface significantly reduces the number of pins required to connect the display to the Arduino (only two pins: SDA and SCL), making it efficient and space-saving. This display allows the user to see real-time updates and helps in the setup process through a user-friendly interface.

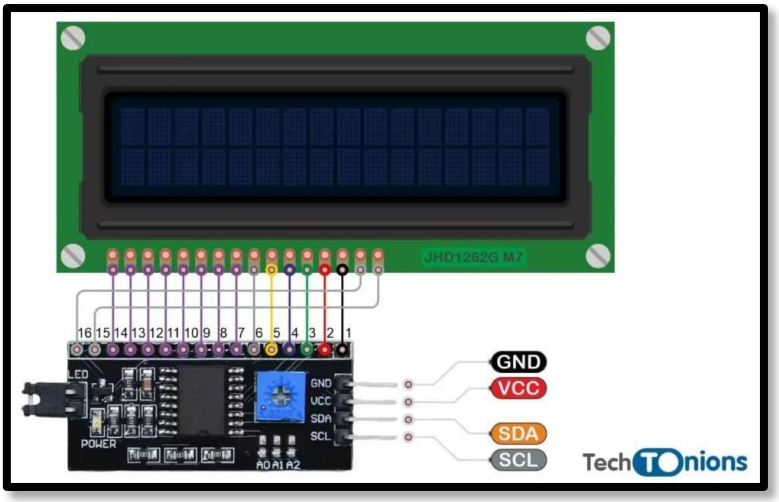

Figure 3: 16×2 LCD Display with I2C

4. 4×4 Matrix Keypad:

The 4x4 matrix keypad consists of 16 keys arranged in a 4-row and 4-column format. It allows the user to interact with the system by entering time, navigating the menu, or setting alarm times. Each key press is scanned by checking the row and column combination, making it ideal for numeric and functional inputs in microcontroller projects.

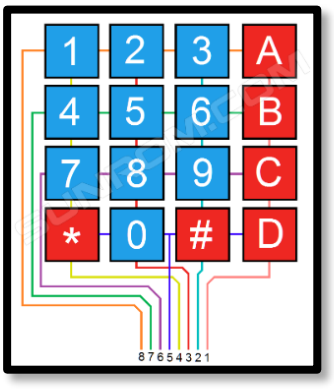

Figure 4: 4×4 Matrix Keypad

5. Buzzer:

A 5V piezo buzzer is used in this project as the alarm or bell. It produces a sound when triggered by the Arduino, indicating a scheduled event or class change. Although a relay and external bell can be used for actual school bells, the buzzer is suitable for testing or demonstration purposes.

Figure 5: Buzzer

6. Jumper Wires:

Jumper wires are used to connect the different components to the Arduino board. These are typically male-to-male or male-to-female wires that come in various lengths. In this project, they are essential for establishing electrical connections between the Arduino, RTC module, LCD, keypad, and buzzer.

Figure 6: Jumper Wires

Block Diagram of Designed Model:

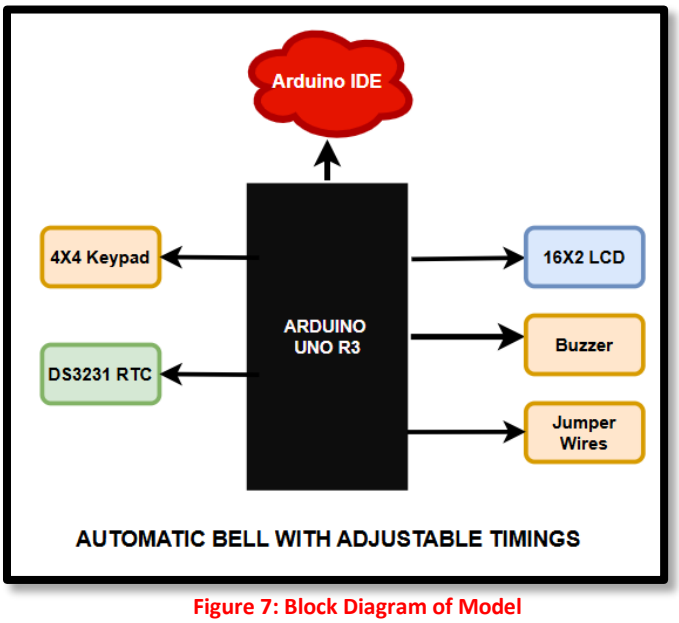

Figure 7: Block Diagram of Model

Working of Designed Model:

- Firstly, take an Arduino UNO R3 and 16X2 LCD with I2C module reduces the required connections from 12 pins to just 4.

LCD Pin (via I2C)	Arduino Uno Pin
VCC	5V
GND	GND
SDA	A4
SCL	A5

- The RTC provides precise time tracking and continues running even if Arduino is powered off (battery backup inside). The RTC sends the real-time date and time to Arduino, which is compared with stored bell timings.

RTC Pin	Arduino Uno Pin
VCC	5V
GND	GND
SDA	A4
SCL	A5

- Keypad allows the user to **enter time**, **set bell schedule**, **navigate menus**, etc.

Keypad Pin	Arduino Uno Pin
R0	D9
R1	D8
R2	D7
R3	D6
C0	D5
C1	D4
C2	D3
C3	D2

- **Key Functions:**
  * → Menu
  # → Confirm/save input
  B → Stop the Bell
  D → Backspace
  Alphanumeric keys → Enter timings, days, hours, etc.

- The buzzer is used to **emit a sound** for a specific duration when the bell time is triggered.

Buzzer Pin	Arduino Uno Pin
Positive (+)	A0
Negative (-)	GND

- When powered on LCD shows the current time (read from RTC). User can press * to open the menu.
  Menu options allow:

  Set Date/Time Set Number of Bells Set Bell Timings (HH:MM)

  When real-time clock matches any stored bell time: Arduino activates buzzer and bell rings

Pictures of Prototype:

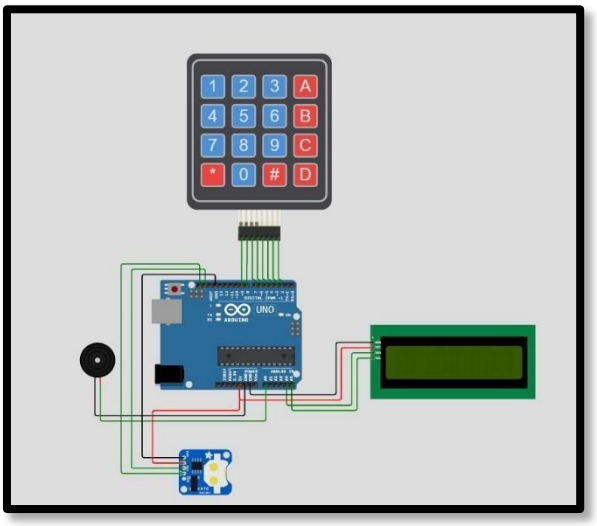

Figure 8: Circuit Diagram

Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <RTClib.h>
#include <EEPROM.h>
#include <Keypad.h>

#define buzzerPin A0
#define maxBells 10

RTC_DS3231 rtc;
LiquidCrystal_I2C lcd(0x27, 16, 2);

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'} // D = Backspace, # = Enter, B = Stop buzzer
};
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

struct BellTime {
  byte hour;
  byte minute;
};
BellTime bellSchedule[maxBells];
byte numBells = 0;
bool bellRinging = false;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  lcd.init();
  lcd.backlight();
  pinMode(buzzerPin, OUTPUT);

  if (!rtc.begin()) {
    lcd.print("RTC ERROR");
    while (1);
  }

  // Set RTC only once by uncommenting below, then re-upload with it commented
  //rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

  lcd.setCursor(0, 0);
  lcd.print("CLG BELL SYSTEM");
  delay(2000);
  lcd.clear();
```

```
  loadBellSchedule();
}

void loop() {
  DateTime now = rtc.now();
  lcd.setCursor(0, 0);
  lcd.print(now.year());
  lcd.print("/");
  lcd.print(now.month());
  lcd.print("/");
  lcd.print(now.day());

  lcd.setCursor(0, 1);
  lcd.print(now.hour());
  lcd.print(":");
  lcd.print(now.minute());
  lcd.print(":");
  lcd.print(now.second());

  if (!bellRinging) {
    for (byte i = 0; i < numBells; i++) {
      if (bellSchedule[i].hour == now.hour() && bellSchedule[i].minute ==
now.minute() && now.second() == 0) {
        ringBell(i + 1); // Pass bell number (1st bell, 2nd, etc.)
      }
    }
  }

  char key = keypad.getKey();
  if (key == '*') {
    menu();
  }

  delay(1000);
}

void ringBell(byte bellNumber) {
  bellRinging = true;
  for (byte i = 0; i < bellNumber; i++) {
    digitalWrite(buzzerPin, HIGH);
    delay(1000);
    digitalWrite(buzzerPin, LOW);

    // Allow STOP by pressing 'B'
    unsigned long waitStart = millis();
    while (millis() - waitStart < 1000) {
      char key = keypad.getKey();
      if (key == 'B') {
        digitalWrite(buzzerPin, LOW);
        bellRinging = false;
        return;
      }
    }
```

```
  }
  bellRinging = false;
}

void menu() {
  lcd.clear();
  lcd.print("1:Set Time");
  lcd.setCursor(0, 1);
  lcd.print("2:Set Bells");
  delay(1000);
  char key = keypad.waitForKey();
  if (key == '1') {
    setTime();
  } else if (key == '2') {
    setBellSchedule();
  }
}

void setTime() {
  lcd.clear();
  lcd.print("Set Year:");
  int year = getNumber();
  lcd.clear();
  lcd.print("Set Month:");
  int month = getNumber();
  lcd.clear();
  lcd.print("Set Day:");
  int day = getNumber();
  lcd.clear();
  lcd.print("Set Hour:");
  int hour = getNumber();
  lcd.clear();
  lcd.print("Set Minute:");
  int minute = getNumber();
  lcd.clear();
  rtc.adjust(DateTime(year, month, day, hour, minute, 0));
}

void setBellSchedule() {
  lcd.clear();
  lcd.print("No of Bells:");
  numBells = getNumber();
  for (byte i = 0; i < numBells; i++) {
    lcd.clear();
    lcd.print("Bell "); lcd.print(i + 1);
    lcd.setCursor(0, 1);///cursorrrr
    lcd.print("Hour:");
    bellSchedule[i].hour = getNumber();
    lcd.clear();
    lcd.print("Minute:");
    bellSchedule[i].minute = getNumber();
    lcd.clear();
  }
```

```
  saveBellSchedule();
}

int getNumber() {
  String input = "";
  lcd.setCursor(6, 1);
  while (true) {
    char key = keypad.getKey();
    if (key) {
      if (key == '#') {
        break;
      } else if (key == 'D') { // Backspace
        if (input.length() > 0) {
          input.remove(input.length() - 1);
          lcd.setCursor(6, 1);
          lcd.print("    "); // Clear the whole 4-digit space
          lcd.setCursor(6, 1);
          lcd.print(input);
        }
      } else if (isdigit(key)) {
        if (input.length() < 4) {
          input += key;
          lcd.print(key);
        }
      }
    }
  }
  return input.toInt();
}

void saveBellSchedule() {
  EEPROM.write(0, numBells);
  for (byte i = 0; i < numBells; i++) {
    EEPROM.write(i * 2 + 1, bellSchedule[i].hour);
    EEPROM.write(i * 2 + 2, bellSchedule[i].minute);
  }
}

void loadBellSchedule() {
  numBells = EEPROM.read(0);
  for (byte i = 0; i < numBells; i++) {
    bellSchedule[i].hour = EEPROM.read(i * 2 + 1);
    bellSchedule[i].minute = EEPROM.read(i * 2 + 2);
  }
}
```

Output of Deigned Model/Prototype:

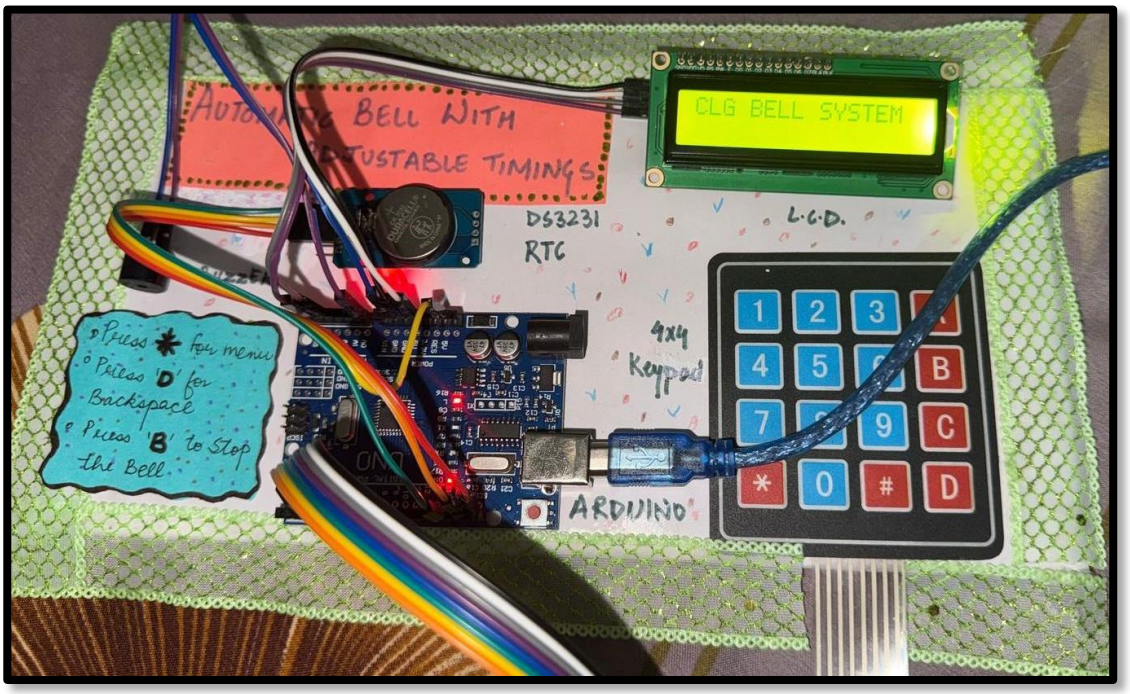

Figure 9: when system is connected to USB

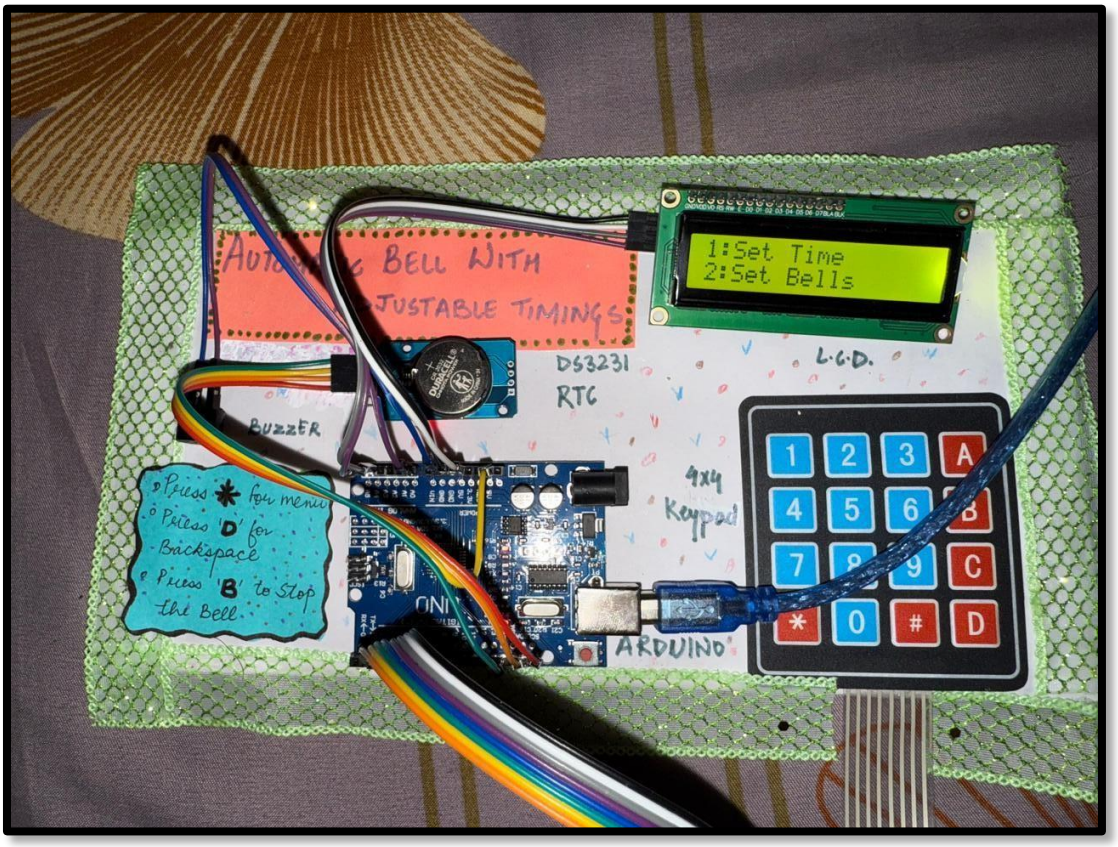

Figure 10: On pressing *, this menu appears

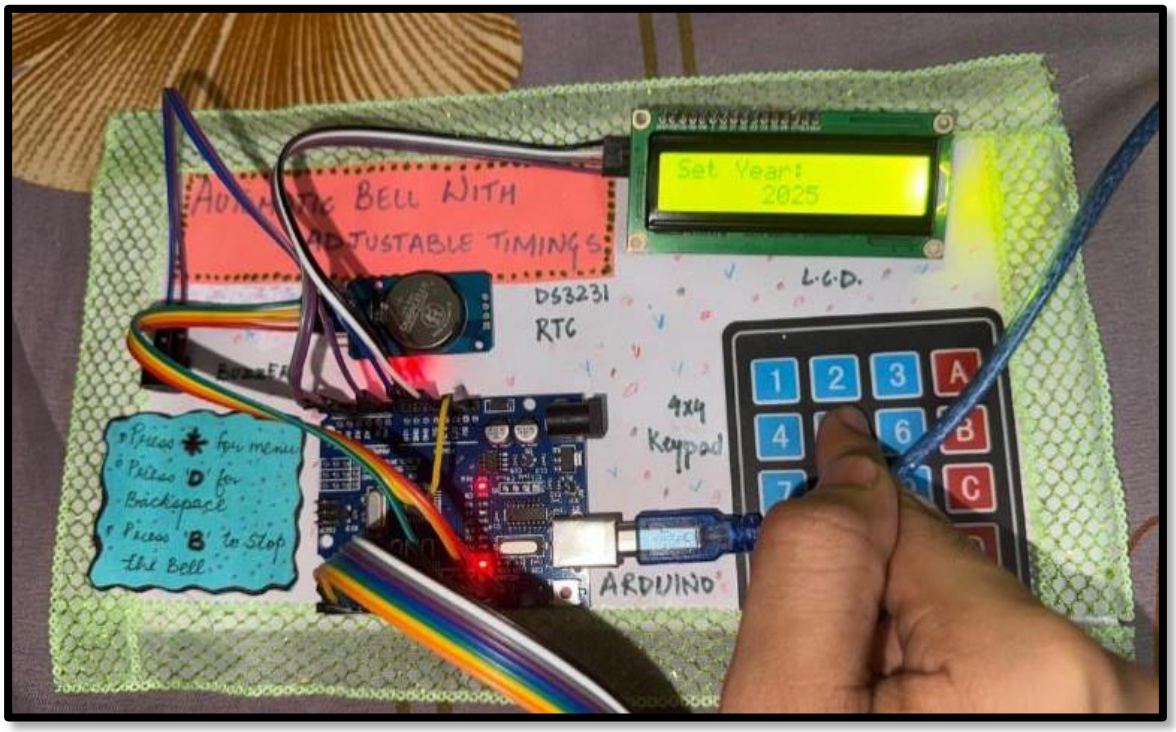

Figure 11: on pressing 1, setting of date and time is done via keypad

Figure 12: after entering details , press # to save the input

Figure 13: again press * , set bells

Figure 14: enter number of bells you want to buzz

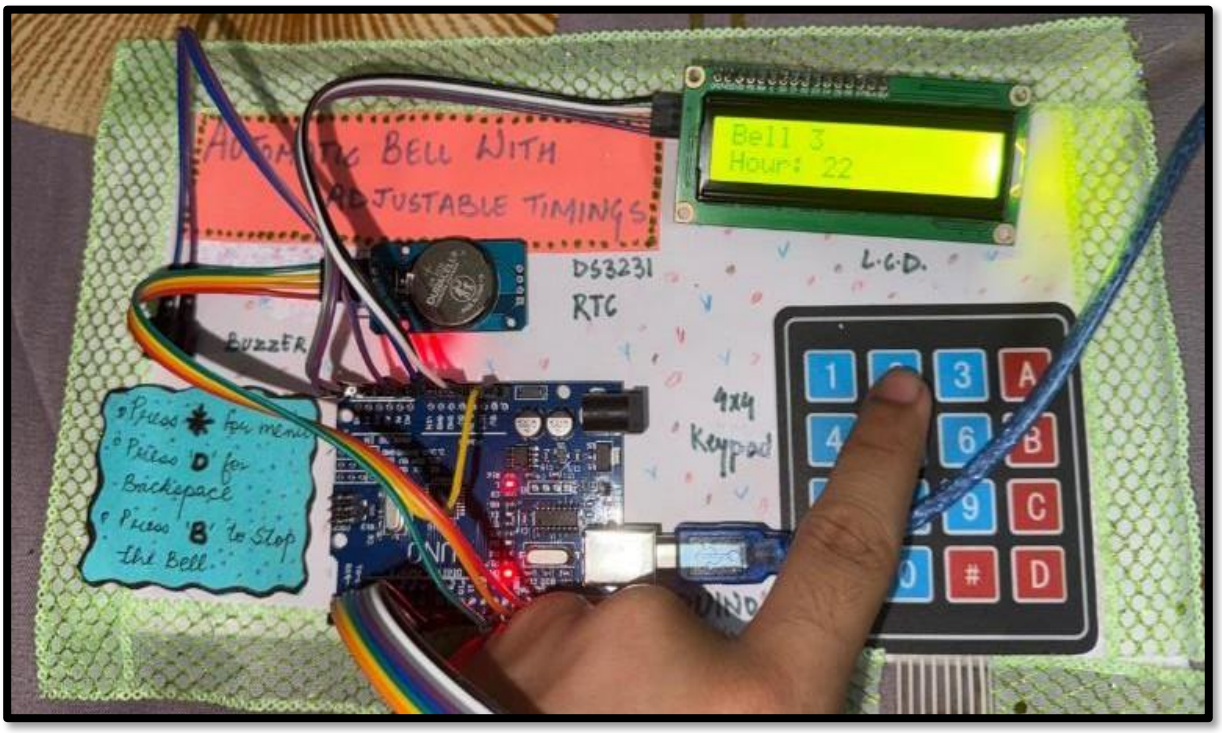

Figure 15: set the time at which you want the bell to buzz automatically

Learning outcomes (What I have learnt):

1. Gained hands-on experience in building an embedded system using Arduino and integrating various hardware components.
2. Understood how to use the DS3231 RTC module to accurately track and utilize time in automation projects.
3. Built a keypad-based menu system to allow real-time interaction with the device for setting time and bell schedules.
4. Understood how automation can replace manual tasks—like ringing school bells—enhancing punctuality and reliability.
5. Learned how to troubleshoot hardware and software issues during development and make the system robust and user-friendly.