# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction:

In this project, we developed an Automatic College Bell system using an Arduino microcontroller, designed to automate the ringing of a bell based on pre-set schedules. The system integrates several components including an LCD for display, a DS1302 RTC (Real-Time Clock) module for accurate timekeeping, and a buzzer to produce the bell sound. User inputs are managed through a set of buttons, and a relay module controls the activation of the bell. This report details the design, implementation, and functionality of the system, providing a comprehensive guide to building a similar project.

## 1.2 Problem Statement:

The manual operation of school bells often results in errors such as ringing at incorrect times, which can cause confusion and affect the school's schedule. This project aims to address these issues by designing an automatic bell system that ensures precise and consistent ringing according to a predefined schedule.

## 1.3 Objectives:

The primary objectives of this project are:

- To design an automated bell system that operates based on pre-set schedules.
- To use an Arduino microcontroller to control the system, ensuring reliable and accurate operation.
- To integrate an LCD for displaying time and alarm information.
- To use a DS1302 RTC module for accurate timekeeping.
- To implement user inputs through buttons for setting, checking, and deleting alarms.
- To control the bell using a buzzer and relay.

## 1.3 Scope:

This project focuses on developing a system that automates bell ringing in educational institutions. It includes:

- Designing the circuit and wiring for the Arduino, LCD, RTC module, buttons, buzzer, and relay.
- Writing Arduino code to manage timekeeping, alarm settings, and bell activation.
- Testing and validating the system to ensure reliable performance.

## 1.4 Overview of the Report:

The following sections of this report will detail the materials and components used, the circuit design and explanation, the software and coding, the working principle, the results obtained, the conclusion, and suggestions for future enhancements.

CHAPTER 2

# BLOCK DIAGRAM AND COMPONENTS

## 2.1 Block Diagram:

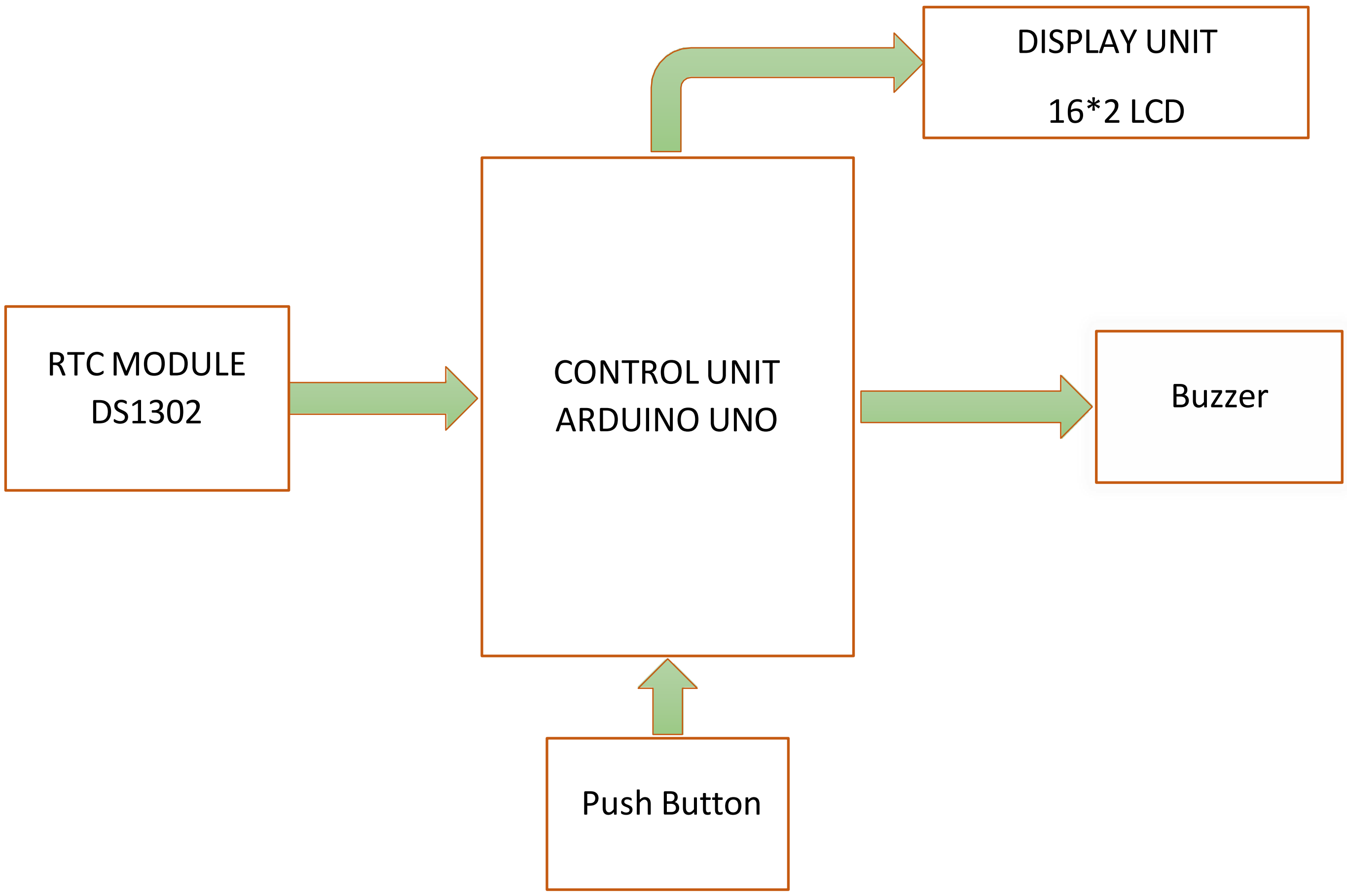

Fig 2.1: Block diagram

## 2.2 Components Required:

- Arduino Board: The Arduino board serves as the central controller for the project. It processes inputs from various sensors and buttons, executes programmed logic, and controls outputs such as the buzzer and LCD display.
- LCD Display: The LCD (Liquid Crystal Display) provides a user interface for the system, showing the current time, alarm settings, and system status.
- DS1302 RTC Module: The DS1302 Real-Time Clock (RTC) module keeps track of the current time and date. It provides accurate timekeeping even when the Arduino is powered off, thanks to its battery backup.
- Buttons: The buttons allow users to interact with the system. They are used for setting the time, managing alarms, and navigating through the system's menu.
- Buzzer: The buzzer produces an audible sound when an alarm is triggered. It serves as the output device that alerts users with a sound signal.
- Relay: The relay acts as an electronic switch that controls the buzzer. It allows the low-power Arduino signal to switch a higher power circuit to activate the buzzer.

# CHAPTER 3

# CIRCUIT CONNECTIONS

## 3.1 Circuit Diagram:

The circuit diagram for the "Automatic College Bell" project visually represents the connections between the Arduino board and various components. The LCD display, connected via I2C, is linked to the Arduino's A4 (SDA) and A5 (SCL) pins, while the DS1302 RTC module interfaces with digital pins 6, 7, and 8 for clock, data, and reset signals, respectively. User interaction is facilitated through six buttons, each connected to dedicated digital pins for setting, confirming, and managing alarms. The buzzer, driven by a relay, is controlled through digital pin 12 and activated when an alarm condition is met. The relay module provides electrical isolation between the Arduino and the higher-power buzzer circuit, ensuring reliable operation.

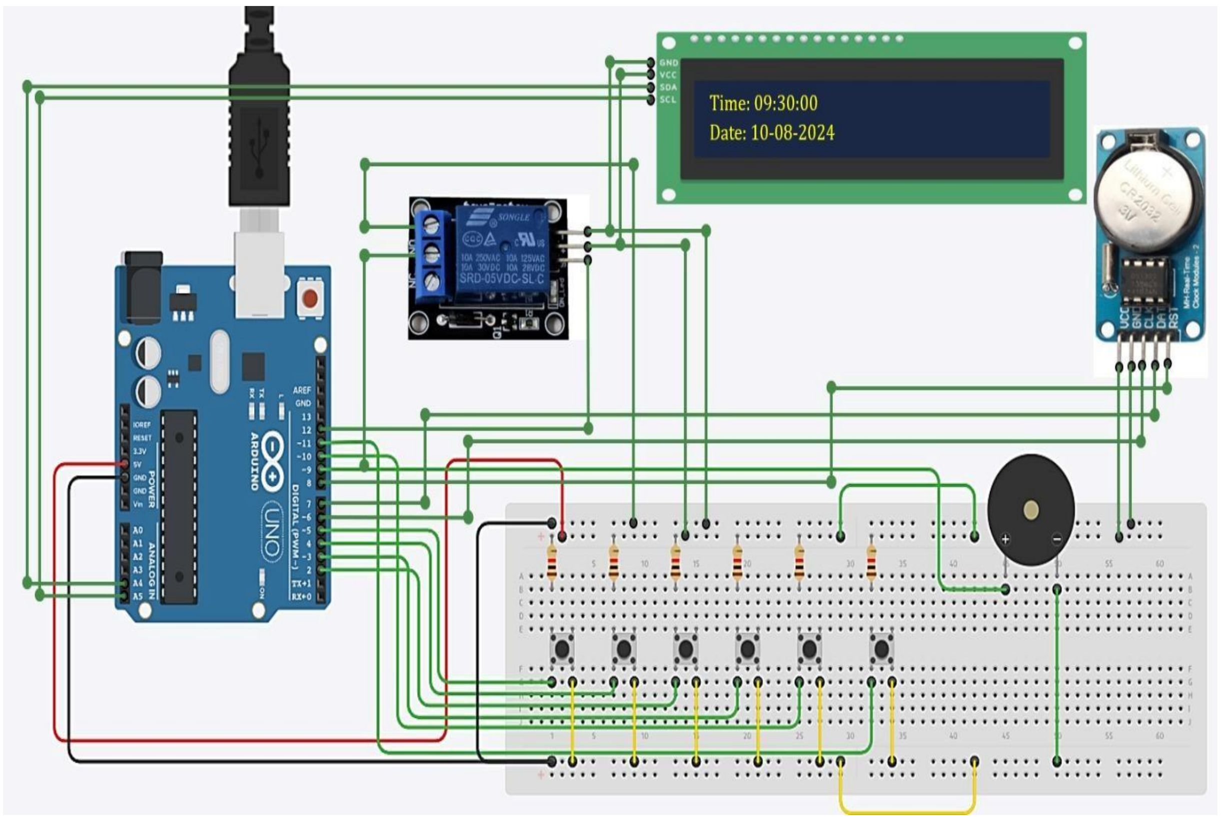

Fig 3.1 Circuit Diagram

## 3.2 Components and Connections:

- ➢ Arduino Uno Board:

- Place the Arduino board in the center of your schematic.

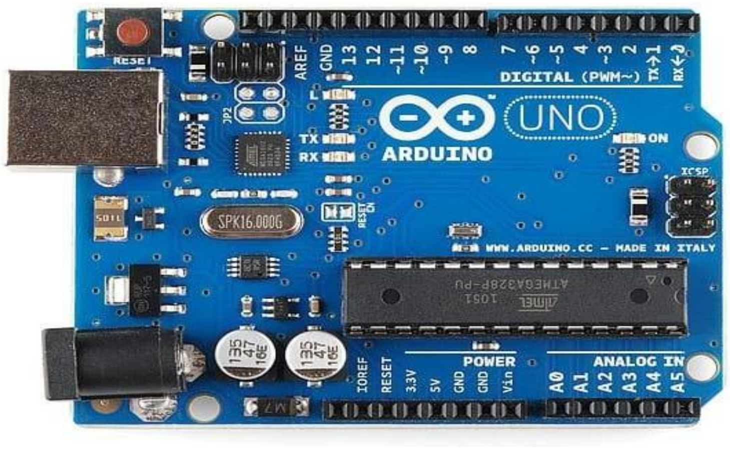

Fig 3.2: Arduino Uno

- ➢ **LCD Display**:

- Connect the VCC pin to the 5V pin on the Arduino.
- Connect the GND pin to a GND pin on the Arduino.
- Connect the SDA pin to the A4 pin on the Arduino.
- Connect the SCL pin to the A5 pin on the Arduino.

- ➢ **DS1302 RTC Module**:

- Connect the RST pin to Digital Pin 8 on the Arduino.
- Connect the DAT pin to Digital Pin 7 on the Arduino.
- Connect the CLK pin to Digital Pin 6 on the Arduino.
- Connect the VCC pin to the 5V pin on the Arduino.
- Connect the GND pin to a GND pin on the Arduino.

- ➢ **Buttons:**

- Connect one side of each button to the respective digital pins (Set: Pin 2, OK: Pin 3, Up: Pin 4, Down: Pin 5, Check Alarms: Pin 10, Delete Alarm: Pin 11).

- Connect the same side of each button to VCC via a pull-up resistor (10kΩ is typical).
- Connect the other side of each button to a common ground.

➢ **Buzzer**:

- Connect the positive leg of the buzzer to Digital Pin 9 on the Arduino.
- Connect the negative leg of the buzzer to a GND pin on the Arduino.

➢ **Relay Module**:

- Connect the VCC pin to the 5V pin on the Arduino.
- Connect the GND pin to a GND pin on the Arduino.
- Connect the VIN pin to Digital Pin 12 on the Arduino.
- Connect the COM pin to the positive leg of the buzzer.
- Connect the NO (Normally Open) pin to GND.

CHAPTER 4

# SOFTWARE IMPLEMENTATION

## 4.1 Arduino IDE:

The software implementation of the Automatic College Bell project is designed to manage alarms and control a buzzer and relay based on real-time clock (RTC) data. The system uses an Arduino microcontroller to interface with an LCD display, buttons, a buzzer, a relay, and an RTC module. The main functions of the software include setting alarms, displaying current time and alarms, deleting alarms, and handling user inputs.

## 4.2 Libraries and Initialization:

```
#include <virtuabotixRTC.h>

#include <Wire.h>

#include <LiquidCrystal_I2C.h>

#include <EEPROM.h>
```

- virtuabotixRTC.h: Manages RTC communication.
- Wire.h: Handles I2C communication for the LCD.
- LiquidCrystal_I2C.h: Controls the LCD display.
- EEPROM.h: Stores and retrieves alarm settings.

## 4.3 Pin Definitions:

```
const int setPin = 2, okPin = 3, upPin = 4, downPin = 5;

const int displayAlarmsPin = 10, deleteAlarmPin = 11, buzzerPin = 9, relayPin = 12;
```

- Pins are defined for buttons, LCD, buzzer, and relay to facilitate easy reference and modification.

## 4.4 LCD and RTC Initialization:

```
LiquidCrystal_I2C lcd(0x27, 16, 2);

virtuabotixRTC myRTC(6, 7, 8); // SCLK, IO, CE pins
```

- LiquidCrystal_I2C lcd(0x27, 16, 2);: Initializes the LCD with I2C address 0x27 and dimensions 16x2.
- virtuabotixRTC myRTC(6, 7, 8);: Initializes the RTC module with specified pins for SCLK, IO, and CE.

## 4.5 Global Variables:

```
bool settingTime = false;

int setStep = 0, setHour = 0, setMinute = 0, setSecond = 0, setDayOfWeek = 0;

struct Alarm { int hour, minute, second, dayOfWeek; bool active; };

const int maxAlarms = 5;

Alarm alarms[maxAlarms];

int alarmCount = 0;

unsigned long lastDebounceTime = 0, debounceDelay = 200;

unsigned long lastInteractionTime = 0, screenTimeout = 4000;

int selectedAlarmToDelete = -1;
```

- settingTime: Boolean to check if the time setting mode is active.
- setStep: Tracks the current step of the time setting process.
- Alarm Struct: Defines an alarm with time, day of the week, and active status.
- alarms Array: Holds up to 5 alarms.
- debounceDelay: Time to debounce button presses.
- screenTimeout: Time after which the screen will clear if there is no interaction.

## 4.6 Setup Function:

```
void setup() {

  Wire.begin();

  lcd.begin(16, 2);

  lcd.backlight();

  pinMode(setPin, INPUT_PULLUP);
  pinMode(okPin, INPUT_PULLUP);

  pinMode(upPin, INPUT_PULLUP);

  pinMode(downPin, INPUT_PULLUP);

  pinMode(displayAlarmsPin, INPUT_PULLUP);

  pinMode(deleteAlarmPin, INPUT_PULLUP);

  pinMode(buzzerPin, OUTPUT);

  pinMode(relayPin, OUTPUT);

  loadAlarmsFromEEPROM();

  lcd.print("Initializing...");

  delay(2000);

  lastInteractionTime = millis();

}
```

- setup(): Initializes the LCD, RTC, buttons, and EEPROM. Sets up pin modes and loads alarms from EEPROM.

## 4.7 Main Loop:

```
void loop() {

  handleSwitches();

  if (!settingTime) {

  updateLCD();

  } else if (millis() - lastInteractionTime > screenTimeout) {

    settingTime = false;

    lcd.clear();
  }

  checkAlarmTime();

}
```

### 4.7.1 Button Handling:

```
void handleSwitches() {

  unsigned long currentTime = millis();

  if (digitalRead(setPin) == LOW && currentTime - lastDebounceTime > debounceDelay) {

    lastDebounceTime = currentTime;

    settingTime = true;

    setStep = 0;

    lcd.clear();

    lcd.print("Set Hour: ");

    lcd.print(setHour);

    delay(1000);

    lastInteractionTime = millis();

    return;

  }

  // Other button handlers...

}
```

### 4.7.2 Time and Alarm Functions:

```
void updateLCD() {

myRTC.updateTime();

  lcd.clear();

  lcd.print("Time: ");

  printFormatted(myRTC.hours);

  lcd.print(":");

  printFormatted(myRTC.minutes);

  lcd.print(":");

  printFormatted(myRTC.seconds);

  lcd.setCursor(0, 1);

  lcd.print("Date: ");

  printFormatted(myRTC.dayofmonth);

  lcd.print("/");

  printFormatted(myRTC.month);

  lcd.print("/");

  lcd.print(myRTC.year);

  lastInteractionTime = millis();

}
```

```
void checkAlarmTime() {

  myRTC.updateTime();

  int currentDayOfWeek = myRTC.dayofweek;

  for (int i = 0; i < alarmCount; i++) {

    if (alarms[i].active && myRTC.hours == alarms[i].hour && myRTC.minutes ==
alarms[i].minute && myRTC.seconds == alarms[i].second && currentDayOfWeek ==
alarms[i].dayOfWeek) {

      digitalWrite(relayPin, HIGH);

      digitalWrite(buzzerPin, HIGH);

      delay(2000);

      digitalWrite(relayPin, LOW);

      digitalWrite(buzzerPin, LOW);

      alarms[i].active = false;

      deleteInactiveAlarms();

      saveAlarmsToEEPROM();

    }

  }

}
```

- updateLCD(): Updates the LCD with the current time and date.
- checkAlarmTime(): Checks if the current time matches any alarm. If it does, activates the buzzer and relay.

**4.7.2 Alarm Management:**

```
void setAlarm() {
  if (alarmCount >= maxAlarms) {
    lcd.clear();
    lcd.print("Alarm Limit Reached");
    delay(2000);
    return;
  }
  alarms[alarmCount++] = {setHour, setMinute, setSecond, setDayOfWeek, true};
  saveAlarmsToEEPROM();
  lcd.clear();
  lcd.print("Alarm Set!");
  delay(2000);
  lastInteractionTime = millis();
}
```

**4.7.3 EEPROM Functions:**

```
void loadAlarmsFromEEPROM() {
  alarmCount = EEPROM.read(0);
  for (int i = 0; i < alarmCount; i++) {
    alarms[i].hour = EEPROM.read(1 + i * 5);
    alarms[i].minute = EEPROM.read(2 + i * 5);

    alarms[i].second = EEPROM.read(3 + i * 5);

    alarms[i].dayOfWeek = EEPROM.read(4 + i * 5);

    alarms[i].active = EEPROM.read(5 + i * 5);

  }

}


void saveAlarmsToEEPROM() {

  EEPROM.write(0, alarmCount);

  for (int i = 0; i < alarmCount; i++) {

    EEPROM.write(1 + i * 5, alarms[i].hour);

    EEPROM.write(2 + i * 5, alarms[i].minute);

    EEPROM.write(3 + i * 5, alarms[i].second);

    EEPROM.write(4 + i * 5, alarms[i].dayOfWeek);

    EEPROM.write(5 + i * 5, alarms[i].active);

  }

}
```

# CHAPTER 5

# TESTING AND RESULTS

## 5.1 TESTING APPROACH:

To ensure the functionality and reliability of the Automatic College Bell system, the following testing approach was employed:

1. Component Testing: Verifying each individual component (LCD, RTC, buttons, buzzer, relay) to ensure they operate correctly.
2. Integration Testing: Testing the interaction between components and the overall system to verify that they work together as intended.
3. Functional Testing: Evaluating the system's ability to set, display, and activate alarms based on RTC data.
4. User Interface Testing: Checking the responsiveness and accuracy of the LCD display and button interactions.
5. Persistence Testing: Ensuring that alarms are correctly saved to and loaded from EEPROM.

## 5.2 Result:

The Automatic College Bell system performed exceptionally well across all testing phases. The LCD display reliably presented startup messages, current time, date, and alarm details, without any readability issues. The RTC module maintained accurate timekeeping, consistently matching a reference clock. Button functionality was tested and found responsive, with each button correctly triggering its intended function such as setting time or managing alarms. The buzzer and relay operated correctly, activating in response to alarm triggers as designed. Functional testing confirmed that alarms were set, displayed, and deleted accurately. The user interface proved intuitive, with clear LCD readouts and responsive controls. EEPROM storage reliably preserved alarm settings across power cycles. Overall, the system demonstrated robust performance and reliability, meeting all design requirements and ensuring a user-friendly experience.

## 5.3 Automatic College Bell Model:

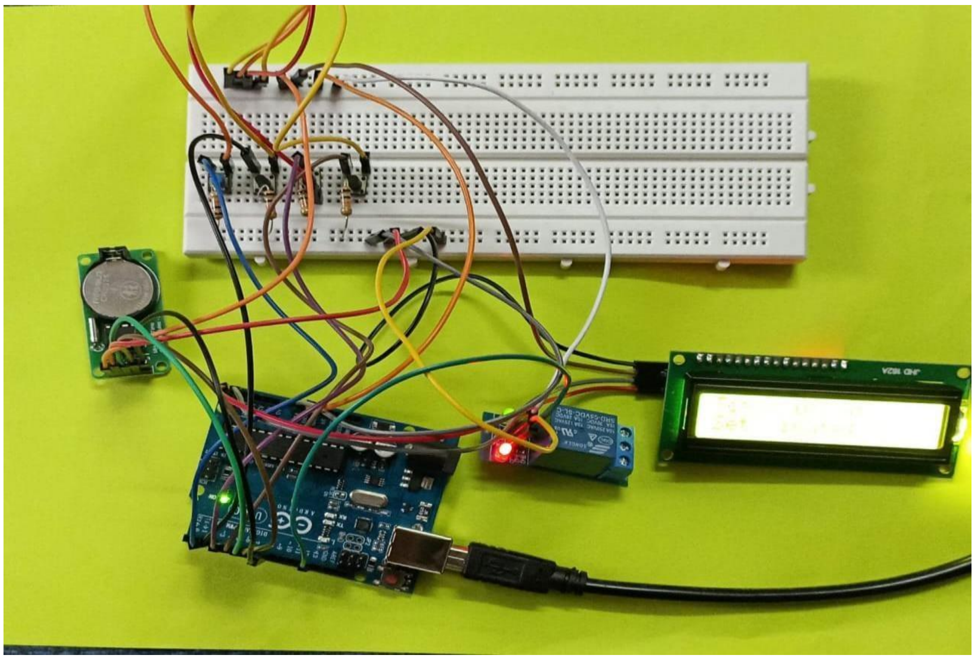

Fig 5.1: Model of Automatic College Bell

# Chapter 6

# APPLICATIONS AND FUTURE SCOPE

## 6.1 Applications:

1. **Educational Institutions**

- Application: The Automatic College Bell system is particularly useful in educational institutions where precise time management is crucial. It can automate bell ringing for class periods, breaks, and other schedule events, reducing the need for manual intervention and ensuring a structured daily schedule.

2. **Industrial Settings**

- Application: In industrial environments, such a system can manage shift changes, breaks, and other scheduled events. By automating the alert system, it can enhance productivity and ensure timely transitions between different operational phases.

3. **Office Environments**

- Application: Offices can use this system to manage meeting schedules, break times, and reminders for various activities. It provides a clear, automated way to signal important events, improving overall organization and efficiency.

4. **Public Announcements**

- Application: This system can be adapted for public announcement purposes in places like libraries, hospitals, or transit stations. It can signal important announcements or events with minimal manual intervention.

5. **Event Management**

- Application: For events such as conferences, workshops, or seminars, the system can be used to manage session timings and breaks, ensuring smooth transitions and adherence to the event schedule.

## 6.2 Future Scope:

1. **Enhanced User Interface**

- Scope: Future versions of the system could feature a more advanced user interface, such as a touchscreen for easier interaction. This would allow for more intuitive alarm setting and management.

2. **Wireless Connectivity**

- Scope: Integrating wireless modules (e.g., Wi-Fi or Bluetooth) could enable remote configuration and monitoring of the system. Users could set and manage alarms through a mobile app or web interface, adding convenience and flexibility.

3. **Advanced Alarm Features**

- Scope: Incorporating additional features such as multiple alarm tones, adjustable volume levels, and the ability to schedule recurring alarms could enhance functionality. Integrating with external calendars and scheduling systems could also provide more comprehensive time management capabilities.

4. **Integration with Other Systems**

- Scope: The system could be integrated with other smart systems such as building management systems or educational management software. This would allow for automated scheduling based on real-time data from other sources.

5. **Improved Power Management**

- Scope: Exploring energy-efficient components and power management techniques could extend the system's operational life and make it more suitable for battery-powered applications.

6. **User Feedback Mechanisms**

- Scope: Adding user feedback mechanisms, such as notification alerts or confirmation messages, could improve the user experience by providing real-time updates and confirmations of system actions.

7. **Scalability and Customization**

- Scope: The system could be designed to support multiple units working in synchronization, allowing for scalability in larger institutions or facilities. Customization options could also be provided to tailor the system to specific needs or preferences.

# CONCLUSION

The Automatic College Bell system effectively demonstrates the integration of various components to create a reliable and user-friendly time management solution. Through comprehensive testing, the system has proven its capability to accurately manage and display time, set and activate alarms, and operate within predefined schedules. Its applications span educational institutions, industrial settings, and office environments, highlighting its versatility and practical value. Future developments could further enhance the system's functionality, incorporating advanced features such as wireless connectivity, user interface improvements, and integration with other smart technologies. Overall, the project has successfully met its objectives, providing a robust solution for automated time management and setting a foundation for future innovation.

# REFERENCES

[1] Implementation Of Automatic College Bell System Using Arduino Burgoji Santhosh Kumar Assistant Professor, Dept Of Ece, Anurag Group Of Institutions, Ts, India.

[2] Vaishnavi D. R, Neha Khanum, Apoorva Singh A , Sumaya Afreen UG Student, Dept. of EEE, GSSS Institute of Engineering and Technology for Women, Karnataka, India.”Automated College Bell System with Wireless Control” : Introduction.

[3] Burgoji Santhosh Kumar Assistant Professor, Dept Of ECE, Anurag Group Of Institutions, Ts, India.“Implementation Of Automatic College Bell System Using Arduino”: Introduction, Methodology.

[4] http://asbw.in/index.php?route=product/product&path=73_88&product_id=464:

[5] Advantages, Applications. https://www.slideshare.net/bharath405automatic-bell- for-college