IT AND BUSINESS COLLEGE

Department: Computer Science

Project title: School bell

Team name: Team-3

Members:

Kyrmanchieva Aidai - 238715011

Tairova Kamilla - 238715023

Anarbekov Azimbek - 238711048

Bekmuhambetov Daniyar - 238715001

Instructor: Nurbekov Mirlan

Group: SCA-23A

Date: 17.12.25

Table of contents

List of figures

List of tables

Abstract

We implemented an Arduino-based automated school bell system that triggers scheduled bells to replace manual operation. The system addressed the problem of human error in marking class periods, aiming to ensure punctuality 16x2 and automate routine tasks. It used a software-based clock running on an Arduino UNO, with time displayed on an LCD screen. The system was initialized with a default schedule starting at 8:30 on Monday, featuring 40-minute lessons followed by 5-minute breaks and an extended lunch break from 12:10 to 13:00. When the internal time matched a scheduled period change, the Arduino activated a buzzer. Two buttons allowed manual time adjustment: one to advance the hour and another to advance the 5 minutes. In testing, the system reliably rang bells at all scheduled times and correctly handled the lunch break interval. Key results included precise timing for lesson and break durations and successful manual time correction via the interface. This project contributes to educational administration by providing a low-cost, self-contained scheduling solution that requires no external real-time clock module.

1.Introduction & Background

Time management is critical in educational institutions, where bells signal the beginning, intermission, and conclusion of classes. Traditionally, school bells were rung manually by staff or via simple timers, an approach that proved both error-prone and inflexible. An automated school bell system was developed to eliminate this manual labor by sounding alerts at precisely pre-programmed intervals. By ensuring that activities commenced and concluded punctually, the automated system promoted student discipline and fostered a structured learning environment.

The Arduino-Based Automatic Institute Bell Ringing System was designed to automate the process of ringing bells in educational institutions, such as schools, colleges, and universities. In conventional settings, bells had been manually controlled, requiring designated personnel to ensure they sounded at the correct times to signal the start and end of classes, breaks, and other scheduled events. This manual method frequently resulted in mistakes, delays, or even missed bells, which disrupted the daily schedule. The automation of this system eliminated the need for manual intervention, ensuring that bells rang at precise times according to a pre-programmed schedule, thereby enhancing the institution's time management. Through the implementation of an Arduino-based system, the institution gained significantly greater control over its daily operations.

.

1.1 History of different school bell variables

In previous works, considering automated timing and alert systems, it was found that Arduino-based controllers had been utilized for institutional scheduling and bell systems.
An Arduino-based automatic bell ringing system was proposed by Pavan (2020) (see Figure 1). In this work, the author developed a system where an Arduino microcontroller was used to control two main components: a real-time clock module and an electronic buzzer or bell relay to automate scheduled alerts. The function of the real-time clock was to maintain precise timing, while the buzzer or relay activated the bell according to a pre-programmed schedule.

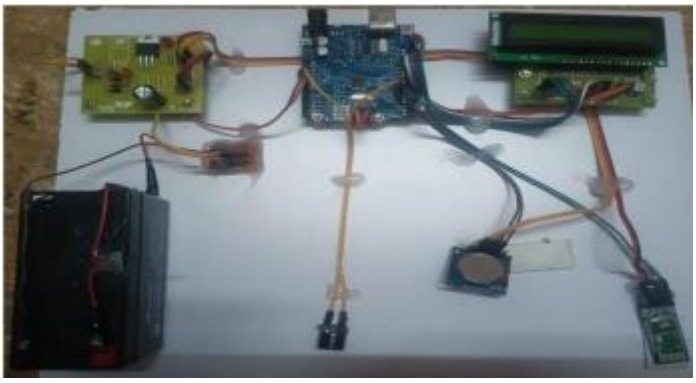


Figure 1. Prototype by Pavan and others

1.2 Types of school bell system

1.2.1 Manual bell system

Manual bell systems relied on a physical bell, typically a brass or iron cast bell installed in a schoolyard or tower (as shown in figure 2). A staff member, such as a janitor or administrator, manually rang the bell by pulling a rope or striking it with a hammer to signal the start and end of classes, breaks, and other periods. This traditional method was common in older institutions, particularly in rural or village settings, where it served as an acoustic landmark for the local community. Its operation depended entirely on human intervention, making it susceptible to delays, inconsistencies, and missed signals due to human error or absence.

Figure 2. Manual bell

1.2.2 Semi-automated bell system

Semi-automated bell systems utilized an electric bell or buzzer controlled by a manual push-button or a simple mechanical timer (as illustrated in figure 3). In this setup, an authorized person had to be present to press a button at the scheduled time or to activate a preset timer at the beginning of the day. This type represented an intermediary step between fully manual and fully automated systems, reducing but not eliminating the need for human involvement. While it offered more reliability than a purely manual system, it still carried a risk of operational failure if the responsible individual forgot to initiate the system or was unavailable.

Figure 3. Semi-automated bell system

1.2.3 Fully automated bell system

Fully automated bell systems employed an electronic controller, such as a programmable logic controller (PLC) or a microcontroller (Arduino), integrated with a real-time clock (RTC) module to trigger an electric bell or digital sounder (as presented in figure 4). The entire schedule was programmed in advance, and the system executed the alerts autonomously without any daily human intervention. This type provided the highest level of precision, reliability, and flexibility, allowing for complex schedules, multiple bell tones, and easy adjustments. It became the standard in modern educational institutions to ensure strict adherence to timetables and to foster a disciplined, structured learning environment.

Figure 4. Fully automated bell system

2.Problem statement and objectives

Schools need a system that rings bells exactly at class start and end times without human intervention. The main problem is to eliminate manual errors in bell scheduling and to automate alerts for lesson transitions. Our objectives were to:

1. Design an Arduino Uno-based bell system that rings a buzzer at a predefined timetable of lessons and breaks.

2. Display the current day, time, and lesson status on an LCD to inform users of the schedule.

3. Allow manual adjustment of the clock (hours and minutes) via pushbuttons, since no real-time clock (RTC) module is used.

4. Implement volume control for the buzzer via a 10K potentiometer.

5. Ensure the system is low-cost, runs on 5 V, and is safe (low current, no exposed high voltage).

In short, the system must automatically activate the bell at each lesson's start and end, show time/status on the display, and allow straightforward user control (simple buttons)
Non-functional constraints include a 5 V supply, low power consumption, and overall cost under typical student project budgets. Mechanically, the device must fit on a breadboard in class; electrically, it must use safe voltage levels (5 V logic and a 5 V buzzer)..

3.System requirements and constrains

3.1 Functional requirements

The bell system must ring a buzzer at the beginning and end of each lesson period (e.g. 08:30, 09:15, etc.) and at break times. It must display the current weekday and time on an LCD, along with an indication of the current lesson or break period. Two pushbuttons allow the user to increment the clock by 1 hour or by 5 minutes (and handle day rollover) to set the initial time. A potentiometer adjusts the buzzer volume. The bell rings for a fixed duration (2 seconds) at each trigger.

3.2 Constrains

3.2.1 Non-functional requirements

The design uses an Arduino Uno (operating at 16 MHz, 5 V) and readily available components. The system must run on a single 5 V supply (e.g. USB or battery). It should maintain timing accuracy for at least a school day; lacking an RTC module, the clock may drift over long periods (an unavoidable trade-off for simplicity). The response time for button presses and bell triggers should be essentially instantaneous; latency is limited by the millisecond timer and was found acceptable in tests. Cost is minimized by omitting wireless modules and using only one resistor (for the status LED).

3.2.2 Mechanical constraints and safety

The circuit is built on a small breadboard. No heavy moving parts or high voltages are involved (the buzzer is 5 V). The Arduino's digital pins can source up to 40 mA each, our design uses a pull-up resistor input (no external pull-down needed) and a 220 Om resistor for the indicator LED to stay within limits. The system is enclosed in a plastic box in final tests, and all wiring is insulated.

4. System design

The system works by taking input and calculating the moisture in the soil and when the value exceeds 440 it starts watering, you can also see the diagram in figure 5:

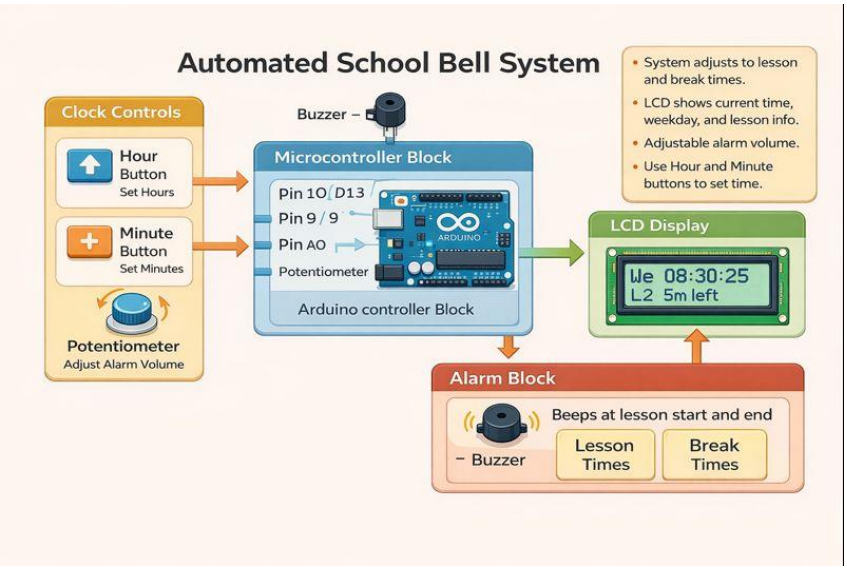


Figure 5. Automatic school bell system

4.1 Hardware design

Below is a table listing the components we used, and also includes their part numbers in the images to help illustrate them:

Component	**Purpose**	**Image Reference**
Arduino Uno R3	Central microcontroller	**Figure 6**
Buzzer	Sound reproducer	**Figure 7**
LCD 16x2	Displaying the time screen and calculating the time	**Figure 8**
10K-ohm potentiometer	Increase or decrease the volume	**Figure 9**
2 buttons	The first button rewinds time by an hour, the second by 5 minutes.	**Figure 10**
Power supply (power bank)	Provides stable power	**Figure 11**

Table 1. Components

Figure 6. Arduino UNO R3

Figure 7. Buzzer

Figure 8. LCD 16x2

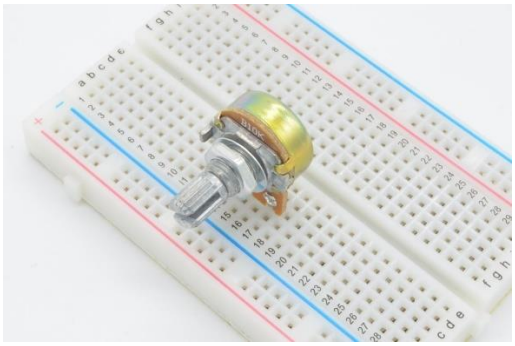


Figure 9. 10K-ohm potentiometer

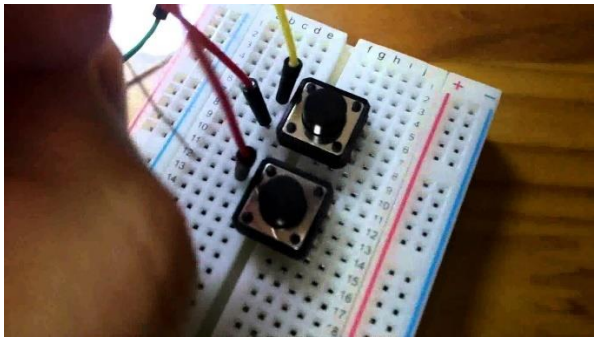


Figure 10. Buttons

Figure 11. Power supply

4.2 Construction materials

The design of the project was made of cardboard, where it completely imitates the basement of the Ala-Too college.

4.3 Software design

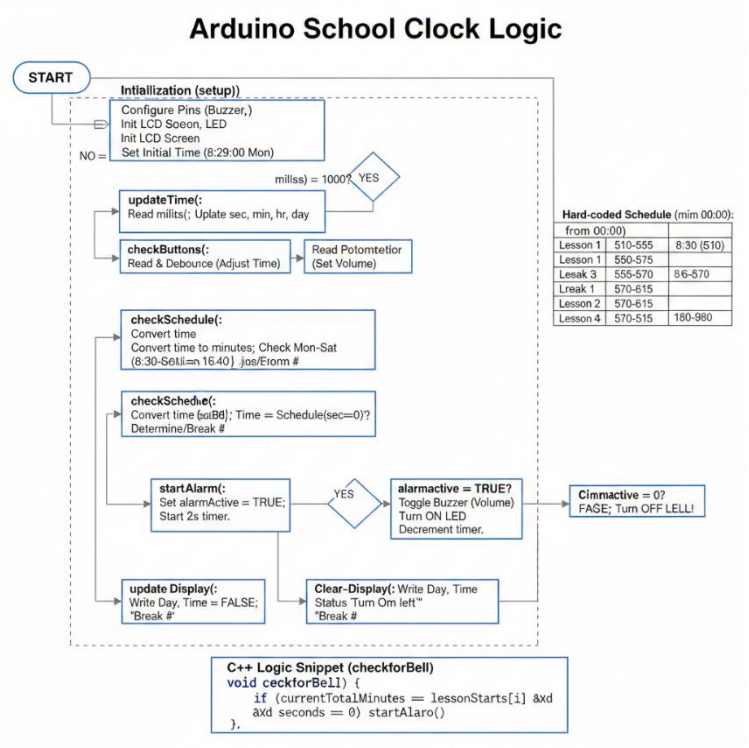


Figure 12. Flowchart for bell

1. Clock Update: A millisecond timer updates hours, minutes, seconds.

2. Input Processing: Read buttons with debounce to adjust time. Read potentiometer to set volume.

3. Schedule Logic: Determine if current time falls within school hours (Mon–Sat 8:30–16:40). If so, identify whether it is in a lesson or break by comparing to the hard-coded lesson start/end times.

4. Alarm Trigger: If the current time exactly equals a lesson start or end time (and the seconds count is zero), set an “alarmActive” flag.

5. Alert Action: While alarmActive (about 2 seconds), toggle the buzzer output with the set volume. Also turn on the indicator LED. After the duration, clear the alarm flag.

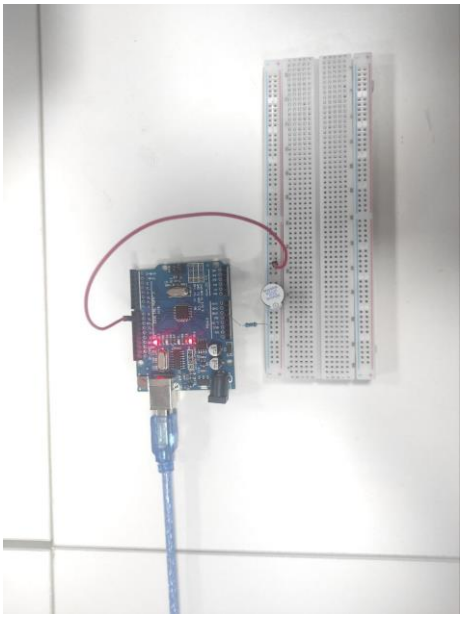
6. Display Update: Write the weekday, time, and status message (“L3 10m left”, “Break 2”, or “BELL!”) to the LCD.

Lesson	Start	End
1	08:30	09:10
2	09:15	09:55
3	10:00	10:40
4	10:45	11:25
5	11:30	12:10
6	13:00	13:40
7	13:45	14:25
8	14:30	15:10
9	15:15	15:55
10	16:00	16:40

Table 2. Schedule of lessons in our system

5.Prototypes versions and progress

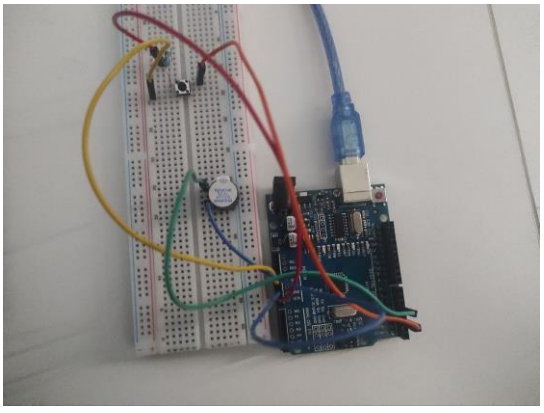
5.1 Prototype 0000



Figures 13. Prototype 0000

This prototype in figure 13 has a buzzer connected to it that makes a sound..

5.2 Prototype 0001



Figures 14. Prototype 0001

In the next prototype in figure 14, buttons were connected so that when pressed, the buzzer would make a sound.

5.4 Prototype 0002

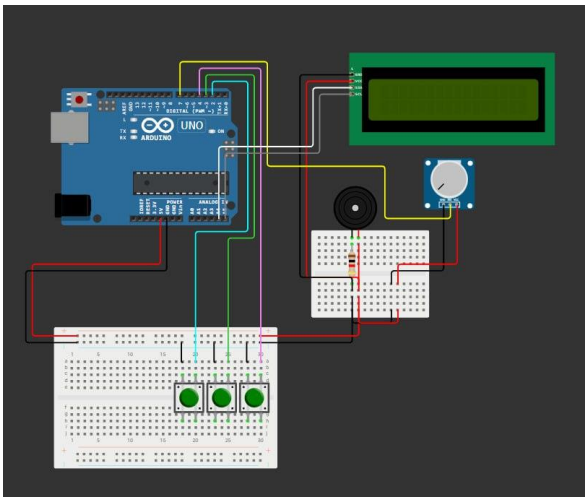
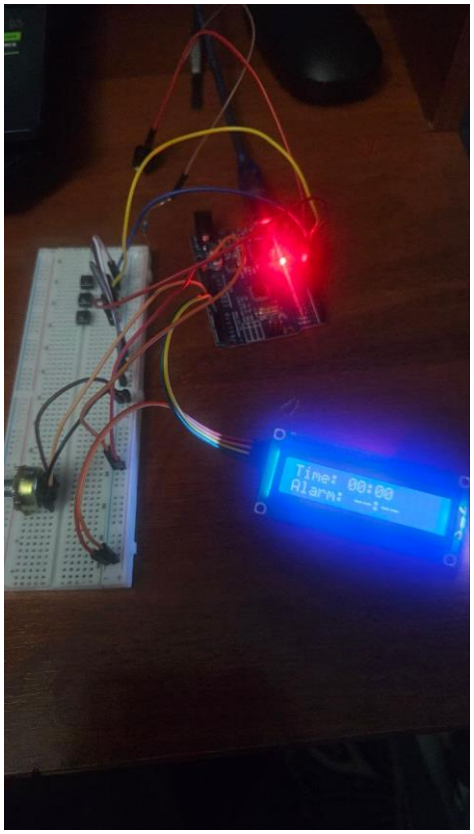


Figure 15. Prototype 0002

In figure 16, a complete circuit was developed, where an LCD screen was also connected, this prototype showed that all connected elements worked properly..

5.5 Prototype 0003



Figures 16. Prototype 0003

On this part, the LCD showed the time, and after a certain time, it was supposed to make a sound from the buzzer, but unfortunately, only the visual part worked, but not the sound.

5.6 Prototype 0004

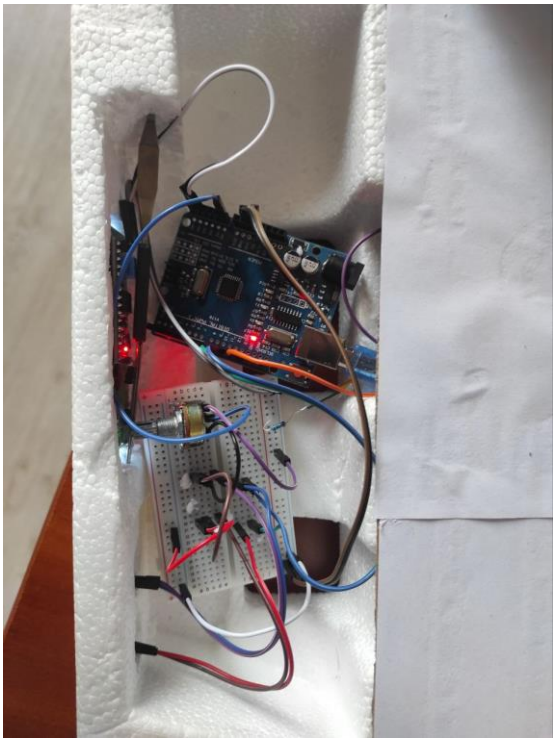


Figure 17. Prototype 0004

This prototype fixes the previous issue (see prototype 0003). Everything works properly. Button presses affect the LCD, and a loud bell sounds at the end of a lesson.

5.7 Prototype 0005, final prototype

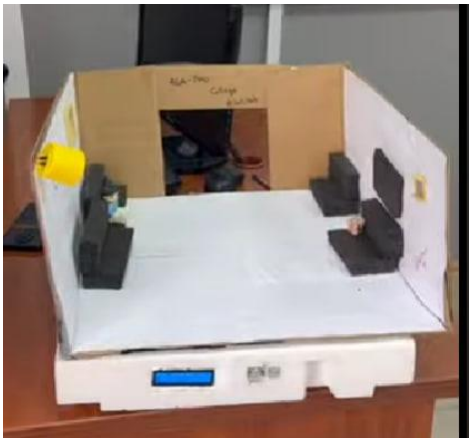


Figure 18. Final prototype

The system was assembled into our college basement design, which now has a working school bell.

6. Implementation

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Figure. 19

Provided communication with the I2C LCD display and enabled text output.

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

Figure. 20

Defined a 16x2 LCD screen used for time and status visualization.

```
const int btnHour = 10;
const int btnMinute = 9;
const int potPin = A0;
const int buzzerPin = 13;
```

Figure. 21

Mapped Arduino pins to buttons, potentiometer, and buzzer.

```
int currentHour = 8;
int currentMinute = 30;
int currentSecond = 0;
int currentDay = 1;
```

Figure. 22

Stored the current simulated time without using an RTC module.

```
bool isSchoolTime = false;
bool isLesson = true;
int lessonNumber = 1;
bool alarmActive = false;
```

Figure. 23

Tracked school time, lesson state, and alarm activity.

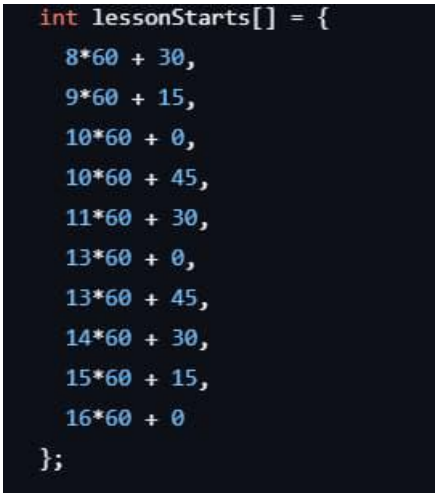


Figure. 24

Defined the starting time of each lesson in minutes.

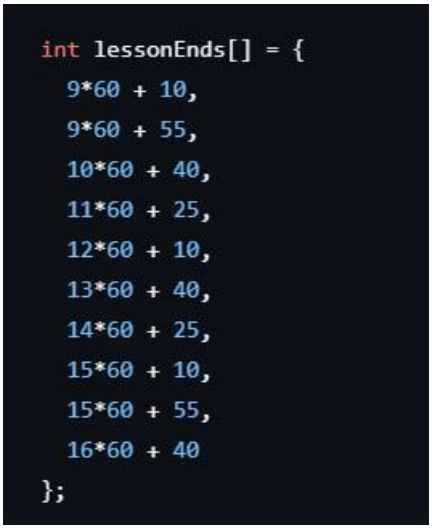


Figure. 25

Defined the ending time of each lesson in minutes.

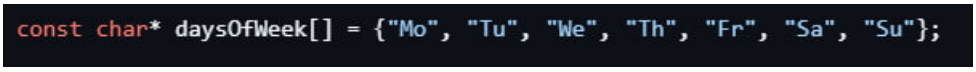


Figure. 26

Stored abbreviated day names for LCD output.

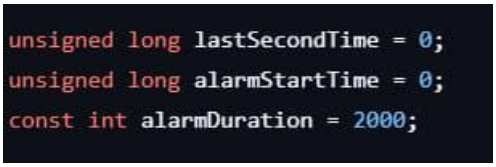


Figure. 27

Controlled second updates and alarm duration.

```
bool lastHourState = HIGH;
bool lastMinuteState = HIGH;
unsigned long lastHourPress = 0;
unsigned long lastMinutePress = 0;
const unsigned long buttonDebounce = 300;
```

Figure 28.

Prevented multiple readings from a single button press.

```
void setup() {
  pinMode(btnHour, INPUT_PULLUP);
  pinMode(btnMinute, INPUT_PULLUP);
  pinMode(buzzerPin, OUTPUT);

  lcd.init();
  lcd.backlight();

  lcd.setCursor(0, 0);
  lcd.print("School Timer");
  lcd.setCursor(0, 1);
  lcd.print("2 buttons only");
  delay(2000);
  lcd.clear();
}
```

Figure 29.

Initialized pins, LCD display, and startup messages.

```
void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - lastSecondTime >= 1000) {
    lastSecondTime = currentMillis;
    updateTime();
  }

  checkButtons();
  checkSchedule();
  checkForBell();

  if (alarmActive) {
    int potValue = analogRead(potPin);
    int volume = map(potValue, 0, 1023, 50, 255);

    if ((currentMillis / 250) % 2 == 0) {
      analogWrite(buzzerPin, volume);
      tone(buzzerPin, 800);
    } else {
      noTone(buzzerPin);
      digitalWrite(buzzerPin, LOW);
    }

    if (currentMillis - alarmStartTime >= alarmDuration) {
      alarmActive = false;
      noTone(buzzerPin);
      digitalWrite(buzzerPin, LOW);
    }
  } else {
    noTone(buzzerPin);
    digitalWrite(buzzerPin, LOW);
  }

  updateDisplay();
  delay(50);
}
```

Figure 30

Executed time updates, input checks, scheduling logic, and display refresh.

```
void updateTime() {
  currentSecond++;
  if (currentSecond >= 60) {
    currentSecond = 0;
    currentMinute++;

    if (currentMinute >= 60) {
      currentMinute = 0;
      currentHour++;

      if (currentHour >= 24) {
        currentHour = 0;
        currentDay++;

        if (currentDay > 7) {
          currentDay = 1;
        }
      }
    }
  }
}
```

Figure 31

Incremented seconds, minutes, hours, and days in software.

```
void checkSchedule() {
  bool isWorkDay = (currentDay >= 1 && currentDay <= 6);
  int currentTimeInMinutes = currentHour * 60 + currentMinute;

  isSchoolTime = isWorkDay && (currentTimeInMinutes >= 8*60+30) &&
                 (currentTimeInMinutes <= 16*60+40);

  if (isSchoolTime) {
    for (int i = 0; i < 10; i++) {
      if (currentTimeInMinutes >= lessonStarts[i] &&
          currentTimeInMinutes < lessonEnds[i]) {
        isLesson = true;
        lessonNumber = i + 1;
        return;
      }
    }
    isLesson = false;
  } else {
    isLesson = false;
    lessonNumber = 0;
  }
}
```

Figure 32

Determined whether the system was in lesson time, break time, or outside school hours.

```
void checkForBell() {
  if (!isSchoolTime || alarmActive) return;

  int currentTimeInMinutes = currentHour * 60 + currentMinute;

  for (int i = 0; i < 10; i++) {
    if (currentTimeInMinutes == lessonStarts[i] && currentSecond == 0) {
      startAlarm();
      return;
    }
  }

  for (int i = 0; i < 10; i++) {
    if (currentTimeInMinutes == lessonEnds[i] && currentSecond == 0) {
      startAlarm();
      return;
    }
  }
}
```

Figure 33

Checked lesson boundaries and activated the bell when required.

```
void startAlarm() {
  if (!alarmActive) {
    alarmActive = true;
    alarmStartTime = millis();
  }
}
```

Figure 34

Started the buzzer sound for a fixed duration.

```
void checkButtons() {
  if (currentMinuteState == LOW && lastMinuteState == HIGH) {
    if (currentMillis - lastMinutePress > buttonDebounce) {
      lastMinutePress = currentMillis;

      int oldHour = currentHour;
      currentMinute += 5;

      if (currentMinute >= 60) {
        currentMinute = currentMinute - 60;
        currentHour++;

        if (oldHour == 23 && currentHour == 24) {
          currentHour = 0;
          currentDay++;
          if (currentDay > 7) currentDay = 1;
        } else if (currentHour >= 24) {
          currentHour = 0;
        }
      }

      currentSecond = 0;
    }
  }
  lastMinuteState = currentMinuteState;

  static unsigned long hourPressStart = 0;
  if (currentHourState == LOW) {
    if (hourPressStart == 0) {
      hourPressStart = currentMillis;
    } else if (currentMillis - hourPressStart > 2000) {
      currentHour = 8;
      currentMinute = 30;
      currentSecond = 0;
      hourPressStart = 0;
    }
  } else {
    hourPressStart = 0;
  }

  static unsigned long minutePressStart = 0;
  if (currentMinuteState == LOW) {
    if (minutePressStart == 0) {
      minutePressStart = currentMillis;
    } else if (currentMillis - minutePressStart > 2000) {
      currentDay = 1;
      minutePressStart = 0;
    }
  } else {
    minutePressStart = 0;
  }
```

Figure 35

Allowed manual adjustment of hours, minutes, and reset actions.

```
void updateDisplay() {
  lcd.print(" ");

  if (currentHour < 10) lcd.print("0");
  lcd.print(currentHour);
  lcd.print(":");
  if (currentMinute < 10) lcd.print("0");
  lcd.print(currentMinute);
  lcd.print(":");
  if (currentSecond < 10) lcd.print("0");
  lcd.print(currentSecond);

  lcd.setCursor(0, 1);

  for (int i = 0; i < 16; i++) {
    lcd.print(" ");
  }
  lcd.setCursor(0, 1);

  if (alarmActive) {
    lcd.print("BELL! L");
    lcd.print(lessonNumber);
  } else if (isSchoolTime) {
    if (isLesson) {
      lcd.print("L");
      lcd.print(lessonNumber);
      lcd.print(" ");

      int timeNow = currentHour * 60 + currentMinute;
      int remaining = lessonEnds[lessonNumber-1] - timeNow;
      if (remaining >= 0) {
        lcd.print(remaining);
        lcd.print("m left");
      }
    } else {
      lcd.print("Break");

      if (lessonNumber > 0) {
        lcd.print(" ");
        lcd.print(lessonNumber);

        if (lessonNumber == 5) {
          lcd.print(" Lnch");
        }
      }
    }
  } else {
    if (currentDay == 7) {
      lcd.print("Sunday - Rest");
    } else if (currentHour < 8 || (currentHour == 8 && currentMinute < 30)) {
      lcd.print("Before school");
```

Figure 36

Displayed current time, day, lesson number, and system status on the LCD screen.

7. Discussion

Although the implemented system performed its core function successfully, it was acknowledged that several practical limitations were present. A primary concern was the reliance on the Arduino's internal timer for timekeeping, which lacked the precision of a dedicated real-time clock module. This design choice inevitably led to cumulative time drift, necessitating periodic manual correction to maintain schedule accuracy. Future iterations of the system would benefit significantly from integrating a dedicated RTC chip, such as the DS3231, to ensure long-term temporal precision autonomously.

The system's operational schedule was hard-coded into the firmware, rendering any modifications to the timetable dependent on reprogramming and re-uploading the software to the microcontroller. This approach stood in contrast to more advanced systems featuring wireless interfaces or physical keypads for on-the-fly schedule adjustments. The minimal two-button interface represented a deliberate trade-off, prioritising simplicity and cost-effectiveness at the expense of operational flexibility. Consequently, while suitable for a static schedule, the system was not readily adaptable to varying institutional timetables.

Furthermore, the auditory output was generated by a low-voltage piezoelectric buzzer. Its tone was perceptibly quieter and sharper than the resonant sound of a traditional school bell or a modern amplified tone. For deployment in larger spaces such as corridors or multiple classrooms, the output device would require an upgrade, potentially to a high-power electronic siren or a relay-controlled mains-voltage bell system.

The software also incorporated specific logical constraints. The system was programmed to recognise a standard six-day school week, explicitly identifying Sundays as a non-operational day. It did not account for public holidays, custom academic calendars, or exceptional schedule changes. These limitations were consistent with the project's scope as a functional prototype for a standardised environment but would need to be addressed for reliable deployment in a real-world institutional setting.

8. Conclusion and future works

We have demonstrated an Arduino-based automated school bell that reliably rings at scheduled times and displays current status on an LCD. The final system solves the core problem of timely bell ringing: it removed manual intervention and ensured classroom periods and breaks are signaled correctly. This contributes to smarter campus infrastructure and could be further integrated into educational IoT solutions.
For future work, we suggest adding features such as:

1. Precision timekeeping: Integrate a real-time clock (RTC) module for accurate time synchronization, eliminating manual adjustment.

2. User programming interface: Include a keypad or mobile app to allow on-the-fly schedule changes without code uploads.

3. Remote alerts: Add a wireless module (Bluetooth or Wi-Fi) to broadcast bell signals to classroom devices or to allow remote monitoring.

4. Enclosure and assembly: Build a custom PCB or enclosure for durability, as the current prototype is on a breadboard.

5. Enhanced sound: Use a louder speaker or alarm bell and add a relay to ring a standard AC bell for large halls.

In summary, our system met its objectives as a low-cost automated bell. With these enhancements, it could become a robust product for schools, further reducing staff effort and improving the learning environment.

9. References

[1] Al-Saadi, H.A. & Mohsen, M.R. (2025) “Leveraging Arduino for Automated Bell Ringing to Optimize Time and Effort in Schools,” *International Journal of Scientific Research and Modern Technology*, 4(8), pp. 18–24.

[2] Sekhar, K.M. (2025) “Automatic Bell With Adjustable Timings,” *ElectronicsForYou,* 6 March, pp. 1093-1100.

[3] Shiral, S.G. *et al.* (2024) “Arduino Based Automatic College Bell System,” *International Journal of Multidisciplinary Research in Science, Engineering and Technology*, 7(3), pp. 4961–4967.

[4] Monk, S. (2016) *Programming Arduino: getting started with sketches*. 2nd edn. New York: McGraw-Hill Education.

[5] ARDUINO-BASED AUTOMATIC INSTITUTE BELL SYSTEM Pavan kumar B,lokesh Reddy Y V,Pavan M,Krishnaiah K,Pulla reddy A

10.Appendices

The following links will take you to direct access to the project's digital assets, including the full source code and a demo video.

Resource	Description	Value / Image Reference
GitHub repository	Full Arduino C/C++ source code	https://github.com/EntertainmentLover/School-bell-Arduino
YouTube video	Demonstration of the final project in operation.	https://youtube.com/shorts/QA7N-nOm_DA

Table 2. Appendices