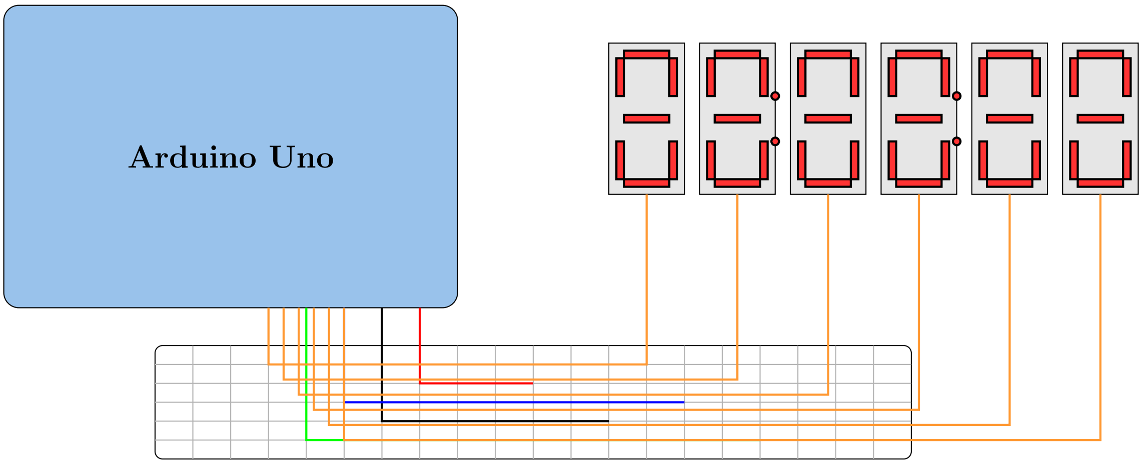
Digital Clock with Arduino

Building a Six-Digit Clock Using Seven-Segment Displays



Digital Clock with Six Seven-Segment Displays

Complete Project Guide with:

- Detailed Circuit Diagrams
- Step-by-Step Assembly Instructions
- Complete Arduino Code
- Troubleshooting and Enhancement Tips

March 23, 2025

Contents

1 Introduction

This guide details building a digital clock using an Arduino Uno and six seven-segment displays. The clock displays time in `HH:MM:SS` format, combining hardware interfacing, circuit design, and software programming.

1.1 Project Overview

The project covers:

- Interfacing seven-segment displays with Arduino
- Time management with optional RTC integration
- Multiplexing techniques to control multiple displays
- Circuit design for efficient power management
- User interface design with buttons for time-setting

1.2 Skill Level

This project is suitable for intermediate Arduino enthusiasts with:

- Basic knowledge of Arduino programming
- Understanding of simple electronic circuits
- Familiarity with breadboard prototyping
- Experience with digital input/output operations

Tip

Beginners should first explore simpler Arduino projects involving LEDs and pushbuttons.

1.3 Time and Cost Estimates

- **Time Required:** 4-6 hours
- **Cost Range:** $15-$30 (excluding Arduino Uno)
- **Difficulty:** Intermediate

2 Materials and Components

2.1 Required Components

2.2 Tools Required

- Computer with Arduino IDE installed
- USB cable for Arduino programming
- Multimeter for troubleshooting

Component	Quantity	Notes
Arduino Uno	1	Main microcontroller
Seven-segment displays	6	Common cathode preferred
74HC595 shift registers	2	8-bit serial-in, parallel-out
BC547 transistors	6	For display control (NPN)
220 resistors	8	Current limiting for segments
1k resistors	6	For transistor bases
10k resistors	2	Pull-up for buttons
Push buttons	2	For setting time
Breadboard	1	For prototyping
Jumper wires	40	Mix of various lengths
DS3231 RTC module	1	Optional but recommended

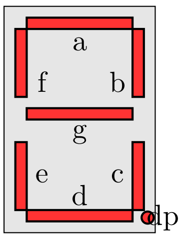
Table 1: List of components required for the digital clock project

- Wire cutters/strippers (optional)
- Soldering iron and solder (for permanent assembly, optional)

2.3 Component Details

2.3.1 Seven-Segment Displays

Seven-segment displays come in two common configurations: common cathode and common anode. This project uses common cathode displays, where all segment LEDs share a common ground connection.



Pin	Connection
1	Segment e
2	Segment d
3	Common Cathode 1
4	Segment c
5	Segment dp
6	Segment b
7	Segment a
8	Common Cathode 2
9	Segment f
10	Segment g

Figure 1: Seven-segment display layout and typical pinout

2.3.2 74HC595 Shift Register

The 74HC595 is an 8-bit serial-in, parallel-out shift register that allows us to control multiple outputs using just three Arduino pins. This is essential for managing the 42 individual LED segments (7 segments $\times$ 6 displays) efficiently.

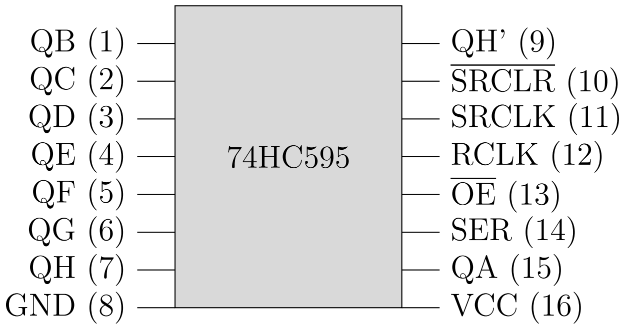


Figure 2: 74HC595 shift register pinout

Tip

For our project, we'll use these key pins:

- SER (Pin 14): Data input from Arduino
- RCLK (Pin 12): Register clock, latch pin
- SRCLK (Pin 11): Shift register clock
- $\overline{\text{OE}}$ (Pin 13): Output enable (active low)
- QA-QH (Pins 15, 1-7): Outputs to seven-segment display
- QH' (Pin 9): Serial output for daisy-chaining

3 Circuit Design

3.1 Overall Circuit Architecture

The digital clock circuit consists of three main sections:

1. **Control section:** Arduino Uno and optional RTC module
2. **Driver section:** Shift registers and transistors
3. **Display section:** Six seven-segment displays

The Arduino controls the shift registers, which provide the segment patterns, while the transistors activate each display in sequence.

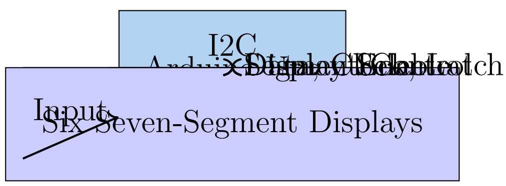


Figure 3: Block diagram of the digital clock circuit

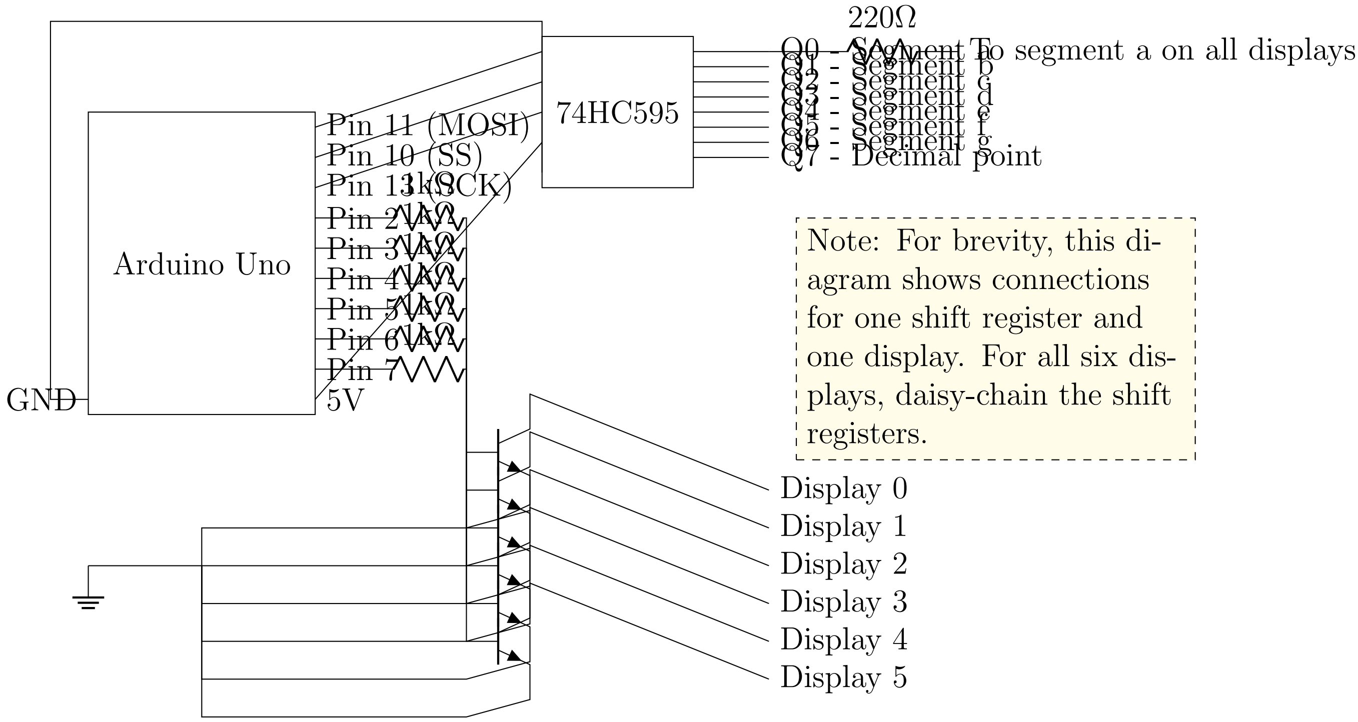


Figure 4: Simplified circuit diagram for the digital clock

3.2 Detailed Circuit Diagram

3.3 Multiplexing Technique

To control six displays with limited Arduino pins, we use a technique called multiplexing. This involves:

1. Activating only one display at a time
2. Quickly cycling through all displays (¿60Hz to avoid visible flicker)
3. Persistence of vision makes all displays appear lit simultaneously

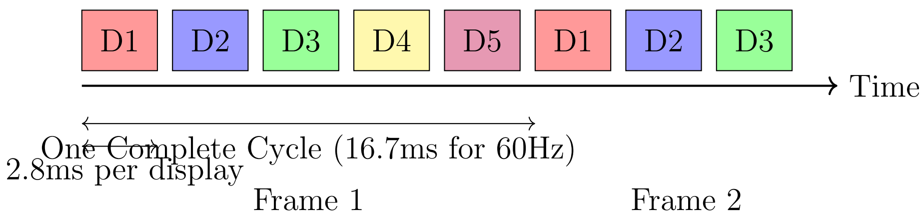


Figure 5: Multiplexing timing diagram for six displays

4 Assembly Instructions

4.1 Breadboard Layout

Begin by placing components on the breadboard according to this recommended layout:

1. Place the Arduino Uno at the edge of your workspace

2. Position the breadboard in front of the Arduino
3. Place the shift registers in the center of the breadboard
4. Arrange the six seven-segment displays in sequence at the top of the breadboard
5. Place transistors below the displays
6. Add resistors and wiring

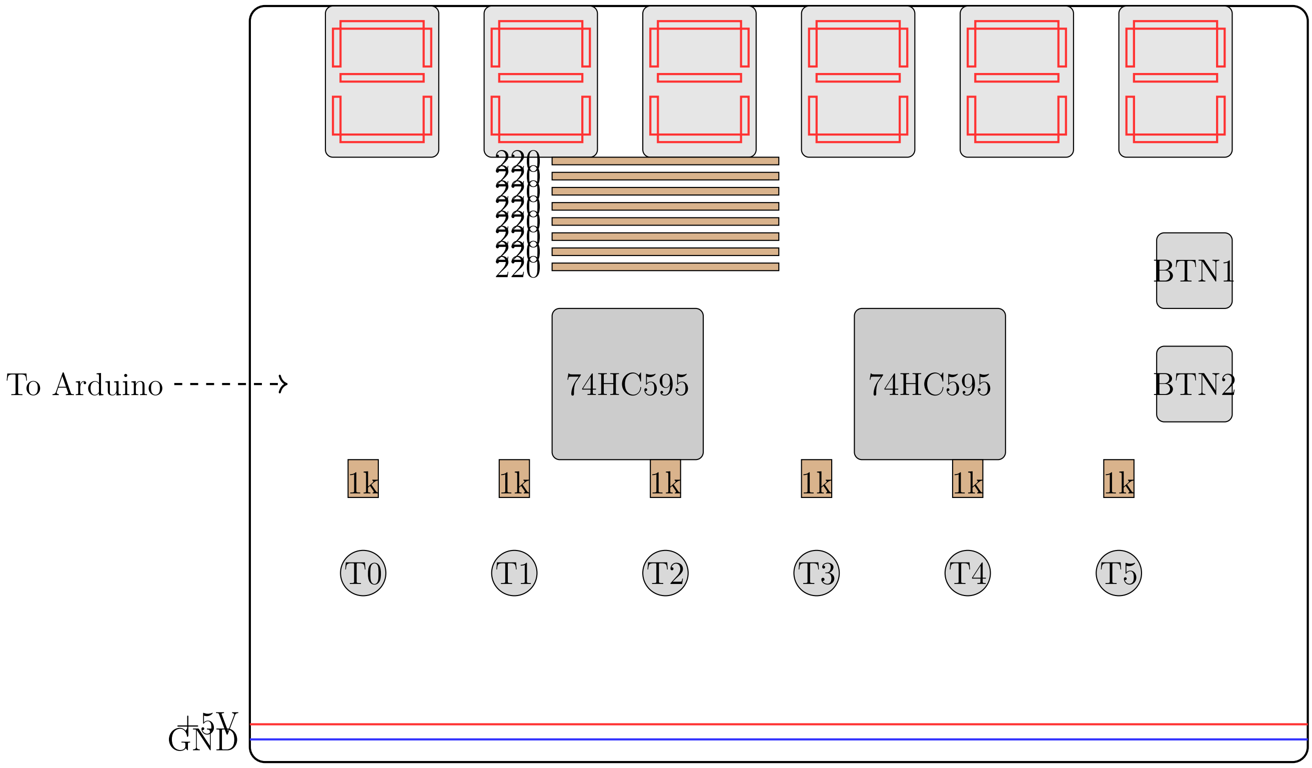


Figure 6: Breadboard layout for the digital clock project

4.2 Step-by-Step Wiring Instructions

Follow these steps carefully to wire the entire circuit:

4.2.1 Power Connections

- Connect Arduino 5V to the breadboard power rail
- Connect Arduino GND to the breadboard ground rail
- Ensure all components have access to power and ground

4.2.2 Shift Register Connections

- Connect Arduino Pin 11 (MOSI) to the first shift register's SER pin (14)
- Connect Arduino Pin 13 (SCK) to both shift registers' SRCLK pins (11)
- Connect Arduino Pin 10 (SS) to both shift registers' RCLK pins (12)

- Connect the first shift register's QH' pin (9) to the second shift register's SER pin (14)
- Connect both shift registers' VCC pins (16) to 5V
- Connect both shift registers' GND pins (8) to ground
- Connect both shift registers' OE pins (13) to ground (to enable output)
- Connect both shift registers' SRCLR pins (10) to 5V (clear is active low)

4.2.3 Transistor Base Connections

- Connect Arduino pins 2-7 to the six transistor bases through 1k resistors

4.2.4 Display Control

- Connect each transistor collector to the common cathode of its corresponding display
- Connect all transistor emitters to ground

4.2.5 Segment Connections

- Connect each shift register output (QA-QH) through 220 resistors to the corresponding segments (a-g and dp) of all displays

4.2.6 Button Connections

- Connect both buttons with one side to ground
- Connect the other sides to Arduino pins 8 and 9 through 10k pull-up resistors to 5V

5 Programming the Clock

5.1 Software Requirements

Before proceeding with programming, ensure you have:

- Arduino IDE (version 1.8.x or later)
- Required libraries installed:
  - ShiftRegister74HC595 library (optional but recommended)
  - RTClib library (if using the DS3231 module)
  - Time library

You can install these libraries through the Arduino IDE's Library Manager.

5.2 Basic Clock Code

```
// Digital Clock with 6 Seven-Segment Displays
// Uses two 74HC595 shift registers and multiplexing

// Pin definitions
const int dataPin = 11;    // DS (Serial data input)
const int latchPin = 10;   // ST_CP (Storage register clock)
const int clockPin = 13;   // SH_CP (Shift register clock)

// Display selection pins (transistor control)
const int displayPins[6] = {2, 3, 4, 5, 6, 7};

// Button pins
const int hourButton = 8;
const int minuteButton = 9;

// Time variables
int hours = 12;
int minutes = 0;
int seconds = 0;

// Digit patterns for 0-9 (common cathode configuration)
// Segments: a, b, c, d, e, f, g
const byte digits[10] = {
  B11111100,  // 0
  B01100000,  // 1
  B11011010,  // 2
  B11110010,  // 3
  B01100110,  // 4
  B10110110,  // 5
  B10111110,  // 6
  B11100000,  // 7
  B11111110,  // 8
  B11110110   // 9
};

void setup() {
  // Initialize shift register pins
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);

  // Initialize display selection pins
  for (int i = 0; i < 6; i++) {
    pinMode(displayPins[i], OUTPUT);
    digitalWrite(displayPins[i], LOW);  // All displays off initially
  }

  // Initialize button pins with pull-up resistors
  pinMode(hourButton, INPUT_PULLUP);
  pinMode(minuteButton, INPUT_PULLUP);

  // Optional: Serial for debugging
  Serial.begin(9600);
}

void loop() {
  // Check for button presses (time setting)
  checkButtons();
```

```

  // Update time (increment seconds)
  static unsigned long lastSecond = 0;
  if (millis() - lastSecond >= 1000) {
    updateTime();
    lastSecond = millis();
  }

  // Display the time using multiplexing
  displayTime();
}

void updateTime() {
  // Increment seconds and handle overflow
  seconds++;
  if (seconds >= 60) {
    seconds = 0;
    minutes++;
    if (minutes >= 60) {
      minutes = 0;
      hours++;
      if (hours >= 24) {
        hours = 0;
      }
    }
  }
}

void checkButtons() {
  // Check hour button (with debounce)
  static unsigned long lastHourPress = 0;
  if (digitalRead(hourButton) == LOW) {
    if (millis() - lastHourPress > 200) {  // Simple debounce
      hours = (hours + 1) % 24;
      lastHourPress = millis();
    }
  }

  // Check minute button (with debounce)
  static unsigned long lastMinutePress = 0;
  if (digitalRead(minuteButton) == LOW) {
    if (millis() - lastMinutePress > 200) {  // Simple debounce
      minutes = (minutes + 1) % 60;
      seconds = 0;  // Reset seconds when setting minutes
      lastMinutePress = millis();
    }
  }
}

void displayTime() {
  // Break time into individual digits
  int h1 = hours / 10;
  int h2 = hours % 10;
  int m1 = minutes / 10;
  int m2 = minutes % 10;
  int s1 = seconds / 10;
  int s2 = seconds % 10;

  // Display each digit with brief delay (~2ms)
  // This creates the multiplexing effect
```

```
  displayDigit(0, h1);
  delay(2);
  displayDigit(1, h2, true);  // Add decimal point (colon)
  delay(2);
  displayDigit(2, m1);
  delay(2);
  displayDigit(3, m2, true);  // Add decimal point (colon)
  delay(2);
  displayDigit(4, s1);
  delay(2);
  displayDigit(5, s2);
  delay(2);
}

void displayDigit(int display, int digit, bool decimalPoint = false) {
  // Turn off all displays
  for (int i = 0; i < 6; i++) {
    digitalWrite(displayPins[i], LOW);
  }

  // Prepare the digit pattern (with optional decimal point)
  byte pattern = digits[digit];
  if (decimalPoint) {
    pattern |= B00000001; // Set DP bit
  }

  // Send the pattern to shift registers
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, pattern);
  shiftOut(dataPin, clockPin, MSBFIRST, 0);  // Second register (if needed)
  digitalWrite(latchPin, HIGH);

  // Turn on the current display
  digitalWrite(displayPins[display], HIGH);
}
```

5.3 Enhanced Code with RTC Integration

For more accurate timekeeping, add an RTC module with this enhanced code:

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

RTC_DS3231 rtc;

// Rest of the code from above...

void setup() {
  // Initialize the RTC
  Wire.begin();
  if (!rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  // Only set the time if RTC lost power
  if (rtc.lostPower()) {
    Serial.println("RTC lost power, setting default time!");
    // Following line sets the RTC to date & time this sketch was compiled
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
```

```
  }

  // Rest of setup code...
}

void updateTime() {
  // Get time from RTC
  DateTime now = rtc.now();

  // Update our variables
  hours = now.hour();
  minutes = now.minute();
  seconds = now.second();
}

// When setting time with buttons, also update the RTC
void setRTC() {
  rtc.adjust(DateTime(2023, 1, 1, hours, minutes, seconds));
}

// Update the checkButtons function to call setRTC when time is changed
```

6 Troubleshooting

6.1 Common Issues and Solutions

1. **Display shows incorrect digits**
   - Check segment wiring connections
   - Verify the digit patterns in the code match your display type (common cathode vs common anode)
   - Ensure the shift register outputs are connected to the correct segments
2. **Flickering or uneven display brightness**
   - Increase display refresh rate by reducing delay times in multiplexing loop
   - Check transistor connections and ensure they're fully switching on
   - Add capacitors (100nF) between power and ground near displays for stability
3. **Time drifts or resets on power cycle**
   - Add the RTC module for accurate timekeeping
   - Check battery in RTC module if it's still resetting
   - Verify I2C connections to the RTC module
4. **One or more displays don't light up**
   - Check the transistor and its connections for that specific display
   - Verify common cathode connections for each display
   - Test display with direct power connection to isolate the issue
5. **Buttons not responding**

- Check pullup resistor connections for the buttons
- Add software debounce logic or extend the debounce delay time
- Verify button pins in code match physical connections

6.2 Testing Individual Components

When troubleshooting, it's helpful to test individual components:

```
// Test all displays cycle through numbers 0-9
void testDisplays() {
  for (int digit = 0; digit <= 9; digit++) {
    for (int i = 0; i < 6; i++) {
      displayDigit(i, digit);
      delay(500);  // Longer delay for visual test
    }
  }
}

// Test shift register by sequentially lighting each segment
void testShiftRegister() {
  for (int i = 0; i < 8; i++) {
    byte pattern = (1 << i);
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, MSBFIRST, pattern);
    shiftOut(dataPin, clockPin, MSBFIRST, 0);
    digitalWrite(latchPin, HIGH);
    delay(500);
  }
}
```

7 Enhancements and Modifications

7.1 Adding Temperature Display

If you're using the DS3231 RTC module, it includes a temperature sensor that you can access to display temperature readings:

```
void displayTemperature() {
  float temp = rtc.getTemperature();
  int tempInt = (int)temp;
  int tempDec = (int)(temp * 10) % 10;

  // Display temperature for a few seconds
  for (int i = 0; i < 20; i++) {  // ~2 seconds
    displayDigit(0, tempInt / 10);
    delay(2);
    displayDigit(1, tempInt % 10, true);  // Decimal point
    delay(2);
    displayDigit(2, tempDec);
    delay(2);
    displayDigit(3, 12);  // Display 'C' for Celsius
    delay(2);
  }
}
```

7.2 Adding Alarm Functionality

Add a buzzer and an additional button to implement an alarm feature:

```
const int alarmButton = 12;
const int buzzerPin = A0;
int alarmHour = 7;
int alarmMinute = 0;
bool alarmEnabled = false;

void setup() {
  // Existing setup code...
  pinMode(alarmButton, INPUT_PULLUP);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  // Existing loop code...

  // Check if alarm should trigger
  if (alarmEnabled && hours == alarmHour && minutes == alarmMinute &&
   seconds < 30) {
    // Sound the alarm for 30 seconds
    tone(buzzerPin, 1000, 500);
    delay(1000);
  }
}

// Add to button checking function
void checkAlarmButton() {
  static unsigned long lastAlarmPress = 0;
  if (digitalRead(alarmButton) == LOW) {
    if (millis() - lastAlarmPress > 200) {
      // Cycle through setting alarm hour, minute, or toggling on/off
      static int alarmState = 0;
      alarmState = (alarmState + 1) % 3;

      switch (alarmState) {
        case 0:
          alarmEnabled = !alarmEnabled;
          break;
        case 1:
          alarmHour = (alarmHour + 1) % 24;
          break;
        case 2:
          alarmMinute = (alarmMinute + 1) % 60;
          break;
      }

      lastAlarmPress = millis();
    }
  }
}
```

7.3 Creating a PCB Version

For a more permanent and compact solution, consider creating a printed circuit board (PCB). This offers several advantages:

- More reliable connections

- Smaller footprint
- Professional appearance
- Easier troubleshooting

Many online services allow you to upload your design and receive manufactured PCBs within weeks. Popular options include JLCPCB, PCBWay, and OSH Park.

8 Conclusion

This Arduino-based digital clock project combines fundamental electronics concepts with practical programming skills. By successfully building this clock, you've gained experience with:

- Microcontroller programming
- Time-keeping and RTC interfacing
- Shift register usage for output expansion
- Multiplexing techniques for displays
- Transistor switching applications
- User interface design with buttons

Consider extending this project with additional features like:

- Weather display
- Alarm with snooze function
- Customizable brightness
- Different display modes (date, temperature, etc.)
- WiFi or Bluetooth connectivity for automatic time synchronization

9 References and Resources

9.1 Books

- "Arduino Cookbook" by Michael Margolis
- "Make: Electronics" by Charles Platt