Digital Clock Implementation with Arduino

EE24BTECH11024 - Abhimanyu Koushik

March 24, 2025

Contents

1 Introduction

This report describes the design, implementation, and analysis of a digital clock using an *Arduino Uno* microcontroller, *7-segment displays*, and *multiplexing techniques*. The project employs AVR-GCC programming for efficient control and accurate timekeeping.

2 Objectives

The primary objectives of this project are:

- To create a digital clock capable of displaying hours, minutes, and seconds.
- To use *multiplexing* techniques for reducing the number of required microcontroller pins.
- To implement precise time management using *Timer1 interrupts*.
- To demonstrate AVR-GCC direct register manipulation for efficient hardware control.

3 Materials Required

- *Arduino Uno* board (ATmega328P)
- Six *7-segment displays*
- *7447 BCD-to-7-segment decoders*
- *180Ω resistors* (current limiting resistors)
- *Push buttons* (for time adjustments)
- Breadboard and jumper wires
- Power supply or USB connection for the Arduino

4 Circuit Description

The clock uses six 7-segment displays to represent *HH:MM:SS*. The digits are arranged as follows:

$$\text{HH:MM:SS} = \text{Hour Tens} + \text{Hour Units} + \text{Minute Tens} + \text{Minute Units} + \text{Second Tens} + \text{Second Units}$$

4.1 Wiring Configuration

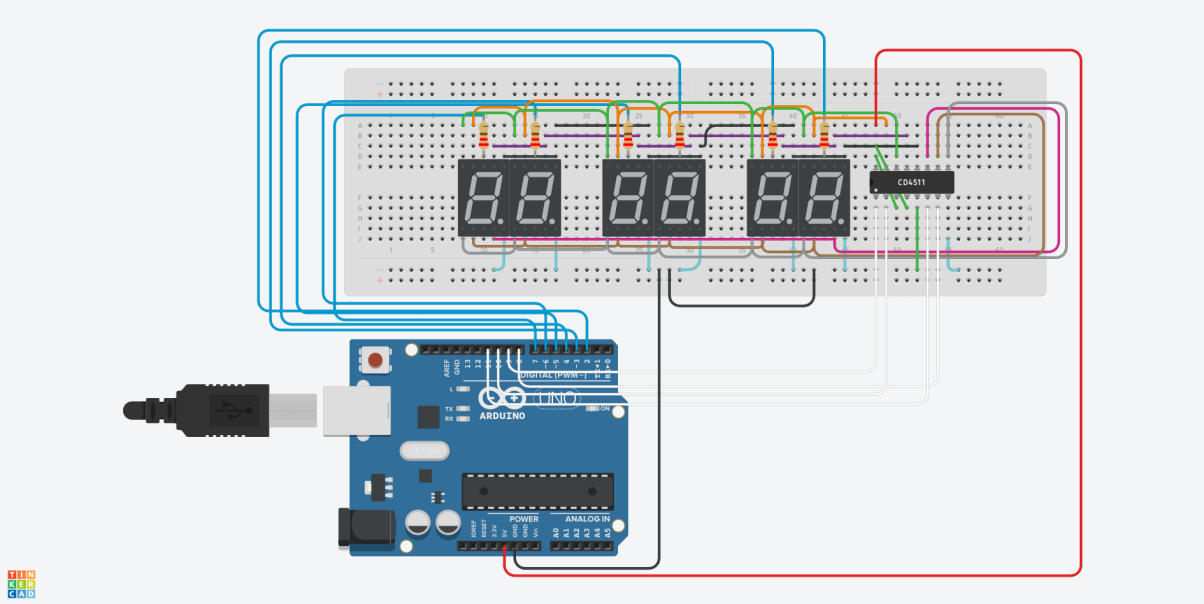

Figure 1: Digital Clock Circuit using Arduino and 7-segment displays

4.2 Pin Diagrams

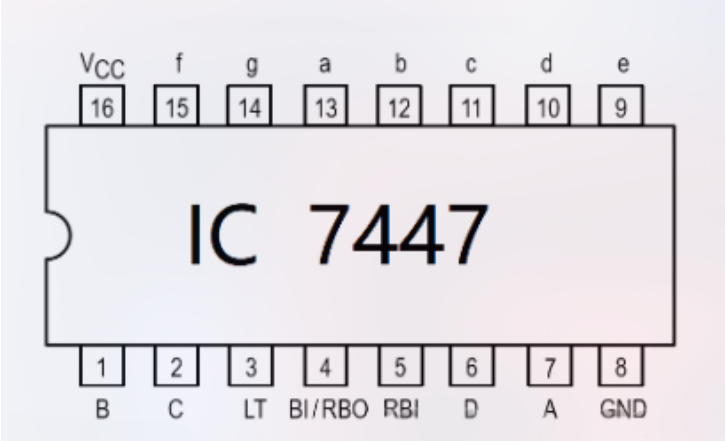

Figure 2: 7447 BCD-to-7-segment Decoder Pinout

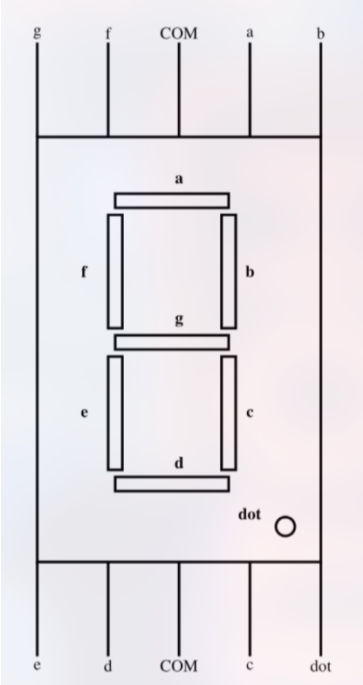

Figure 3: 7-segment Display Pinout

4.3 Connections

4.3.1 7447 Decoder to Arduino Connections

- D2 → A (LSB) on 7447
- D3 → B on 7447
- D4 → C on 7447
- D5 → D (MSB) on 7447
- GND → GND (Common ground)

4.3.2 7447 to 7-segment Display Connections

- 7447 Pin 9 → Segment E
- 7447 Pin 10 → Segment D
- 7447 Pin 11 → Segment C
- 7447 Pin 12 → Segment B
- 7447 Pin 13 → Segment A
- 7447 Pin 14 → Segment G
- 7447 Pin 15 → Segment F

4.3.3 7-segment Display to Arduino Connections

- A0 → Display 1 Common Anode
- A1 → Display 2 Common Anode
- A2 → Display 3 Common Anode
- A3 → Display 4 Common Anode
- A4 → Display 5 Common Anode
- A5 → Display 6 Common Anode

5 Working Principle

The clock uses *multiplexing* to drive multiple 7-segment displays while reducing the required number of pins. The Arduino cycles through each display quickly (approximately every 2ms), creating the illusion of simultaneous illumination.

5.1 Timing and Multiplexing

The *Timer1 interrupt* triggers every second to update the clock values. The display refresh rate is approximately:

$$\text{Refresh rate} = \frac{1}{12\text{ms}} \approx 83\text{Hz}$$

6 Software Implementation

6.1 Timer1 Configuration

The *Timer1* interrupt generates a 1-second pulse to increment the time. Timer1 is configured in **CTC mode** with a prescaler of 1024, and the **Output Compare Register (OCR1A)** is set to 15625, ensuring a precise 1-second interval:

```
// ======= Timer1 Configuration =======
void initTimer1() {
    TCCR1B |= (1 << WGM12);              // CTC mode (Clear Timer on Compare Match)
    TCCR1B |= (1 << CS12) | (1 << CS10); // Prescaler 1024
    OCR1A = 15625;                       // 1-second interval (16 MHz / 1024)
    TIMSK1 |= (1 << OCIE1A);             // Enable Timer1 interrupt
    sei();                               // Enable global interrupts
}
```

6.2 ISR (Interrupt Service Routine)

The *ISR* handles time increments and rollovers every 1 second:

```
// ======= ISR (Interrupt Service Routine) =======
ISR(TIMER1_COMPA_vect) {
    seconds++;                          // Increment seconds every 1 second
    if (seconds == 60) {
        seconds = 0;
        minutes++;
    }
    if (minutes == 60) {
        minutes = 0;
        hours++;
    }
    if (hours == 24) {
        hours = 0;
    }
}
```

6.3 BCD Conversion for 7-segment Display

The *BCD encoding* function converts the current time values to displayable digits by setting the corresponding bits on the **7447 BCD to 7-segment decoder**:

```
// ======= BCD Setting Function =======
void setBCD(int digit, const int* decoderPins) {
    digitalWrite(decoderPins[0], digit & 0x01);       // A (LSB)
    digitalWrite(decoderPins[1], (digit >> 1) & 0x01); // B
    digitalWrite(decoderPins[2], (digit >> 2) & 0x01); // C
    digitalWrite(decoderPins[3], (digit >> 3) & 0x01); // D (MSB)
}
```

6.4 Multiplexing Routine

The **multiplexing routine** activates one display at a time, ensuring that only one digit is displayed at any moment. This prevents ghosting and reduces power consumption:

```
// ======= Multiplexing Function =======
void displayTime() {
    int digits[6] = {
        hours / 10, hours % 10,
        minutes / 10, minutes % 10,
        seconds / 10, seconds % 10
    };

    for (int display = 0; display < 6; display++) {
        // Disable all displays
        for (int i = 0; i < 6; i++) {
            digitalWrite(displayPins[i], LOW);
        }

        // Set the BCD value for the current digit
        setBCD(digits[display], decoderPins);

        // Enable current display
        digitalWrite(displayPins[display], HIGH);

        _delay_ms(2);  // Display refresh rate
    }
}
```

7 Challenges and Solutions

- Flickering: Increasing the refresh rate resolved flickering issues.
- Time accuracy: Timer1 was configured with a prescaler of 1024 to ensure accurate 1-second timekeeping.
- Pin limitations: Multiplexing allowed the use of fewer pins by controlling each digit sequentially.

8 Conclusion

This project successfully demonstrates the implementation of a digital clock using *AVR-GCC* programming on an *Arduino Uno*. The use of multiplexing reduces the required number of pins, while Timer1 interrupts ensure accurate timekeeping. The clock displays *hours, minutes, and seconds* using six 7-segment displays and a 7447 BCD-to-7-segment decoder.

9 Source Code and Documentation

The complete source code and documentation are available on GitHub:

https://github.com/AbhimanyuKoushik/Digital-Clock-Arduino