Digital Clock

Spoorthi Yellamanchali - EE24BTECH11065

March 22, 2025

1 Aim

To design and implement a digital clock using an Arduino Uno.

2 Components Used

Table 1 lists the components used in this project along with their purpose and connections.

Component	**Quantity**	**Purpose and Connection**
Arduino Uno	1	Main microcontroller unit
7447 Decoder	1	Converts BCD to 7-segment display signals
Common Anode 7-Segment Display	6	Displays time digits
220Ω Resistors	6	Current limiting resistors for displays
Push Buttons	3	Used for manual hour, minute, and second adjustments
Breadboards	2	For making connections
Jumper Wires	Multiple	For electrical connections

Table 1: List of components and their usage

3 Working Principle

- This clock operates using a multiplexing technique, where only one 7-segment display is activated at a time.
- The Arduino rapidly switches between displays, creating an illusion of a continuous display to the human eye. The 7447 decoder is used to convert BCD values into 7-segment outputs.

4 Multiplexing Explanation

Multiplexing is used to drive multiple 7-segment displays with fewer microcontroller pins. By cycling through displays quickly, we reduce the number of required I/O pins. This technique involves:

- Assigning a unique enable pin to each display.
- Sending the appropriate BCD value to the 7447 decoder.
- Activating one display at a time while others are off.

This process is performed fast enough that the human eye perceives a steady display.

5 Connections

5.1 BCD Pins (Arduino to 7447 Decoder)

The following table describes the connection between the Arduino and the 7447 BCD decoder:

BCD Pin (Arduino)	Pin on ATmega328P	7447 Decoder Input	Function
BCD_A	PD2	A	Least significant(BCD)
BCD_B	PD3	B	Second bit of BCD
BCD_C	PD4	C	Third bit of BCD
BCD_D	PD5	D	Most significant bit(BCD)

Table 2: BCD Pin Connections

5.2 7-Segment Display Selection

Each of the six 7-segment displays is controlled through multiplexing:

Display	Arduino Pin	ATmega328P Pin	Function
DISP1	PD6	10	Selects Hour Tens display
DISP2	PD7	11	Selects Hour Units display
DISP3	PB0	8	Selects Minute Tens display
DISP4	PB1	9	Selects Minute Units display
DISP5	PB2	10	Selects Second Tens display
DISP6	PB3	11	Selects Second Units display

Table 3: 7-Segment Display Selection

The microcontroller rapidly switches between these displays using multiplexing, ensuring that all digits appear visible simultaneously.

5.3 Push Button Connections

The push buttons allow manual time adjustment:

Button	Arduino Pin	ATmega328P Pin	Function
HOUR_BUTTON	PB4	12	Increments the hour
MIN_BUTTON	PC0	A0	Increments the minute
SEC_BUTTON	PC1	A1	Increments the second

Table 4: Push Button Connections

Each button uses an internal pull-up resistor, meaning it reads HIGH when unpressed and LOW when pressed. A software debounce mechanism prevents false triggers.

6 Code implementation

This document explains the modular structure of the **Arduino-based digital clock** that uses a **7447 BCD to 7-segment decoder** for display multiplexing.

6.1 Modules of the Code

6.1.1 BCD Encoder Module

The function `setBCD(uint8_t num)` converts a decimal number into a 4-bit BCD format and sends it to the 7447 decoder.

```
void setBCD(uint8_t num) {
    // Clear previous BCD data while preserving unaffected bits
    PORTD = (PORTD & 0xC3) | ((num & 0x0F) << 2);
}
```

6.1.2 Display Selection Module

The function `selectDisplay(uint8_t disp)` selects one of the six 7-segment displays.

```
void selectDisplay(uint8_t disp) {
    // Turn off all display selection lines before enabling the
        required one
    PORTD &= ~((1 << DISP1) | (1 << DISP2));
    PORTB &= ~((1 << DISP3) | (1 << DISP4) | (1 << DISP5) | (1 <<
        DISP6));

    // Activate only the required display based on the index
    switch (disp) {
        case 0: PORTD |= (1 << DISP1); break;  // Hour Tens
        case 1: PORTD |= (1 << DISP2); break;  // Hour Units
        case 2: PORTB |= (1 << DISP3); break;  // Minute Tens
        case 3: PORTB |= (1 << DISP4); break;  // Minute Units
        case 4: PORTB |= (1 << DISP5); break;  // Second Tens
        case 5: PORTB |= (1 << DISP6); break;  // Second Units
    }
}
```

6.1.3 Multiplexed Display Module

The function `displayTime()` cycles through all displays and updates them.

```
void displayTime() {
    // Extract individual digits for hours, minutes, and seconds
    uint8_t digits[6] = {
        hours / 10, hours % 10,  // Hour Tens and Units
        minutes / 10, minutes % 10,  // Minute Tens and Units
        seconds / 10, seconds % 10   // Second Tens and Units
    };

    static uint8_t currentDisplay = 0;  // Keeps track of the active display

    setBCD(digits[currentDisplay]);  // Send the digit value to the 7447 decoder
    selectDisplay(currentDisplay);   // Activate the respective 7-segment display
    currentDisplay = (currentDisplay + 1) % 6;  // Cycle through displays
}
```

6.1.4 Button Handling Module

This function reads the button inputs and updates the time accordingly.

```
void checkButtons() {
    // Check if the hour button is pressed
    if (!(PINB & (1 << HOUR_BUTTON))) {  // Active low button press
        _delay_ms(50);  // Debounce delay
        if (!(PINB & (1 << HOUR_BUTTON))) {  // Ensure the button is still pressed
            hours = (hours + 1) % 24;  // Increment the hour, rolling over at 24
        }
    }

    // Check if the minute button is pressed
    if (!(PINC & (1 << MIN_BUTTON))) {
        _delay_ms(50);
        if (!(PINC & (1 << MIN_BUTTON))) {
            minutes = (minutes + 1) % 60;  // Increment minutes, rolling over at 60
        }
    }

    // Check if the second button is pressed
    if (!(PINC & (1 << SEC_BUTTON))) {
        _delay_ms(50);
        if (!(PINC & (1 << SEC_BUTTON))) {
            seconds = (seconds + 1) % 60;  // Increment seconds, rolling over at 60
        }
    }
}
```

6.2 Compilation and Upload Process

The code for the digital clock was written in **Embedded C** and compiled using the **AVR-GCC** toolchain. The following steps were followed:

1. The code was written and compiled using the AVR-GCC ‘make‘ command to generate the binary file.

2. The compiled binary was then transferred into the **precompiled section** of **ArduinoDroid**, an Android-based IDE for Arduino development.

3. Using ArduinoDroid, the binary was uploaded to the **Arduino Uno** microcontroller, ensuring the clock’s execution.

This method allows seamless integration of **AVR low-level programming** with **Arduino-based deployment** for better flexibility and control.

7 Conclusion

This project successfully implements a digital clock using an Arduino Uno and a 7447 BCD decoder. Multiplexing effectively reduces hardware requirements while maintaining clear and readable digit output. The modular code structure ensures easy understanding and modifications.