Digital Clock using Arduino UNO and 7447 Decoder

S. Sai Akshita - EE24BTECH11054

March 24, 2025

1 Introduction

This document provides an in-depth explanation of the digital clock implemented using an Arduino UNO, six seven-segment displays, a 7447 BCD to seven-segment decoder, three push buttons, and other required components.

2 Components Used

- Arduino - 1
- Breadboard - 1
- Common-anode Seven segment displays - 6
- 7447 Decoder - 1
- USB A to USB B cable - 1
- OTG adapter - 1
- Jumper wires (Male-Male) - 70
- Resistors 220 Ω - 6
- Push Buttons - 3

3 Pin Configuration and Breadboard Connections

3.1 Arduino UNO to 7447 Decoder (BCD Inputs)

The BCD inputs of the 7447 decoder (A, B, C, D) are connected to the following digital pins of the Arduino:

7447 BCD Pin	Arduino Digital Pin	ATmega328P Pin
A	D2 (PD2)	4
B	D3 (PD3)	5
C	D4 (PD4)	6
D	D5 (PD5)	7

3.2 Multiplexing Control (Display Selection)

Since there are 6 displays but only one 7447 decoder, we use digital pins to control which display is active at any moment.

Display Position	Arduino Pin	ATmega328pin
Hour Tens (DISP1)	D6 (PD6)	12
Hour Units (DISP2)	D7 (PD7)	13
Minute Tens (DISP3)	D8 (PB0)	8
Minute Units (DISP4)	D9 (PB1)	9
Second Tens (DISP5)	D10 (PB2)	10
Second Units (DISP6)	D11 (PB3)	11

3.3 Seven-Segment Display Connections

Each 7-segment display shares the same segment control lines coming from the 7447 decoder.

7447 Decoder Output	7-Segment Pins
a	Segment A
b	Segment B
c	Segment C
d	Segment D
e	Segment E
f	Segment F
g	Segment G

3.4 Push Button Connections

Push buttons are used to increment hours, minutes, and seconds manually.

Button Function	Arduino Pin	ATmega328P Pin
Hour Button	D12 (PB4)	12
Minute Button	A0 (PC0)	23
Second Button	A1 (PC1)	24

Note: Internal pull-up resistors are enabled in the code to avoid external pull-up resistors.To prevent bouncing, where the button state might change rapidly due to physical contact (This can cause multiple increments instead of just one), a debounce delay is added (50ms), which ensures that only one increment happens for each button press (or jumper wire tap).

4 Power Supply Connection

The Arduino UNO is powered via USB from a mobile phone. The VCC and GND connections for the circuit are:

- **VCC (+5V)** from Arduino to Breadboard Power Rail

- **GND (0V)** from Arduino to Breadboard Ground Rail
- The common anode terminals of all 7-segment displays are connected to **+5V**

5 Circuit Working Mechanism

- The Arduino outputs BCD signals to the 7447 decoder, which then converts them into signals to drive the segments.
- Display multiplexing is handled by enabling displays at a time via digital pins.
- A timer in the Arduino updates the seconds count automatically.
- The three push buttons allow manual adjustments of the time.

6 Code Implementation

The following code is used to drive the seven-segment displays and implement the timekeeping functionality.

6.1 Code Listing

```
#include <avr/io.h>
#include <util/delay.h>

#define F_CPU 16000000UL // CPU clock speed

// **BCD Pins (Arduino -> 7447 Decoder)**//
#define BCD_A PD2
#define BCD_B PD3
#define BCD_C PD4
#define BCD_D PD5

// **7-Segment Display Selection Pins**
#define DISP1 PD6
#define DISP2 PD7
#define DISP3 PB0
#define DISP4 PB1
#define DISP5 PB2
#define DISP6 PB3

// **Push Buttons**
#define HOUR_BUTTON PB4
#define MIN_BUTTON PC0
#define SEC_BUTTON PC1

// **Time Variables**
volatile uint8_t hours = 0, minutes = 0, seconds = 0;
```

```
// **BCD Encoder Function**
void setBCD(uint8_t num) {
    PORTD = (PORTD & 0xC3) | ((num & 0x0F) << 2);
}

// **Multiplexing Function for Display Selection**
void selectDisplay(uint8_t disp) {
    PORTD &= ~((1 << DISP1) | (1 << DISP2));
    PORTB &= ~((1 << DISP3) | (1 << DISP4) | (1 << DISP5) | (1 <<
       DISP6));

    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
    }
}

// **Update Display with Current Time (Improved Multiplexing)**
void displayTime() {
    uint8_t digits[6] = {
        hours / 10, hours % 10,
        minutes / 10, minutes % 10,
        seconds / 10, seconds % 10
    };

    static uint8_t currentDisplay = 0;

    setBCD(digits[currentDisplay]); // Send BCD to 7447
    selectDisplay(currentDisplay); // Activate one display
    currentDisplay = (currentDisplay + 1) % 6; // Cycle through displays
}

// **Software Timer to Keep Accurate Seconds (Non-blocking)**
void incrementTime() {
    static uint16_t counter = 0;

    counter++;
    if (counter >= 200) { // ~200 x 5ms = 1 second
        counter = 0;

        seconds++;
        if (seconds == 60) {
            seconds = 0;
```

```
            minutes++;

            if (minutes == 60) {
                minutes = 0;
                hours++;

                if (hours == 24) {
                    hours = 0;
                }
            }
        }
    }
}

// **Button Handling**
void checkButtons() {
    if (!(PINB & (1 << HOUR_BUTTON))) {
        _delay_ms(50);
        if (!(PINB & (1 << HOUR_BUTTON))) {
            hours = (hours + 1) % 24;
        }
    }
    if (!(PINC & (1 << MIN_BUTTON))) {
        _delay_ms(50);
        if (!(PINC & (1 << MIN_BUTTON))) {
            minutes = (minutes + 1) % 60;
        }
    }
    if (!(PINC & (1 << SEC_BUTTON))) {
        _delay_ms(50);
        if (!(PINC & (1 << SEC_BUTTON))) {
            seconds = (seconds + 1) % 60;
        }
    }
}

// **Main Function**
int main() {
    // **Set BCD and Display Pins as Outputs**
    DDRD |= (1 << BCD_A) | (1 << BCD_B) | (1 << BCD_C) | (1 << BCD_D);
    DDRD |= (1 << DISP1) | (1 << DISP2);
    DDRB |= (1 << DISP3) | (1 << DISP4) | (1 << DISP5) | (1 << DISP6);

    // **Set Push Buttons as Inputs with Pull-ups**
    DDRB &= ~(1 << HOUR_BUTTON);
    DDRC &= ~((1 << MIN_BUTTON) | (1 << SEC_BUTTON));
    PORTB |= (1 << HOUR_BUTTON);
    PORTC |= (1 << MIN_BUTTON) | (1 << SEC_BUTTON);
```

```
    // **Clock starts at 00:00:00**
    hours = 0;
    minutes = 0;
    seconds = 0;

    while (1) {
        checkButtons(); // Check button presses
        incrementTime(); // Update time if needed
        displayTime(); // Refresh the display continuously
        _delay_ms(5); // Short delay for proper multiplexing
    }
}
```

7 Code Explanation

7.1 Definitions and Setup

- The code starts by defining the CPU frequency and setting up the BCD output pins, display selection pins, and push buttons.
- The ”volatile” keyword is used for hours, minutes, and seconds to ensure proper time-keeping.

7.2 Functions

- setBCD(): Encodes the number into BCD format.
- selectDisplay(): Activates one of the six seven-segment displays.
- displayTime(): Updates the display continuously with the correct digits.
- incrementTime(): Maintains accurate time by incrementing seconds, minutes, and hours.
- checkButtons(): Allows manual time adjustment using push buttons.

7.3 Main Loop

- The main loop continuously checks for button presses, increments time, and updates the display.
- A delay of 5ms ensures proper multiplexing.

8 Advantages of Using AVR-GCC Over C++

Using AVR-GCC instead of C++ for implementing the digital clock provides several benefits:

- **Efficiency:** AVR-GCC generates highly optimized machine code, resulting in faster execution and reduced memory usage.

- **Precise Timing:** Direct control over hardware registers allows for more accurate timing, crucial for maintaining a stable clock display.
- **Lower Overhead:** Unlike C++ with its runtime overhead, AVR-GCC offers minimal abstraction, reducing unnecessary processing load.
- **Fine-Grained Hardware Control:** Allows direct manipulation of registers and ports, ensuring precise control over multiplexing and display updates.
- **Smaller Code Size:** Since AVR-GCC avoids C++ features like classes and dynamic memory allocation, the compiled code size remains smaller.
- **Better Debugging:** With direct register access, debugging at the hardware level becomes more transparent compared to C++ abstractions.

9 Precautions

9.1 Hardware

1. Always connect 220 Ω resistors in series with the seven-segment display segments to prevent excessive current draw and potential damage to the Arduino.
2. Double-check all the wirings on the breadoard to ensure no accidental short circuits, which could damage the microcontroller or display.
3. Before making any changes to the circuit, disconnect the Arduino from the power source to prevent accidental damage.

9.2 Software

1. Make sure that the delay in the multiplexing loop is optimized to avoid flickering or unreadable digits. Usually, a 1.5-5ms delay per digit makes things better.
2. Make sure that the correct pins are assigned to the display segments and common anodes before uploading the code.