Digital Clock

ArnavYadnopavit
EE24BTECH11007

March 24, 2025

1 Introduction

This report describes the implementation of a digital clock using an Arduino Uno board with a 7447 BCD decoder. The project uses a multiplexed seven-segment display to show the current time, which can be adjusted using pushbuttons.

2 Hardware Connections

The following table lists the hardware connections between the microcontroller and other components:

Microcontroller Pin	Connected Component	Function
D2-D5	7447 BCD Decoder	BCD Digit Selection
D6, D7	Digit Enable	Selects Active Display Digit
PB0-PB3	Digit Enable	Enables Different Digits
PB4	Hour Button	Increments Hours
PB5	Minute Button	Increments Minutes

Table 1: Pin Connections

3 Software Implementation

3.1 Timer Configuration

The clock utilizes Timer0 in CTC (Clear Timer on Compare Match) mode to generate a 1ms time base. This is essential for keeping track of time and handling display multiplexing.

Listing 1: Timer0 Configuration

```
TCCR0A = (1 << WGM01);   // CTC Mode
TCCR0B = (1 << CS01) | (1 << CS00);   // Prescaler 64
OCR0A = 249;   // 1ms interval
TIMSK0 |= (1 << OCIE0A);   // Enable Timer Interrupt
```

3.2 Multiplexed Display

Multiplexing is used to drive all six digits using fewer GPIO pins. Instead of having separate lines for each seven-segment display, only four BCD output lines (PD2-PD5) are used. Each digit is enabled sequentially using control signals (PD6, PD7, PB0-PB3). This rapid switching creates the illusion that all digits are continuously lit.

The multiplexing process involves:

- Setting the correct BCD value for a digit.
- Enabling only one digit at a time.
- Introducing a small delay before switching to the next digit.
- Repeating the cycle continuously to maintain a steady display.

Listing 2: Digit Multiplexing

```
void displayDigit(uint8_t digit, uint8_t position) {
    // Set the BCD output bits on PD2–PD5.
    PORTD = (PORTD & 0xC3) | ((digit & 0x0F) << 2);

    // Enable the appropriate digit.
    PORTD &= ~((1 << PD6) | (1 << PD7));
    PORTB &= ~((1 << PB0) | (1 << PB1) | (1 << PB2) | (1 << PB3));

    switch(position) {
        case 0: PORTD |= (1 << PD6); break;
        case 1: PORTD |= (1 << PD7); break;
        case 2: PORTB |= (1 << PB0); break;
        case 3: PORTB |= (1 << PB1); break;
        case 4: PORTB |= (1 << PB2); break;
        case 5: PORTB |= (1 << PB3); break;
    }

    _delay_ms(3);  // Short delay to reduce flicker.
}
```

3.3 Time Keeping

The system maintains time using a software counter that increments every second. The `millis()` function retrieves the elapsed time based on Timer0.

Listing 3: Time Keeping

```
uint32_t millis(void) {
    uint32_t ms;
    cli();
```

```
    ms = timer_millis;
    sei();
    return ms;
}
```

3.4 Button Handling with Debouncing

Push buttons are connected to PB4 and PB5 for adjusting hours and minutes. Since mechanical switches cause bouncing effects, a software debounce mechanism is implemented using a time delay.

Listing 4: Button Debouncing

```
void checkButtons(void) {
    bool hourState = (PINB & (1 << PB4)) != 0;
    bool minuteState = (PINB & (1 << PB5)) != 0;
    uint32_t currentMillis = millis();

    if (!hourState && lastHourState && ((currentMillis - lastHourPress) > debounc
        hours = (hours + 1) % 24;
        lastHourPress = currentMillis;
    }

    if (!minuteState && lastMinuteState && ((currentMillis - lastMinutePress) >
        minutes = (minutes + 1) % 60;
        lastMinutePress = currentMillis;
    }

    lastHourState = hourState;
    lastMinuteState = minuteState;
}
```

For codes refer to:
`https://github.com/ArnavYadnopavit/EE1003/tree/main/clock`

Thank you