Digital Clock using AVR-GCC through Arduino

EE1003: Scientific Programming for Electrical Engineers

Y. Harsha Vardhan Reddy EE24BTECH11063

Contents

1 Introduction

This document explains the implementation of a digital clock using an AVR microcontroller. The system uses a BCD to 7-segment decoder (7447 IC) and six 7-segment displays to show the time in hours, minutes, and seconds. The clock includes functionalities for setting the time using increment and decrement buttons.

2 Hardware Components

The digital clock is built using the following hardware components:

- AVR microcontroller (operating at 16MHz)
- Six 7-segment displays (common anode)
- BCD to 7-segment decoder (74LS47)
- Four push buttons (for incrementing and decrementing hours and minutes)

3 Pin Configuration

Component	**Pins**	**Function**
BCD Outputs	PD2-PD5 (PORTD)	Connect to 74LS47 inputs A, B, C, D
Common Anodes	PC0-PC5 (PORTC)	Control which display is active
Hour Increment Button	PD6	Increase hours
Minute Increment Button	PD7	Increase minutes
Hour Decrement Button	PB0	Decrease hours
Minute Decrement Button	PB1	Decrease minutes

Table 1: Pin Configuration

4 Hardware Operation

The system operates using the following principles:

4.1 Multiplexed Display

Since we are using six 7-segment displays but only one BCD-to-7-segment decoder, we implement time-division multiplexing to control all six displays. This technique involves:

1. Rapidly switching between displays (only one display is active at a time)
2. Setting the appropriate BCD value for the active display
3. Creating the illusion that all displays are simultaneously active due to persistence of vision

4.2 BCD Output

The 7447 IC(7-segment decoder) translates 4-bit BCD values into the appropriate segment patterns to display decimal digits (0-9). The four BCD pins connect from PORTD (PD2-PD5) to the 7447 IC's inputs.

4.3 User Input

Four push buttons allow the user to set the time:

- Two buttons for incrementing hours and minutes
- Two buttons for decrementing hours and minutes

5 Software Implementation

5.1 Time Representation

The time is represented in BCD (Binary-Coded Decimal) format, where each decimal digit is encoded using 4 bits. This approach simplifies displaying the digits on 7-segment displays.

Listing 1: Time Variables

```
// Time variables (BCD format)
volatile int seconds = 0x00, minutes = 0x30, hours = 0x15;
```

In this BCD representation:

- `0x15` represents 15 hours (1 in tens place, 5 in ones place)
- `0x30` represents 30 minutes (3 in tens place, 0 in ones place)
- `0x00` represents 00 seconds

5.2 System Initialization

The setup function initializes the microcontroller:

Listing 2: System Initialization

```
void setup() {
    // Configure BCD pins (PD2-PD5) as outputs
    BCD_DDR |= BCD_MASK;
    BCD_PORT &= ~BCD_MASK; // Initialize BCD output to LOW

    // Configure common anode control (PC0-PC5) as outputs
    COMMON_DDR = 0xFF;
    COMMON_PORT = 0x00; // Turn off all displays

    // Configure button pins as inputs with pull-ups
```

```
    PORTD |= (1 << HOUR_INC_PIN) | (1 << MIN_INC_PIN); // Enable
       internal pull-ups
    DDRB &= ~((1 << HOUR_DEC_PIN) | (1 << MIN_DEC_PIN)); // Set as
       inputs
    PORTB |= (1 << HOUR_DEC_PIN) | (1 << MIN_DEC_PIN); // Enable
       internal pull-ups

    // Timer1 setup for 1-second interrupt
    TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10); // CTC mode,
        Prescaler = 1024
    OCR1A = 0x3D08; // Compare match value for 1Hz (1 sec) - 16MHz
       /1024/1Hz - 1
    TIMSK1 |= (1 << OCIE1A); // Enable Timer1 compare interrupt

    sei(); // Enable global interrupts
}
```

Key initialization tasks:

- Configure output pins for the BCD decoder and common anode control
- Set up input pins with internal pull-ups for the buttons
- Configure Timer1 in CTC (Clear Timer on Compare) mode to generate an interrupt every second
- Enable global interrupts

5.3 Timer Interrupt for Clock Timekeeping

The system uses Timer1 in CTC mode to generate an interrupt every second. This interrupt increments the time and handles the BCD adjustments:

Listing 3: Timer Interrupt Service Routine

```
ISR(TIMER1_COMPA_vect) {
    // Increment seconds
    seconds++;
    // Adjust if lower digit > 9
    if ((seconds & 0x0F) > 0x09) {
        seconds = (seconds & 0xF0) + 0x10;
    }
    // Check for 60 seconds
    if (seconds >= 0x60) {
        seconds = 0;

        // Increment minutes
        minutes++;
        // Adjust if lower digit > 9
        if ((minutes & 0x0F) > 0x09) {
```

```
            minutes = (minutes & 0xF0) + 0x10;
        }
        // Check for 60 minutes
        if (minutes >= 0x60) {
            minutes = 0;

            // Increment hours
            hours++;
            // Adjust if lower digit > 9
            if ((hours & 0x0F) > 0x09) {
                hours = (hours & 0xF0) + 0x10;
            }
            // Check for 24 hours
            if (hours >= 0x24) {
                hours = 0;
            }
        }
    }
}
```

The interrupt service routine:

- Increments seconds, minutes, and hours
- Performs BCD adjustment when necessary
- Handles overflow conditions (60 seconds, 60 minutes, 24 hours)

5.4 Display Multiplexing

The displayTime function implements the multiplexing logic for the six 7-segment displays:

Listing 4: Display Multiplexing Function

```
void displayTime() {
    // Extract individual digits correctly
    int digits[0x06] = {
        (hours >> 0x04) & 0x0F,     // Hours tens digit
        hours & 0x0F,               // Hours ones digit
        (minutes >> 0x04) & 0x0F,   // Minutes tens digit
        minutes & 0x0F,             // Minutes ones digit
        (seconds >> 0x04) & 0x0F,   // Seconds tens digit
        seconds & 0x0F              // Seconds ones digit
    };

    for (int i = 0; i < 0x06; i++) {
        COMMON_PORT = (1 << i);  // Activate current display
        setBCD(digits[i]);       // Send BCD value to 7447
        _delay_ms(0x02);            // Short delay for persistence
           of vision
```

```
        COMMON_PORT = 0;          // Deactivate all displays
    }
}
```

This function:

- Extracts individual digits from hours, minutes, and seconds
- Cycles through each display, activating only one at a time
- Sends the appropriate BCD value to the decoder
- Maintains a short delay for persistence of vision

5.5 BCD Output Function

The setBCD function outputs the BCD value to the appropriate pins:

Listing 5: BCD Value Output

```
void setBCD(int value) {
    BCD_PORT = (BCD_PORT & ~BCD_MASK) | ((value << 2) & BCD_MASK);
}
```

This function:

- Clears the BCD pins using the mask
- Sets the new value shifted by 2 positions (to map to PD2-PD5)

5.6 Button Input Handling

The system checks for button presses to adjust the time:

Listing 6: Button Input Handling

```
void checkButtons() {
    // Check increment buttons
    bool isHourIncPressed = !(PIND & (1 << HOUR_INC_PIN));
    bool isMinuteIncPressed = !(PIND & (1 << MIN_INC_PIN));

    // Check decrement buttons
    bool isHourDecPressed = !(PINB & (1 << HOUR_DEC_PIN));
    bool isMinuteDecPressed = !(PINB & (1 << MIN_DEC_PIN));

    // Handle hour increment button
    if (isHourIncPressed) {
        _delay_ms(50); // Debounce delay
        isHourIncPressed = !(PIND & (1 << HOUR_INC_PIN)); // Check
           again
        if (isHourIncPressed) {
```

```
            hours++;
            if ((hours & 0x0F) > 0x09) {
                hours = (hours & 0xF0) + 0x10;
            }
            if (hours >= 0x24) { // Reset at 24
                hours = 0;
            }
            seconds = 0;
            while (!(PIND & (1 << HOUR_INC_PIN))); // Wait for
               release
        }
    }

    // Similar logic for minute increment and
    // hour/minute decrement buttons...
}
```

Key aspects of button handling:

- Includes debounce delay (50ms) to prevent false readings
- Adjusts time when buttons are pressed
- Resets seconds to zero when time is adjusted
- Waits for button release to prevent multiple adjustments

5.7 Decrement Functions

Special functions handle the decrementation of hours and minutes, which is more complex due to BCD borrowing:

Listing 7: Hour and Minute Decrement Functions

```
void decrementHours() {
    if (hours == 0) {
        hours = 0x23; // Wrap around to 23 when decrementing from 0
    } else {
        if ((hours & 0x0F) == 0x00) {
            // If the lower nibble is 0, borrow from the upper
               nibble
            hours = (hours - 0x10) | 0x09;
        } else {
            // Simply decrement if no BCD borrowing is needed
            hours--;
        }
    }
}

void decrementMinutes() {
```

```
    if (minutes == 0) {
        minutes = 0x59; // Wrap around to 59 when decrementing from
           0
    } else {
        if ((minutes & 0x0F) == 0x00) {
            // If the lower nibble is 0, borrow from the upper
               nibble
            minutes = (minutes - 0x10) | 0x09;
        } else {
            // Simply decrement if no BCD borrowing is needed
            minutes--;
        }
    }
}
```

These functions handle:

- Wrap-around from 0 to the maximum value (23 for hours, 59 for minutes)
- BCD borrowing logic when the lower digit is 0

5.8 Main Function

The main function orchestrates the entire system:

Listing 8: Main Function

```
int main() {
    setup();
    while (true) {
        checkButtons(); // Read push-buttons
        displayTime(); // Refresh display continuously
    }
}
```

6 Working Principle Summary

1. The system initializes all required peripherals and timers.
2. Timer1 generates an interrupt every second to update the time.
3. The main loop continuously:
    - Checks for button inputs to adjust the time
    - Updates the displays using multiplexing
4. The displayTime function cycles through all six displays rapidly, creating the illusion that all displays are active simultaneously.

5. The BCD to 7-segment decoder converts 4-bit BCD values to appropriate segment patterns.

7 Challenges and Solutions

7.1 BCD Representation

Working with BCD format requires careful handling when incrementing or decrementing values. Each digit must be checked and adjusted when it exceeds the valid range (0-9).

7.2 Multiplexed Display

Multiplexing requires precise timing to ensure all displays appear active simultaneously. The code includes a 2ms delay per display, striking a balance between refresh rate and display stability.

7.3 Button Debouncing

Mechanical buttons typically bounce, causing multiple readings for a single press. The code implements a simple debounce mechanism with a 50ms delay and re-checking the button state.

8 Conclusion

This digital clock implementation demonstrates several important concepts in embedded systems:

- Time-division multiplexing for controlling multiple displays
- BCD representation and manipulation
- Interrupt-based timekeeping
- Button debouncing
- User input handling

The system provides a complete 24-hour clock with hour and minute adjustment capabilities using both increment and decrement buttons, making it a practical and user-friendly device.