Digital Clock Using AVR-GCC

Sruthi Bijili
EE24BTECH11060

March 24, 2025

1 Objective

To design and implement a digital clock using an Arduino board and a seven-segment display, programmed using AVR-GCC.

2 Introduction

This experiment demonstrates the implementation of a digital clock using an Arduino board. The timekeeping is achieved through a timer interrupt, while multiplexed seven-segment displays are used to show the time in HH:MM:SS format.

3 Apparatus

- bread board
- Seven-segment display (Common Anode)
- Resistors
- Arduino UNO
- power supply
- Connecting Wires

4 Connections

- Place the six seven-segment displays on the breadboard. The first two are used to display the two digits of hours, the next two for minutes, and the next two for seconds.
- Connect the 5V and GND pins from the Arduino to the power rails of the breadboard.
- Connect the COM of each display to the 5V rail through resistors.
- Connect the dot of each display to GND.
- Make connections from the Arduino pins to the seven-segment displays as per the table below.

Arduino Pin	**Seven- Segment Display**
2	a
3	b
4	c
5	d
6	e
7	f
8	g
9	COM of 1st Hours Display
10	COM of 2nd Hours Display
11	COM of 1st Minutes Display
12	COM of 2nd Minutes Display
A0	COM of 1st Seconds Display
A1	COM of 2nd Seconds Display

Table 1: Connections between Arduino pins and seven-segment displays

5 AVR Code

```
// setting cpu frequency to 16 mega hz (for atmega328p)
// atmega328p microcontroller is in the arduino
// frequency is needed for timing calculations
#define F_CPU 16000000UL
```

```
// including required libraries
// they provide access to hardware registers, interrupts, delays
#include <avr/io.h> // standard input-output functions for avr
#include <avr/interrupt.h> // interrupt handling
#include <util/delay.h> // delay functions

// array to store bit patterns for each digit
// low (0) turns on the led segment
const uint8_t digit_map[] = {
    0b00000000,  // 0
    0b11100100,  // 1
    0b10010000,  // 2
    0b11000000,  // 3
    0b01100100,  // 4
    0b01001000,  // 5
    0b00001000,  // 6
    0b11100000,  // 7
    0b00000000,  // 8
    0b01000000   // 9
};

// defining time variables
volatile uint8_t hours = 5, minutes = 11, seconds = 10;

// array to store display digits
uint8_t digits[6];
// function to calculate digit values for display
void update_digits() {
    digits[0] = hours / 10;   // first display
    digits[1] = hours % 10;   // second display
    digits[2] = minutes / 10; // third display
    digits[3] = minutes % 10; // fourth display
    digits[4] = seconds / 10; // fifth display
    digits[5] = seconds % 10; // sixth display
}

// function to update time at every second
void update_time() {
    seconds++;
    if (seconds >= 60) { seconds = 0; minutes++; }
    if (minutes >= 60) { minutes = 0; hours++; }
```

```
    if (hours >= 24) { hours = 0; } // 24-hour clock

    // update digits array after time update
    update_digits();
}

// interrupt service routine (ISR) for Timer1 compare match A
// calls update_time() function every second
ISR(TIMER1_COMPA_vect) {
    update_time();
}

// function to display a single digit (multiplexing)
void display_digit(uint8_t display, uint8_t digit) {
    PORTB &= ~(0b00011110); // turn off PB1-PB4
    PORTC &= ~(0b00000011); // Turn off PC0-PC1

    PORTD = digit_map[digit];  // set segments a-f
     // control segment G (PB0)
    if (digit == 0 || digit == 1 || digit == 7) {
        PORTB |= (1 << PB0); // turn on g segment
    } else {
        PORTB &= ~(1 << PB0); // turn off g segment
    }

    // enable the correct digit select pin
    if (display < 4) {
        // PB1-PB4 for first to fourth displays
        PORTB |= (1 << (display + 1));
    } else {
        // PC0-PC1 for fifth and sixth displays
        PORTC |= (1 << (display - 4));
    }

    // ensuring the digit is viible before switching
    _delay_ms(2);
}

int main(void) {
    // setting PD2-PD7 as output for segments a-f
    DDRD |= 0b11111100;
```

```
    // setting PB0 as output for segment g
    DDRB |= (1 << PB0);
    // setting PORTB (PIN 9-12) and PORTC (A0-A1) as output
    // (for digit selection)
    DDRB |= (1 << PB1) | (1 << PB2) | (1 << PB3) | (1 << PB4);
    DDRC |= (1 << PC0) | (1 << PC1);

    // initialize display digits before Timer starts
    update_digits();
      // set-up Timer1 (1 hz interrupt)
    TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10);
    OCR1A = 15625;  // compare value for 1-second tick
    TIMSK1 |= (1 << OCIE1A);  // enable Timer1 compare interrupt
    sei();  // enable global interrupts

    while (1) {
        for (uint8_t i = 0; i < 6; i++) {
            // cycle through all 6 digits
            display_digit(i, digits[i]);
        }
    }
}
```

6 Code Explanation

1. The program sets the CPU frequency to **16 MHz** and includes necessary AVR libraries for **I/O operations, interrupts, and delays**.
2. A **digit map array** is defined to store bit patterns for displaying numbers (0-9) on a **7-segment display**.
3. Variables `hours`, `minutes`, and `seconds` store the current time, while an array `digits[6]` holds individual digits for display.
4. The function `update_digits()` extracts **each digit** from `hours`, `minutes`, and `seconds` to update the display.
5. The function `update_time()` increments the **time every second**, resetting values when they reach **60 (minutes/seconds) or 24 (hours)**.
6. **Timer1 is configured** to generate an interrupt **every second**, calling `update_time()` automatically.

7. The **display uses multiplexing**, where only **one digit is turned on at a time** using `display_digit()`, reducing the number of required pins.
8. **PORTB, PORTC, and PORTD** are set as **output pins** to control the **segments and digit selection** for the display.
9. The **main loop continuously refreshes** the 7-segment display by cycling through all **6 digits rapidly**.
10. The system runs indefinitely, updating time **every second** and displaying it using **time-sharing multiplexing**.

7 Results

- The clock successfully keeps time, updating every second.
- The multiplexed display correctly shows the HH:MM:SS format.
- Timer1 ISR ensures accurate timekeeping with minimal drift.

8 Conclusion

The experiment successfully demonstrates a working digital clock using an Arduino board and seven-segment displays. By leveraging timer interrupts and display multiplexing, an efficient and accurate timekeeping system is achieved.

Acknowledgement: code sourced from M srujana, EE24BTECH11060