# Digital Clock Using AVR-GCC

Mandara Hosur
EE24BTECH11040

March 21, 2025

**Abstract**

This report presents the design and implementation of a digital clock using ATmega328p microcontroller (in Arduino) and six 7-segment displays. The system utilises Timer1 interrupts to update the time while implementing multiplexing for efficient display control. The program is written in C using AVR-GCC and allows manual setting of hours, minutes, and seconds through predefined values in the code.

## 1 Introduction

Digital clocks are widely used in embedded systems for real-time applications. This project implements a digital clock using six 7-segment displays. The clock allows us to set hours, minutes, and seconds manually and updates the display accordingly. It uses multiplexing to drive the six 7-segment displays and a timer interrupt to keep track of time.

## 2 Hardware Components

The following components are used:

- Six 7-segment displays
- Six resistors (220Ω)
- Arduino UNO
- Breadboard

- Jumper wires
- Power source (cell phone)

# 3 Circuit Design

1. Attach the six 7-segment displays onto the breadboard. Left to right, let them be named first, second, third, fourth, fifth, and sixth.
2. The first two displays are for hours, the next two for minutes, and the last two for seconds.
3. Connect all the displays to the Arduino using the connections specified in Table 1.
4. Connect 5V and GROUND pins from the Arduino onto the breadboard.
5. Connect the COM of each display to 5V via a resistor.

Table 1: Connections to Arduino

| Arduino Pin | 7-Segment Display's Pin |
|---|---|
| 2 | a |
| 3 | b |
| 4 | c |
| 5 | d |
| 6 | e |
| 7 | f |
| 8 | g |
| 9 | COM of first 7-segment display |
| 10 | COM of second 7-segment display |
| 11 | COM of third 7-segment display |
| 12 | COM of fourth 7-segment display |
| A0 | COM of fifth 7-segment display |
| A1 | COM of sixth 7-segment display |

# 4 Software Implementation

The software is written in AVR-C and compiled using AVR-GCC. The program allows manual time setting rather than real-time tracking.

## 4.1 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
```

```c
    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 visible 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]);
        }
    }
}
```

***Remark:*** *Code sourced from M. Srujana, EE24BTECH11042*

# 5 Results

The clock successfully displays the manually set time on the 7-segment displays. The multiplexing technique ensures efficient digit control without flickering.

# 6 Conclusion

This project demonstrates the use of AVR-GCC and microcontroller timers for implementing a digital clock. The current implementation requires manual time setting.