Digital Clock

S.Kavya Anvitha
Electrical Engineering
IITH

March 22, 2025

Abstract

This report presents the design and implementation of a digital clock using an Arduino microcontroller, six seven-segment displays, and resistors. The objective of this project is to develop a functional timekeeping device that accurately displays hours, minutes, and seconds in real-time.

The Arduino serves as the core controller, managing the display updates and timekeeping operations. The six seven-segment displays are used to show the time, with appropriate wiring and control logic to minimize power consumption and optimize visibility. Resistors are incorporated for current limiting and protection of the components. The clock is programmed using the Arduino platform, ensuring precise time tracking and smooth transitions between digits.

Contents

1 Introduction

A digital clock is an essential timekeeping device that displays time in numerical format, making it easy to read and interpret. This report presents the design, construction, and working principle of a digital clock, explaining both hardware and software aspects.

2 Hard Ware components

- Six 7-segment displays
- Six resistors (220Ω)
- Breadboard
- Arduino UNO
- Jumper wires
- Power source (Phone)

3 Circuit Configuration

1. Attach the six 7-segment displays onto the breadboard,Left to right.
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 pin 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 Code Execution

The code 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

```
#define F_CPU 16000000UL  // Set CPU frequency to 16MHz (for ATmega328p

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

// Lookup table for 7-segment display (Common Anode) (A-F segments only
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
};

// Time variables
volatile uint8_t hours = 1, minutes = 11, seconds = 30;
uint8_t digits[6];
```

```c
// Function to update digit values for display
void update_digits() {
    digits[0] = hours / 10;   // H1
    digits[1] = hours % 10;   // H2
    digits[2] = minutes / 10; // M1
    digits[3] = minutes % 10; // M2
    digits[4] = seconds / 10; // S1
    digits[5] = seconds % 10; // S2
}

// Function to update time
void update_time() {
    seconds++;
    if (seconds >= 60) { seconds = 0; minutes++; }
    if (minutes >= 60) { minutes = 0; hours++; }
    if (hours >= 24) { hours = 0; }

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

// Timer1 Interrupt (1 second delay)
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 all PB digit selects (PB1-PB4)
    PORTC &= ~(0b00000011); // Turn off all PC digit selects (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) {
        PORTB |= (1 << (display + 1));  // PB1-PB4 for H1-H2-M1-M2
    } else {
        PORTC |= (1 << (display - 4));  // PC0-PC1 for S1-S2
    }

    _delay_ms(2); // Persistence for multiplexing
}

int main(void) {
    // Set PD2-PD7 as output for segments A-F
    DDRD |= 0b11111100;
    // Set PB0 as output for segment G
    DDRB |= (1 << PB0);
    // Set PORTB (PIN 9-12) and PORTC (A0-A1) as output for digit selec
    DDRB |= (1 << PB1) | (1 << PB2) | (1 << PB3) | (1 << PB4);
    DDRC |= (1 << PC0) | (1 << PC1);

    // Initialize display digits before Timer starts
    update_digits();

    // Setup Timer1 (1Hz interrupt)
    TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10); // CTC mode, pr
    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++) {
            display_digit(i, digits[i]); // Cycle through all 6 digits
        }
    }
}
```

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.