DIGITAL CLOCK

By

K. AKHIL - EE24BTECH11035

March 24, 2025

Contents

I. Required Components

- Breadboard
- Arduino UNO
- Jumper Cables
- Resistors
- Seven segment Displays
- 7447 BCD Decoder

II. Hardware Connections

Component	ATmega328P Pin	Connection Description
BCD Input A	Digital Pin 2	Connected to 7447 A input
BCD Input B	Digital Pin 3	Connected to 7447 B input
BCD Input C	Digital Pin 4	Connected to 7447 C input
BCD Input D	Digital Pin 5	Connected to 7447 D input
COM (Tens Place)	Analog Pin A3 (PC3)	Common pin for 7-segment (Tens)
COM (Units Place)	Analog Pin A4 (PC4)	Common pin for 7-segment (Units)
7-Segment Display	7447 Output	7447 drives segments

TABLE 0
Pin Connections for 7-Segment Display with 7447 and ATmega328P

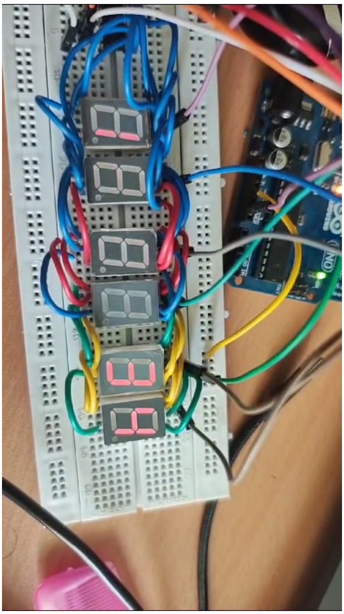

Fig. 0.1.

III. Working Explanation

A. Initialization of I/O and Timer

The ATmega328P microcontroller initializes the required I/O pins and configures Timer1 for interrupt-driven time updates.

- The BCD pins (A, B, C, D) are configured as outputs to send data to the 7447 decoder.
- The common anodes of the six seven-segment displays are connected to separate Arduino analog pins.
- Timer1 is configured to trigger an interrupt every 1 second to increment time.

B. Displaying Time Using Multiplexing

The clock utilizes multiplexing to drive six seven-segment displays efficiently while minimizing the number of required I/O pins.

1) The digits for hours, minutes, and seconds are extracted from the stored time values using bitwise operations.
2) The appropriate BCD values are sent to the 7447 decoder.
3) Only one display is activated at a time using the corresponding common anode pin.
4) A short delay ensures proper visibility before switching to the next digit.
5) This rapid switching occurs continuously, giving the illusion that all digits are displayed simultaneously.

C. Time Keeping and Increment Logic

The microcontroller stores time values in BCD format, ensuring efficient calculations and display updates.

- The seconds value increments every time the Timer1 interrupt triggers.
- If seconds reach 60, they reset to 00, and the minutes value increments.
- If minutes reach 60, they reset to 00, and the hours value increments.
- If hours reach 24, they reset to 00, completing a full-day cycle.

D. Timer1 Interrupt for Precise Timing

Timer1 is configured in **Clear Timer on Compare Match (CTC) Mode** to generate precise 1-second interrupts.

1) The Timer1 interrupt is triggered every second using a compare match value calculated for a 16MHz clock.
2) When the interrupt occurs, the seconds counter increments.
3) If a carry-over condition is met, minutes and hours are updated accordingly.
4) The updated time values are sent to the display in the next iteration.

E. Main Loop Execution

The ‘main()‘ function continuously updates the display while the Timer1 interrupt manages time increments in the background.

- The main loop does not block execution with delays, ensuring smooth operation.
- Display updates occur independently of the timekeeping logic, preventing flickering or lag.
- The system can be expanded to include additional functionality, such as setting the time using push buttons.

IV. Code Outline Explanation

Defining CPU Frequency

```
#define F_CPU 16000000UL
```

The microcontroller runs at 16 MHz. This definition ensures proper timing calculations.

Including Required Headers

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

These headers provide functions for I/O operations, interrupt handling, and delays.

Defining BCD and Display Control Pins

```
#define A PD2
#define B PD3
#define C PD4
#define D PD5

#define H1 PD6
#define H2 PD7
#define M1 PB0
#define M2 PB1
#define S1 PB2
#define S2 PB3
```

- **PD2âPD5** are connected to the 7447 BCD decoder. - **PD6, PD7, PB0âPB3** control which digit is active.

Clock Variables in BCD Format

```
volatile uint8_t hours = 0b00010010;
volatile uint8_t minutes = 0b00000000;
volatile uint8_t seconds = 0b00000000;
```

The time is stored in **Binary-Coded Decimal (BCD)** format.

Displaying a Single Digit

```
void displayDigit(uint8_t digit) {
    PORTD = (PORTD & 0b11000011) | (digit << 0b00000010);
}
```

This function sends a **BCD digit** to **PORTD (PD2âPD5)** while preserving other bits.

Displaying the Complete Time

```
void displayTime() {
    uint8_t h1 = (hours >> 4) & 0x0F;
    uint8_t h2 = hours & 0x0F;
    uint8_t m1 = (minutes >> 4) & 0x0F;
    uint8_t m2 = minutes & 0x0F;
    uint8_t s1 = (seconds >> 4) & 0x0F;
    uint8_t s2 = seconds & 0x0F;

    PORTD |= (1 << H1); displayDigit(h1); _delay_ms(5); PORTD &= ~(1 << H1);
    PORTD |= (1 << H2); displayDigit(h2); _delay_ms(5); PORTD &= ~(1 << H2);
    PORTB |= (1 << M1); displayDigit(m1); _delay_ms(5); PORTB &= ~(1 << M1);
    PORTB |= (1 << M2); displayDigit(m2); _delay_ms(5); PORTB &= ~(1 << M2);
    PORTB |= (1 << S1); displayDigit(s1); _delay_ms(5); PORTB &= ~(1 << S1);
    PORTB |= (1 << S2); displayDigit(s2); _delay_ms(5); PORTB &= ~(1 << S2);
}
```

- Extracts each **tens** and **units** place digit using bitwise operations. - Uses **multiplexing** to display each digit sequentially.

Timer Interrupt for 1-Second Updates

```
ISR(TIMER1_COMPA_vect) {
    seconds += 1;
    if ((seconds & 0x0F) > 9) {
        seconds = (seconds & 0xF0) + 0x10;
    }
    if (seconds >= 0x60) {
        seconds = 0x00;
        minutes += 1;
    }
    if ((minutes & 0x0F) > 9) {
        minutes = (minutes & 0xF0) + 0x10;
    }
    if (minutes >= 0x60) {
        minutes = 0x00;
        hours += 1;
    }
    if ((hours & 0x0F) > 9) {
        hours = (hours & 0xF0) + 0x10;
```

```
    }
    if (hours >= 0x24) {
        hours = 0x00;
    }
}
```

- Increments **seconds** every timer interrupt (1 second). - **Handles carry propagation** from seconds â minutes â hours.

Timer1 Configuration

```
void timer1_init() {
    TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10);
    OCR1A = 15624;
    TIMSK1 |= (1 << OCIE1A);
    sei();
}
```

- **Sets Timer1 to CTC mode**. - **Prescaler 1024** â Results in **1-second intervals**. - Enables **interrupts** on compare match.

Main Function

```
int main(void) {
    DDRD |= (1 << A) | (1 << B) | (1 << C) | (1 << D);
    DDRD |= (1 << H1) | (1 << H2);
    DDRB |= (1 << M1) | (1 << M2) | (1 << S1) | (1 << S2);

    timer1_init();

    while (1) {
        displayTime();
    }
}
```

- **Configures I/O pins** for **BCD outputs** and **digit control**. - **Starts Timer1** for automatic timekeeping. - **Continuously updates the display** in an infinite loop.

V. Conclusion

This project successfully implements a digital clock using an ATmega328P microcontroller, seven-segment displays, and a 7447 BCD to 7-segment decoder. The clock accurately displays time in the HH:MM:SS format by utilizing a hardware timer interrupt for precise timekeeping.

Key takeaways from this project include:

- Efficient multiplexing: Controlling multiple 7-segment displays using limited I/O pins.
- Precise timing: Achieved using Timer1 interrupt to increment seconds every second.
- BCD-based storage: Ensuring correct representation of time without complex conversions.
- Modularity and expandability: The system can be modified for real-time clock (RTC) modules or additional features like alarms.

This project demonstrates how microcontrollers can be used for real-time applications, providing a foundation for more advanced embedded systems. Future improvements may include RTC integration, battery backup, or OLED/LCD display support for enhanced functionality.

VI. References

- AI Suggestions
- Code by Niketh Achanta.
- Hardware connections guide- Online Sites