Digital Clock

EE24BTECH11052 - Rongali Charan

Contents

1 Aim

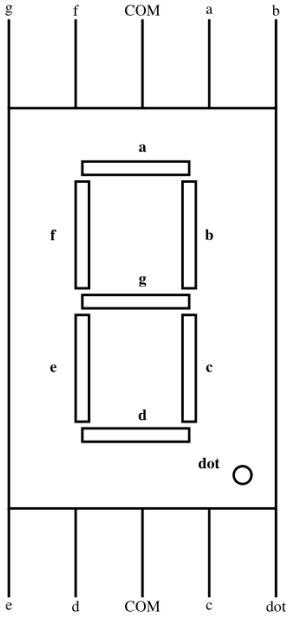
Constructing the digital clock without using flipflops and using avr-gcc

2 Materials Required

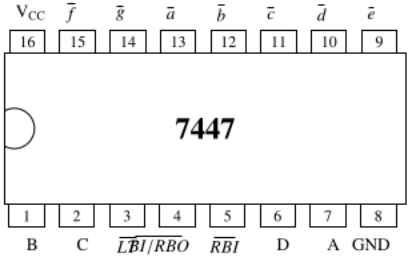
- 6x Common-anode 7-segment displays
- 7447 BCD-to-7-segment decoder
- 6x 180Ω resistors (for common anode current limiting)
- Jumper wires
- Bread Board
- Arduino UNO(atmega328p) and connecting cable
- Jumper wires for arduino and normal wires for seven segment displays
- Push Buttons

3 Hardware Setup

The wiring process involves connecting the 7447 decoder, segment lines, and common anodes to the Arduino pins.



(a) Seven Segment



(b) 7447 Decoder

3.1 7447 Connections

The table below shows the connections between the 7447 decoder and the Arduino Uno:

TABLE 0: 7447 to Arduino Pin Connections

7447 Pin	Arduino Connection
VCC (Pin 16)	+5V
GND (Pin 8)	GND
A (Pin 7)	D2 (BCD input LSB)
B (Pin 1)	D3
C (Pin 2)	D4
D (Pin 6)	D5 (BCD input MSB)

3.2 Segment Connections

The 7447 outputs are connected to the corresponding segments of all six displays, as shown in the table below:

TABLE 0: 7447 to Display Segment Connections

7447 Pin	Display Segment
Pin 13	Segment ”a”
Pin 12	Segment ”b”
Pin 14	Segment ”c”
Pin 15	Segment ”d”
Pin 9	Segment ”e”
Pin 10	Segment ”f”
Pin 11	Segment ”g”

3.3 Common Anode Connections

Each display’s common anode is connected to an Arduino analog pin through a 180Ω resistor to limit current flow.

TABLE 0: Common Anode Connections

Display	Arduino Analog Pin (via 180Ω resistor)
Display 1	A0
Display 2	A1
Display 3	A2
Display 4	A3
Display 5	A4
Display 6	A5

4 Key Design Notes

4.1 Current Limiting

The 180Ω resistors on the common anodes limit the total current per digit. However, with all segments lit, each digit draws approximately:

$$I = 7 \times 15\text{mA} = 105\text{mA}$$

Since the Arduino analog pins are rated for 20mA max, prolonged operation could damage the board. Use brief testing sessions or add transistors to handle the higher current safely.

4.2 Multiplexing

The multiplexing ensures only one digit is active at a time. With a 2ms delay per digit, the full refresh cycle takes:

$$6 \times 2\text{ms} = 12\text{ms}$$

Brightness can be adjusted by modifying the delay value in the code.

4.3 7447 Behavior

The 7447 outputs are active-low, which means the segments turn on when the decoder outputs are LOW. This suits common-anode displays, which require low signals to activate the segments.

5 Testing and Troubleshooting

5.1 Common Issues

TABLE 0: Troubleshooting Guide

Problem	Solution
Dim segments	Reduce resistor value (e.g., 100Ω)
Arduino overheating	Limit testing time or use transistors
Incorrect digits	Verify BCD wiring order

6 Code Implementation

Key Features

- **Real-Time Interrupt-Driven Operation**: Timer1 is configured to trigger 1-second updates for consistent time management.
- **Interrupt Service Routine (ISR)**: Handles asynchronous updates to the Clock, Timer, and Stopwatch modes.
- **Multiplexed Display Control**: Efficient hardware management using binary-coded decimal (BCD) encoding and multiplexing.
- **Debounced Buttons**: Smooth user interaction for adjusting time settings and controlling modes.
- **Dynamic Mode Switching**: Seamlessly switches between Clock, Timer, and Stopwatch functionality using a mode variable.

ISR Overview

The **Interrupt Service Routine (ISR)** is the backbone of the program, ensuring precise real-time operations. The Timer1 interrupt triggers every second, enabling reliable timekeeping and supporting the following functionalities:

- **Clock Mode**: Implements a simple incrementing mechanism using **modulo arithmetic**, where seconds roll over after 60, minutes after 60, and hours after 24.

- **Timer Mode**: Uses a nested logic structure to decrement seconds, minutes, and hours. Special conditions ensure smooth transitions when seconds or minutes reach zero.
- **Stopwatch Mode**: Implements incrementing logic with rollover at 60 seconds and 60 minutes, similar to the Clock Mode. This is done to track elapsed time efficiently.
- **Debugging with PC7**: Toggles the **debugging LED** to ensure the ISR is running correctly and at expected intervals.

Hardware Overview

- **Registers**: `TCCR1B`, `OCR1A`, `PORTD`, `PORTC`, and `PINB`.
- **7-Segment Display**: Controlled using binary-coded decimal (BCD) encoding.
- **Buttons**: Connected to `PD6`, `PD7`, and `PB0/PB1` for mode switching and adjustments.

Code Snippets

Timer Setup

```
TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10);
OCR1A = 15625; // 1-second interrupt
TIMSK1 |= (1 << OCIE1A);
sei(); // Enable global interrupts
```

Interrupt Service Routine (ISR)

```
ISR(TIMER1_COMPA_vect) {
    // Toggle a debug LED to check ISR functionality
    PORTC ^= (1 << 7);

    // Clock Mode Logic
    if (mode == 0) {
        seconds++;
        if (seconds == 60) {
            seconds = 0;
            minutes++;
            if (minutes == 60) {
                minutes = 0;
                hours = (hours + 1) % 24;
            }
        }
    }

    // Timer Countdown Logic
    if (mode == 1 && stopwatch_running) {
        if (timer_seconds > 0 || timer_minutes > 0 || timer_hours > 0) {
            if (timer_seconds == 0) {
                if (timer_minutes > 0) {
                    timer_minutes--;
                    timer_seconds = 59;
                } else if (timer_hours > 0) {
                    timer_hours--;
                    timer_minutes = 59;
```

```
                timer_seconds = 59;
            }
        } else {
            timer_seconds--;
        }
    }
}

// Stopwatch Increment Logic
if (mode == 2 && stopwatch_running) {
    stopwatch_seconds++;
    if (stopwatch_seconds == 60) {
        stopwatch_seconds = 0;
        stopwatch_minutes++;
        if (stopwatch_minutes == 60) {
            stopwatch_minutes = 0;
            stopwatch_hours = (stopwatch_hours + 1) % 24;
        }
    }
}
}
```

Multiplexed Display Control

```
void displayTime() {
    int digits[6];

    if (mode == 0) { // Clock Mode
        digits[0] = hours / 10;
        digits[1] = hours % 10;
        digits[2] = minutes / 10;
        digits[3] = minutes % 10;
        digits[4] = seconds / 10;
        digits[5] = seconds % 10;
    } else if (mode == 1) { // Timer Mode
        // Timer digits logic
    } else { // Stopwatch Mode
        // Stopwatch digits logic
    }

    // Multiplex 7-segment display
    for (int i = 0; i < 6; i++) {
        setBCD(digits[i]); // Send the BCD value
        COMMON_PORT = (1 << i); // Enable the corresponding digit
        _delay_us(500); // Short delay for smooth display
    }
}
```

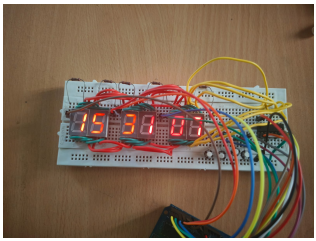
7 Special Logic Implemented

- **State Management Using Conditional Logic**:

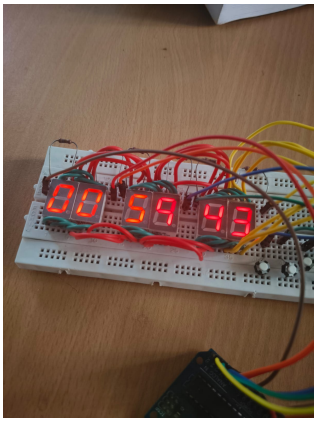
  - The ‘mode‘ variable acts as a state machine, with values 0, 1, and 2 corresponding to Clock, Timer, and Stopwatch modes, respectively.
  - Logical separation of functionality is achieved using ‘if-else‘ conditions within the ISR.
- **Efficient Timing with Timer1 Interrupts**:
  - Timer1 is configured in CTC Mode, where the timer triggers the ISR every second. This eliminates reliance on delay functions or main loop polling.
- **Modular Countdown and Increment Logic**:
  - The Timer Mode implements a cascading decrement mechanism: seconds decrement until zero, at which point minutes or hours are adjusted.
  - This cascading logic mimics the behavior of a **ripple counter** often seen in digital hardware design.
- **Persistence of Vision (Multiplexing)**:
  Uses a sequential activation of each digit on the 7-segment display, controlled through a multiplexing mechanism with BCD encoding. This reduces the number of required microcontroller pins.

8 Conclusion

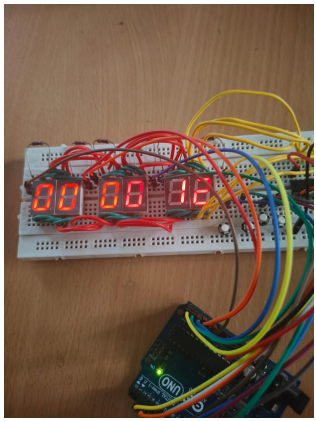
This setup allows efficient control of six 7-segment displays using a single Arduino Uno and a 7447 decoder. However, be mindful of the current limits and use transistors for safer, prolonged operation.
The main logic of this program demonstrates an efficient, interrupt-driven design for a timekeeping system. By leveraging state management and modular timekeeping logic, the system provides robust functionality with minimal resource usage. Key aspects like cascading decrement logic for the timer and multiplexed display control highlight the elegance of combining hardware and software in embedded systems.



(c) Clock



(d) Timer



(e) Stopwatch