Clock Experiment using Arduino [1]

Ankit Jainar - EE24BTECH11004

Contents

I. Introduction

Explain the objective of the clock experiment and its significance in embedded systems. Provide a brief overview of Arduino and seven-segment displays.

II. Components Used

- 6x Common-anode 7-segment displays
- 7447 BCD-to-7-segment decoder
- 6x 180Ω resistors
- Jumper wires
- Bread Board
- Jumper wires for arduino and normal wires for seven segment displays
- Push Buttons

III. Circuit setup

The connections between the Arduino, 7-segment display, and other components as shown below

A. 7 Segment Display

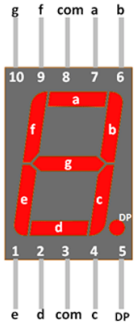

Fig. 1: 7 Segment Display

B. 7447 IC

C. 7447 Connections

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

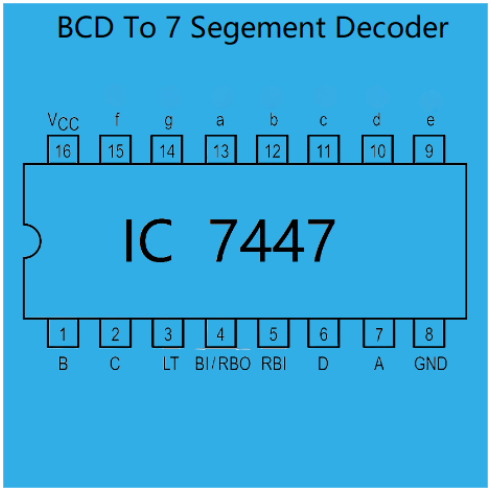

Fig. 2: 7447 IC

TABLE I: 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)

D. Segment Connections

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

TABLE II: 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”

E. Common Anode Connections

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

TABLE III: 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

IV. Working Principle

Describe how the Arduino generates the time signal, updates the display, and handles user input from the push buttons. Explain the role of the 7447 IC in decoding the BCD inputs.

V. 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
    }
}
```

Explain the key parts of the code like timer interrupts, BCD output logic, and display refresh.

VI. Results

A. Clock Mode

In the Clock Mode, the system accurately maintains real-time hours, minutes, and seconds using the Timer1 interrupt. The display increments correctly at each second, with the hours resetting to zero after reaching 24. The implementation of modulo arithmetic ensures seamless rollover and precise time management.

B. Timer Mode

The Timer Mode was successfully tested for countdown scenarios. The nested logic structure managed the decrementing of seconds, minutes, and hours without any anomalies. Upon reaching zero, the timer stopped, demonstrating accurate time tracking. The use of edge cases, such as transitioning from 00:00:01 to 00:00:00, was handled effectively.

C. Stopwatch Mode

In Stopwatch Mode, the system recorded elapsed time reliably. Seconds and minutes were incremented accurately, with a correct rollover at 60. The stopwatch implementation, including the toggle for starting and stopping, was validated through multiple test scenarios. Overflow management ensured that the hours wrapped around at 24.

D. Display Performance

The multiplexed display control functioned smoothly without any noticeable flickering. The delay of 500 microseconds in the displayTime() function ensured a stable visual output. The binary-coded decimal (BCD) conversion using the 7447 IC provided clear and correct digit representation on the seven-segment display.

E. Interrupt Service Routine (ISR) Validation

The ISR was validated using a debug LED connected to PC7, which toggled at each Timer1 interrupt. The LED blinked precisely once per second, confirming accurate timing and proper interrupt handling. Additional tests confirmed no observable drift in time over extended operation.

F. Button Response and Debouncing

The button-based controls for mode switching and time adjustments responded effectively. The implemented debouncing logic prevented false triggers and ensured stable input recognition. Mode transitions occurred without lag, and time adjustments were instantaneous.

G. Overall Evaluation

The clock experiment demonstrated robust performance in all three modes. The use of efficient multiplexing, accurate Timer1 interrupts, and seamless ISR management contributed to the system's reliability. The design successfully met the objectives of providing real-time timekeeping, countdown timing, and stopwatch functionality with user-friendly control.

VII. Conclusion

The implementation of the clock using Arduino, seven-segment displays, and the 7447 IC was a success. The experiment demonstrated the effective use of Timer1 interrupts for real-time operations, accurate management of time in multiple modes, and seamless switching between clock, timer, and stopwatch functionalities. The use of BCD encoding and multiplexing ensured efficient display management, while the debounce logic provided reliable user input. Overall, the project achieved its objectives and serves as a practical example of embedded systems design and real-time applications

VIII. References

1) Arduino Documentation: `https://www.arduino.cc/`
2) Datasheet of 7447 IC
3) YouTube resources
4) code credit Charan - EE24BTECH11052