CLOCK

EE24BTECH11009 - Mokshith Kumar

Contents

1 Introduction

In this guide, we will build a 6-digit clock (HH:MM:SS) using an Arduino, a 74447 BCD to 7-segment decoder, and a multiplexing technique to efficiently display the time. This project helps you learn about timekeeping, display control, and embedded programming.

2 Components Required

Before we start, we need the following components:

- Arduino Uno
- 74447 BCD to 7-Segment Decoder
- 6 x Common-Anode 7-Segment Displays
- Breadboard and Jumper Wires
- Resistors (220Ω)
- 5*V*Power Supply

3 Wiring the Circuit

3.1 Connections of 7447

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 Connections to seven segment display

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 connections to Arduino

The wiring involves connecting: Arduino to the 74447 for BCD control as in the module, then connect 74447 to the 7-segment display segments(to each of them as if thay are in series).Common-anode pins of each display to Arduino pins. Buttons for time adjustment and mode selection(optional).
Proper wiring ensures smooth operation and clear display visibility.

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 Programming the Arduino

4.1 overview

The Arduino controls the clock using: A Timer Interrupt to update time every second. Multiplexing to switch between digits quickly. Button Handling to adjust time and change modes. A simple loop continuously updates the display using BCD encoding and digit control.

4.2 code

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define BCD_PORT PORTD
#define BCD_DDR DDRD
#define BCD_MASK 0b00111100 // PD2 to PD5

#define COMMON_PORT PORTC
#define COMMON_DDR DDRC

#define MODE_BUTTON PB0 // Switch between Clock, Timer, and Stopwatch
#define STOPWATCH_BUTTON PB1 // Start/Stop Stopwatch

volatile int seconds = 0, minutes = 30, hours = 15;
volatile int timer_seconds = 0, timer_minutes = 0, timer_hours = 0;
volatile int stopwatch_seconds = 0, stopwatch_minutes = 0, stopwatch_hours = 0;
```

```
volatile int mode = 0; // 0 = Clock, 1 = Timer, 2 = Stopwatch
volatile int stopwatch_running = 0; // 1 = Running, 0 = Stopped

void setup() {
    // Set BCD display pins (PD2-PD5) as output
    BCD_DDR |= BCD_MASK;
    BCD_PORT &= ~BCD_MASK;

    // Set digit selector pins (PORTC) as output
    COMMON_DDR = 0xFF;
    COMMON_PORT = 0x00;

    // Enable pull-up resistors for buttons
    PORTD |= (1 << PD6) | (1 << PD7);
    PORTB |= (1 << MODE_BUTTON) | (1 << STOPWATCH_BUTTON);

    // Timer1 Setup: CTC Mode, 1-second interval
    TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10);
    OCR1A = 15625; // 1-second interrupt
    TIMSK1 |= (1 << OCIE1A);

    // Debug LED on PC7 (Bit 7 of PORTC) to check if ISR is running
    DDRC |= (1 << 7); // Set PC7 as output
    PORTC &= ~(1 << 7); // Initially turn it off

    sei(); // Enable global interrupts
}


void displayTime();
void setBCD(int value);
void checkButtons();

int main() {
    setup();
    while (1) {
        checkButtons();
        displayTime();
    }
}

// Function to display time on a 6-digit 7-segment display
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
        digits[0] = timer_hours / 10;
        digits[1] = timer_hours % 10;
        digits[2] = timer_minutes / 10;
        digits[3] = timer_minutes % 10;
        digits[4] = timer_seconds / 10;
        digits[5] = timer_seconds % 10;
    } else { // Stopwatch Mode
        digits[0] = stopwatch_hours / 10;
        digits[1] = stopwatch_hours % 10;
        digits[2] = stopwatch_minutes / 10;
        digits[3] = stopwatch_minutes % 10;
        digits[4] = stopwatch_seconds / 10;
        digits[5] = stopwatch_seconds % 10;
    }

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


// Function to set BCD output for 7-segment display
void setBCD(int value) {
    BCD_PORT = (BCD_PORT & ~BCD_MASK) | ((value << 2) & BCD_MASK);
}

// Function to check button inputs and update mode/settings
void checkButtons() {
    if (!(PIND & (1 << PD6))) {
        _delay_ms(50);
        if (!(PIND & (1 << PD6))) {
            if (mode == 0) {
                hours = (hours + 1) % 24;
                seconds = 0;
```

```
            } else if (mode == 1) {
                timer_hours = (timer_hours + 1) % 24;
                seconds = 0;
            }
            while (!(PIND & (1 << PD6))); // Wait for release
        }
    }

    if (!(PIND & (1 << PD7))) {
        _delay_ms(50);
        if (!(PIND & (1 << PD7))) {
            if (mode == 0) {
                minutes = (minutes + 1) % 60;
                seconds = 0;
            } else if (mode == 1) {
                timer_minutes = (timer_minutes + 1) % 60;
                seconds = 0;
            }
            while (!(PIND & (1 << PD7))); // Wait for release
        }
    }

    if (!(PINB & (1 << MODE_BUTTON))) {
        _delay_ms(50);
        if (!(PINB & (1 << MODE_BUTTON))) {
            mode = (mode + 1) % 3; // Cycle through Clock, Timer, and Stopwatch
            while (!(PINB & (1 << MODE_BUTTON))); // Wait for release
        }
    }
}
```

5 Implementation

5.1 How Multiplexing Works

In principle of multiplexing only one display is active at a time. The microcontroller(Arduino) sends the appropriate digit to the common data bus. A control signal activates the desired display. This process repeats quickly (1ms per digit), making all digits appear simultaneously to the human eye. like this we can create an illusion of the displays lighted continuously.

6 Testing and Calibration

After uploading the code, verify that:

- The time increments correctly.
- The buttons adjust the time as expected.

- The display is stable and flicker-free.

Adjust delays if flickering occurs, and check wiring if digits display incorrectly.