DIGITAL CLOCK

EE24BTECH11023 - RASAGNA

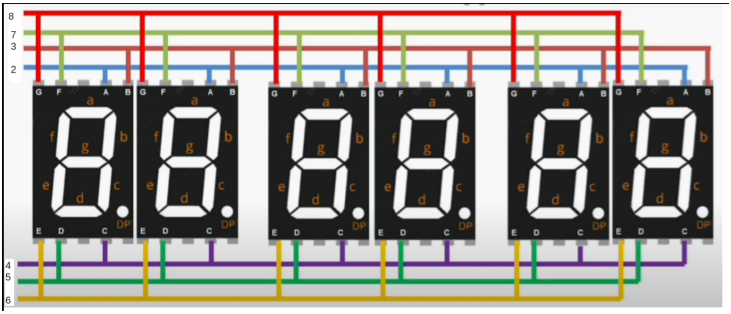
1 Objective

- The main objective of this project is to design and implement a digital clock using six common-anode seven-segment displays and an Arduino.
- The clock accurately displays hours, minutes, and seconds.
- The focus is on direct control of the displays using Arduino's digital I/O pins while implementing precise timekeeping through software-based delay function.
- The clock is further extended and alarm is also included using an active Buzzer.
- This project demonstrates an understanding of seven-segment display interfacing and multiplexing techniques.

2 Components and Equipment

S.No	Component	Quantity
1	Arduino	1
2	Breadboard	1
3	Common-anode Seven segment displays	6
4	USB A to USB B cable	1
5	OTG adapter	1
6	Jumper wires (Male-Male)	70
7	Resistors 220 Ω	6
8	Active Buzzer	1
9	Push Buttons	4

3 Circuit Diagram and Schematic



- The pins of seven segment display, Namely, a,b,c,d,e,f,g, are connected together.These are then connected to 2,3,4,5,6,7,8 pins on the arduino.
- The pin between a and f is COM(Common pin) of first display is connected to pin 9 on arduino.Similarly 2nd,3rd,4th,5th,6th COM pins are connected to 10,11,12,13,A0 pins on the arduino.
- The must be a resistor of 220 Ω between the COM pin and arduino pins to avoid high voltage which may burn out the segments, making them dim permanently or stop working entirely.
- The dot pins are grounded.

4 Working Principle-Software Implementation

4.1 Multiplexing

1) Multiplexing is a technique used to control multiple seven-segment displays using fewer Arduino pins by turning on one display at a time very quickly.
2) This creates an illusion that all displays are ON simultaneously.
3) All A-G segment pins of the displays are connected together and controlled by the same Arduino pins.
4) The Arduino activates one display, sends the digit data, then quickly switches to the next display.
5) The Arduino rapidly cycles through each display thousands of times per second, making it appear that all are ON at the same time.

4.2 Buttons Functionality

1) Initially we connect jumper wires to the analog pins (A1, A2, and A3).
2) A1 controls the hour's units place (h2), A2 controls the minute's units place (m2), and A3 controls the second's units place (s2).
3) We use a method called state change detection to capture when the jumper wire is connected (from HIGH to LOW) to increment the respective value (h2, m2, or s2).
4) When the jumper wire is tapped on A1 (for example), the program checks if the current value of h2 (the hour's units place) is less than 9. If it is, the value increments by one.
5) Similarly, tapping A2 increments the minutes' units place (m2), and A3 increments the seconds' units place (s2).

4.3 Debouncing

1) Mechanical buttons often generate noisy signals, causing bouncing, where the button state might change rapidly due to physical contact. This can cause multiple increments instead of just one.
2) To prevent this, a debounce delay is added, which ensures that only one increment happens for each button press (or jumper wire tap).
3) After detecting a state change, the program waits for a short period (e.g., 300 ms) before checking the button state again.
4) This debounce delay helps ignore any unintended multiple presses from the same action.

4.4 Alarm functioning

1) We use analog pin A4 as the alarm setting button.
2) When 4th Push button is pressed, the clock changes to alarm mode and we can use Hour pin, Minutes pin and seconds pin to set the alarm time.
3) After setting the alarm time we again need to press A4 to fix the alarm time.
4) Whenever the time is reached the buzzer rings for 15 seconds and goes off.

The following code is used to program the Arduino for controlling the digital clock. It handles multiplexing of six seven-segment displays, updates the time, allows us to set alarm and manages display refreshing.

4.5 C Code

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

#define PINS_COUNT 6
#define NO_SEGMENTS 7

//Initialising the clock
uint8_t h1 = 0, h2 = 0, m1 = 0, m2 = 0, s1 = 0, s2 = 0;
uint8_t alarm_h1 = 0, alarm_h2 = 0, alarm_m1 = 0, alarm_m2 = 0, alarm_s1 = 0, alarm_s2
      = 0;
uint8_t alarm_set = 0, alarm_active = 0, setting_alarm = 0;

#define HOUR_PIN PC1
#define MINUTE_PIN PC2
#define SECOND_PIN PC3
#define ALARM_PIN PC4
#define BUZZER_PIN PC5

volatile unsigned long millis_counter = 0;
uint8_t currentDisplay = 0;
unsigned long lastHourPress = 0, lastMinutePress = 0, lastSecondPress = 0,
     lastAlarmPress = 0;
unsigned long alarm_start_time = 0;

void timer0_init(void) {
    TCCR0A = 0;
    TCCR0B = (1 << CS01) | (1 << CS00);
    TIMSK0 = (1 << TOIE0);
    sei();
}

ISR(TIMER0_OVF_vect) {
    millis_counter += 1;
}

unsigned long millis(void) {
    return millis_counter;
}

void clearSegments(void) {
    PORTD &= 0b00000011;
    PORTB &= 0b11111110;
}

//Defining the connections of seven segment display to arduino
void sevenseg(uint8_t a, uint8_t b, uint8_t c, uint8_t d, uint8_t e, uint8_t f,
     uint8_t g) {
```

```
    clearSegments();
    PORTD |= (a << PD2) | (b << PD3) | (c << PD4) | (d << PD5) | (e << PD6) | (f << PD7
        );
    if (g) PORTB |= (1 << PB0); else PORTB &= ~(1 << PB0);
}

//Boolean logic for each segment of seven segment display
void segments(uint8_t A,uint8_t B, uint8_t C, uint8_t D){
       uint8_t a=!(A || C || (!B && !D) || (B && D));
       uint8_t b=!(A || (C && D) || !B ||(!C && !D));
       uint8_t c=!(A || B || !C || D);
       uint8_t d=!(A || ( C && !D) || (!B && !D ) || (!B && C) || (B && !C && D));
       uint8_t e=!((C && !D) || (!B && !D ));
       uint8_t f=!(A || (B && !C) || (B && !D ) || (!C && !D));
       uint8_t g=!((C && !D) || (!C && B) || (C && !B) || A);
       sevenseg(a, b, c, d, e, f, g);
}

//BCD for digits 0 to 9
void displayDigit(uint8_t digit) {
    static const uint8_t segment_map[10][4] = {
        {0, 0, 0, 0},
        {0, 0, 0, 1},
        {0, 0, 1, 0},
        {0, 0, 1, 1},
        {0, 1, 0, 0},
        {0, 1, 0, 1},
        {0, 1, 1, 0},
        {0, 1, 1, 1},
        {1, 0, 0, 0},
        {1, 0, 0, 1}
    };
     segments(segment_map[digit][0], segment_map[digit][1], segment_map[digit][2],
         segment_map[digit][3]);
}

//Function to display time
void displayClock(void) {
    PORTB &= 0b11000001;
    PORTC &= ~(1 << PC0);

    if (currentDisplay == 5) {
        PORTC |= (1 << PC0);
    } else {
        PORTB |= (1 << (currentDisplay + 1));
    }

    uint8_t display_val = (setting_alarm)
                          ? (currentDisplay == 0 ? alarm_h1 : currentDisplay == 1 ?
                              alarm_h2
    : currentDisplay == 2 ? alarm_m1
    : currentDisplay == 3 ? alarm_m2
    : currentDisplay == 4 ? alarm_s1
    : alarm_s2)
    : (currentDisplay == 0 ? h1 : currentDisplay == 1 ? h2
    : currentDisplay == 2 ? m1
    : currentDisplay == 3 ? m2
    : currentDisplay == 4 ? s1
    : s2);
```

```
    displayDigit(display_val);
    currentDisplay = (currentDisplay + 1) % PINS_COUNT;
}

//Function to handle Hour button, Minute button ans seconds button
void handleButtons(void) {
    unsigned long currentMillis = millis();

    // Toggle alarm setting mode
    if (!(PINC & (1 << ALARM_PIN))) {
        if (currentMillis - lastAlarmPress > 300) {
            lastAlarmPress = currentMillis;
            setting_alarm = !setting_alarm;
            if (!setting_alarm) {
                alarm_set = 1;
            }
        }
    }

    // Adjust hours
    if (!(PINC & (1 << HOUR_PIN))) {
        if (currentMillis - lastHourPress > 300) {
            lastHourPress = currentMillis;
            uint8_t *h1_ptr = setting_alarm ? &alarm_h1 : &h1;
            uint8_t *h2_ptr = setting_alarm ? &alarm_h2 : &h2;

            (*h2_ptr)++;
            if (*h2_ptr >= 10) {
                *h2_ptr = 0;
                (*h1_ptr)++;
            }
            if (*h1_ptr >= 2 && *h2_ptr >= 4) {
                *h1_ptr = 0;
                *h2_ptr = 0;
            }
        }
    }

    // Adjust minutes
    if (!(PINC & (1 << MINUTE_PIN))) {
        if (currentMillis - lastMinutePress > 300) {
            lastMinutePress = currentMillis;
            uint8_t *m1_ptr = setting_alarm ? &alarm_m1 : &m1;
            uint8_t *m2_ptr = setting_alarm ? &alarm_m2 : &m2;

            (*m2_ptr)++;
            if (*m2_ptr >= 10) {
                *m2_ptr = 0;
                (*m1_ptr)++;
            }
            if (*m1_ptr >= 6) {
                *m1_ptr = 0;
            }
        }
    }

    // Adjust seconds
    if (!(PINC & (1 << SECOND_PIN))) {
```

```
        if (currentMillis - lastSecondPress > 300) {
            lastSecondPress = currentMillis;
            uint8_t *s1_ptr = setting_alarm ? &alarm_s1 : &s1;
            uint8_t *s2_ptr = setting_alarm ? &alarm_s2 : &s2;

            (*s2_ptr)++;
            if (*s2_ptr >= 10) {
                *s2_ptr = 0;
                (*s1_ptr)++;
            }
            if (*s1_ptr >= 6) {
                *s1_ptr = 0;
            }
        }
    }
}

//Function to check the alarm and activate the buzzer for 15 seconds
void checkAlarm(void) {
    static unsigned long last_toggle_time = 0;
    static uint8_t buzzer_state = 0;

    if (alarm_set && h1 == alarm_h1 && h2 == alarm_h2 && m1 == alarm_m1 && m2 ==
         alarm_m2 && s1 == alarm_s1 && s2 == alarm_s2) {
        alarm_active = 1;
        alarm_start_time = millis();
    }

    if (alarm_active) {
        if (millis() - alarm_start_time < 15000) {
            if (millis() - last_toggle_time > 500) {
                last_toggle_time = millis();
                buzzer_state = !buzzer_state;
                if (buzzer_state) {
                    PORTC |= (1 << BUZZER_PIN);
                } else {
                    PORTC &= ~(1 << BUZZER_PIN);
                }
            }
        } else {
            alarm_active = 0;
            PORTC &= ~(1 << BUZZER_PIN);
        }
    }
}

//Function to update seconds minutes and hours in the clock
void updateClock(void) {
    static unsigned long lastUpdate = 0;
    if (millis() - lastUpdate >= 1000) {
        lastUpdate = millis();
        s2++;
        if (s2 >= 10) { s2 = 0; s1++; }
        if (s1 >= 6) { s1 = 0; m2++; }
        if (m2 >= 10) { m2 = 0; m1++; }
        if (m1 >= 6) { m1 = 0; h2++; }
        if (h2 >= 10) { h2 = 0; h1++; }
        if (h1 == 2 && h2 == 4) { h1 = 0; h2 = 0; }
        checkAlarm();
```

```
    }
}

int main(void) {
    DDRD |= 0b11111100;
    DDRB |= 0b00111111;
    DDRC |= (1 << PC0) | (1 << BUZZER_PIN);
    PORTC |= (1 << HOUR_PIN) | (1 << MINUTE_PIN) | (1 << SECOND_PIN) | (1 << ALARM_PIN)
         ;

    timer0_init();

    while (1) {
        updateClock();
        handleButtons();
        displayClock();
        _delay_ms(2);
    }
}
```

5 Advantages of AVR-GCC

- Direct access to AVR registers which results in smaller and faster machine code.
- No C++ overhead (such as classes, virtual functions, and dynamic memory allocation).
- We can directly manipulate registers (e.g., PORTB, DDRC) for faster execution.
- No need for functions like digitalWrite(), which are slower than direct register access.
- No unnecessary code from high-level abstractions or unused libraries.
- Avoids unexpected behavior caused by library overhead.
- AVR-GCC is faster, more efficient, and gives complete hardware control.

6 Precautions

6.1 Hardware

1) Always connect 220 Ω resistors in series with the seven-segment display segments to prevent excessive current draw and potential damage to the Arduino.
2) Double-check wiring to ensure no accidental short circuits, which could damage the microcontroller or display.
3) Before making any changes to the circuit, disconnect the Arduino from the power source to prevent accidental damage.
4) Also connect a 1 Mega ohm Resistor in parallel to buzzer because buzzer can retain charge due to their internal capacitance.
5) This can cause the buzzer to not turn off immediately or produce unwanted noise after switching off.
6) A 1 MΩ resistor in parallel provides a path for residual charge to dissipate, ensuring the buzzer turns off completely after 15 seconds.

6.2 Software

1) Ensure that the delay in the multiplexing loop is optimized to avoid flickering or unreadable digits. A 1.5-5ms delay per digit works well.

2) Before uploading the code, double-check that the correct pins are assigned to the display segments and common anodes.