DIGITAL CLOCK

EE24BTECH11023 - RASAGNA

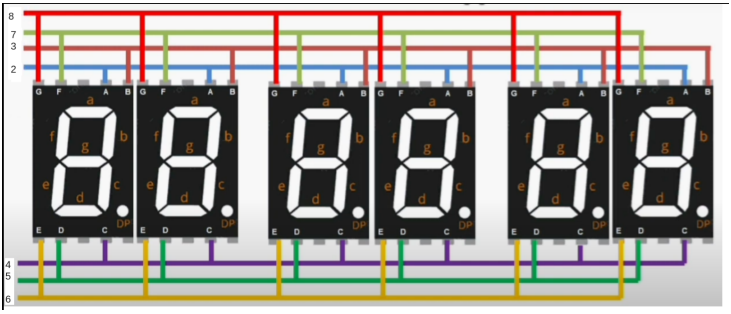
1 Objective

The 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. 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

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.

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, and manages display refreshing.

4.4 CPP Code

```
#include "Arduino.h"

#define PINS_COUNT 6 // Number of displays
#define NO_SEGMENTS 7 // Number of segments per display

int pins[PINS_COUNT] = {9, 10, 11, 12, 13, A0};
int segPins[NO_SEGMENTS] = {2, 3, 4, 5, 6, 7, 8};

unsigned long previousMillis = 0;
const int interval = 2; // 2 ms per digit
int currentDisplay = 0; // Keeps track of which display is active

int h1 = 0, h2 = 0, m1 = 0, m2 = 0, s1 = 0, s2 = -1;

#define HOUR_PIN A1 // Hour increment
#define MINUTE_PIN A2 // Minute increment
#define SECOND_PIN A3 // Seconds increment

bool lastHourState = HIGH;
bool lastMinuteState = HIGH;
bool lastSecondState = HIGH;

void setup() {
  for (int i = 0; i < NO_SEGMENTS; i++) {
    pinMode(segPins[i], OUTPUT);
  }
  for (int i = 0; i < PINS_COUNT; i++) {
    pinMode(pins[i], OUTPUT);
  }
  pinMode(HOUR_PIN, INPUT_PULLUP); // Hour increment
  pinMode(MINUTE_PIN, INPUT_PULLUP); // Minute increment
  pinMode(SECOND_PIN, INPUT_PULLUP); //Second increment
}

void sevenseg(int a, int b, int c, int d, int e, int f, int g) {
  digitalWrite(2, a);
  digitalWrite(3, b);
```

```
  digitalWrite(4, c);
  digitalWrite(5, d);
  digitalWrite(6, e);
  digitalWrite(7, f);
  digitalWrite(8, g);
}

void displayDigit(int digit) {
  switch (digit) {
    case 0: sevenseg(0, 0, 0, 0, 0, 0, 1); break;
    case 1: sevenseg(1, 0, 0, 1, 1, 1, 1); break;
    case 2: sevenseg(0, 0, 1, 0, 0, 1, 0); break;
    case 3: sevenseg(0, 0, 0, 0, 1, 1, 0); break;
    case 4: sevenseg(1, 0, 0, 1, 1, 0, 0); break;
    case 5: sevenseg(0, 1, 0, 0, 1, 0, 0); break;
    case 6: sevenseg(0, 1, 0, 0, 0, 0, 0); break;
    case 7: sevenseg(0, 0, 0, 1, 1, 1, 1); break;
    case 8: sevenseg(0, 0, 0, 0, 0, 0, 0); break;
    case 9: sevenseg(0, 0, 0, 0, 1, 0, 0); break;
  }
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis; // Reset timer

    for (int i = 0; i < PINS_COUNT; i++) {
      digitalWrite(pins[i], LOW);
    }

    digitalWrite(pins[currentDisplay], HIGH);

    // Display the digit on the active display
    switch (currentDisplay) {
      case 0: displayDigit(h1); break;
      case 1: displayDigit(h2); break;
      case 2: displayDigit(m1); break;
      case 3: displayDigit(m2); break;
      case 4: displayDigit(s1); break;
      case 5: displayDigit(s2); break;
    }

    // Move to the next display
```

```
    currentDisplay++;
    if (currentDisplay >= PINS_COUNT) {
      currentDisplay = 0;
    }
  }

  // Update time every second
  static unsigned long lastSecond = 0;
  if (currentMillis - lastSecond >= 1000) {
    lastSecond = currentMillis; // Reset second timer

    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) {
      h1 = 0;
    }
  }

  // Manual Time Adjustment Using Jumper Wires
  // Hour increment
  bool currentHourState = digitalRead(HOUR_PIN);
  if (currentHourState == LOW && lastHourState == HIGH) {
    h2++; // Increment units place of hours
    if (h2 >= 10) {
      h2 = 0;
      h1++; // Increment tens place of hours
```

```
    }
    if (h1 >= 2 && h2 >= 4) { // Hour reset after 23
      h1 = 0;
      h2 = 0;
    }
    delay(300); // Debounce delay
  }
  lastHourState = currentHourState;

  // Minute increment
  bool currentMinuteState = digitalRead(MINUTE_PIN);
  if (currentMinuteState == LOW && lastMinuteState == HIGH) {
    m2++; // Increment units place of minutes
    if (m2 >= 10) {
      m2 = 0;
      m1++; // Increment tens place of minutes
    }
    if (m1 >= 6) { // Reset after 59 minutes
      m1 = 0;
    }
    delay(300); // Debounce delay
  }
  lastMinuteState = currentMinuteState;

  // Second increment
  bool currentSecondState = digitalRead(SECOND_PIN);
  if (currentSecondState == LOW && lastSecondState == HIGH) {
    s2++; // Increment units place of seconds
    if (s2 >= 10) {
      s2 = 0;
      s1++; // Increment tens place of seconds
    }
    if (s1 >= 6) { // Reset after 59 seconds
      s1 = 0;
    }
    delay(300); // Debounce delay
  }
  lastSecondState = currentSecondState;
}
```

5 Precautions

5.1 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.

5.2 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.