Digital Clock

EE24BTECH11049
Patnam Shariq Faraz Muhammed

Abstract

This report focuses on implementation of a digital clock using an AVR microcontroller, displaying time on both a 7-segment display and an LCD. It allows users to manually set the time and switch between multiple time zones using buttons. The clock maintains accurate time using internal timers and does not require an external RTC module. Time zone adjustments are made using pre-defined offsets. The system is efficient, user-friendly, and suitable for embedded applications where real-time clock management is necessary.

CONTENTS

1 Introduction 3

2 Hardware Components 3

3 Hardware Setup 3
3.1 Connections 5

4 How my code works? 6
4.1 Multiplexing (7-Segment Display) 6
4.1.1 Multiplexing Implementation in ISR 6
4.2 Button Functionality 6
4.2.1 Button Press Detection 6
4.3 Debouncing 6
4.4 Time Zone Functionality 7
4.4.1 Time Zone Selection and Adjustment 7
4.5 Timekeeping Using Timers 7

5 Advatanges of AVR-GCC over other 8

6 Code 8

7 Precautions 12
7.1 Hardware 12
7.2 Software 12

1 Introduction

This project is an AVR microcontroller-based digital clock that displays time using both a 7-segment display and an LCD. It allows users to set the time, change time zones, and interact using buttons. The clock uses internal timers for timekeeping and multiplexing the 7-segment display. The implementation emphasizes efficiency by using bitwise operations instead of arithmetic operations like ++ and – in some areas in which I was able to understand properly.

2 Hardware Components

The following are components required for the project

- Six 7-segment displays
- Six resistors (220Ω)
- Arduino UNO
- Breadboard
- Jumper wires
- Cell phone (to power the Arduino)

3 Hardware Setup

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

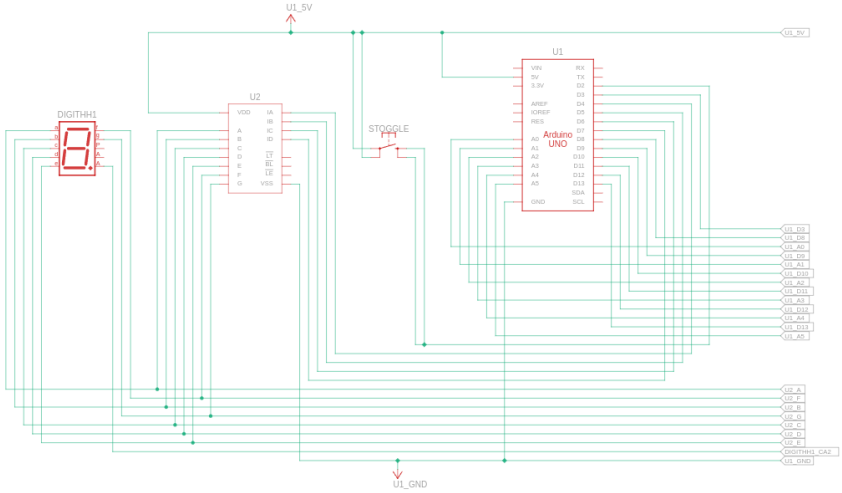

Fig. 0.1: connections

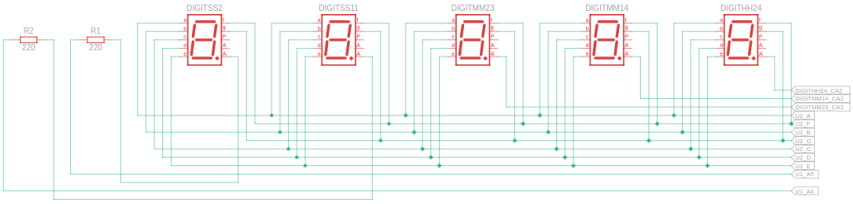

Fig. 0.2: connections

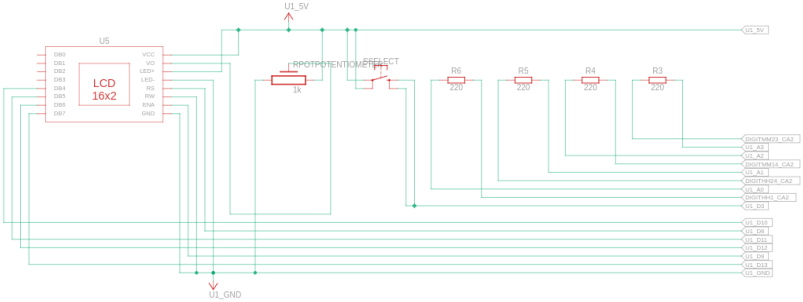

Fig. 0.3: connections

3.1 Connections

1) The table below shows the connections between the 7447 decoder, seven-segments, LCD and the Arduino Uno:

2) Each display's common anode is connected to an Arduino analog pin through a 220Ω resistor to limit current flow.
3)

Component	Arduino Pin	Description
LCD Display		
RS	PB0 (Digital 8)	Register Select line
E	PB1 (Digital 9)	Enable line
D4	PB2 (Digital 10)	Data line 4
D5	PB3 (Digital 11)	Data line 5
D6	PB4 (Digital 12)	Data line 6
D7	PB5 (Digital 13)	Data line 7
7-Segment Display		
Segment A	PD4 (Digital 4)	Segment A line
Segment B	PD5 (Digital 5)	Segment B line
Segment C	PD6 (Digital 6)	Segment C line
Segment D	PD7 (Digital 7)	Segment D line
Common HH_1	PC0 (Analog 0)	Hours tens digit common
Common HH_2	PC1 (Analog 1)	Hours ones digit common
Common MM_1	PC2 (Analog 2)	Minutes tens digit common
Common MM_2	PC3 (Analog 3)	Minutes ones digit common
Common SS_1	PC4 (Analog 4)	Seconds tens digit common
Common SS_2	PC5 (Analog 5)	Seconds ones digit common
Buttons		
Toggle Button	PD2 (Digital 2)	For cycling through options
Select Button	PD3 (Digital 3)	For selecting options

TABLE 3: Arduino to Component Connections

4 How my code works?

4.1 Multiplexing (7-Segment Display)

The code uses multiplexing to display time on a 7-segment display, reducing the number of required I/O pins. This is done by:

- Assigning `PORTD` (higher 4 bits) for segment values.
- Using `PORTC` to control the common cathode/anode lines.
- Using **Timer0 Interrupt Service Routine (ISR)** to rapidly cycle through digits.

4.1.1 Multiplexing Implementation in ISR:

```
ISR(TIMER0_COMPA_vect) {
    static uint8_t digitIndex = 0;
    uint8_t digits[6] = {hours / 10, hours % 10, minutes / 10, minutes % 10
    uint8_t pins[6] = {HH_1, HH_2, MM_1, MM_2, SS_1, SS_2};

    displayDigit(pins[digitIndex], digits[digitIndex]);
    digitIndex = (digitIndex + 1) % 6;
}
```

The function `displayDigit()` updates the display for one digit at a time, creating the illusion of a full display through rapid switching.

4.2 Button Functionality

The system has two buttons:

- **Toggle Button**: Cycles through options (e.g., selecting a time zone or incrementing a digit).
- **Select Button**: Confirms a selection.

4.2.1 Button Press Detection:

```
bool isButtonPressed(uint8_t pin) {
    if (!(PIND & (1 << pin))) {  // Active LOW
        _delay_ms(200);  // Debounce delay
        if (!(PIND & (1 << pin))) return true;
    }
    return false;
}
```

The button press is detected using `PIND` register, and a debounce delay (explained in Section 4.3) ensures reliable operation.

4.3 Debouncing

To prevent false triggers from mechanical button bouncing, the code implements software debouncing:

1) When a button press is detected, the code waits for 200 ms.
2) It checks if the button is still pressed.
3) If the button remains pressed, the function returns `true`.

4.4 Time Zone Functionality

The system supports eight time zones, defined as:

4.4.1 Time Zone Selection and Adjustment: When the user presses the toggle button, the time zone cycles through the available options. The adjustment follows:

```
if (isButtonPressed(TOGGLE_BUTTON)) {
    selectedZone = (~selectedZone) & 7; // Cycle through zones
    int adjHours = hours - timezoneOffsets[currentZone] + timezoneOffsets[s

    if (adjHours < 0) adjHours += 24;  // Handle negative wrap-around
    if (adjHours >= 24) adjHours -= 24;
    hours = adjHours;
    currentZone = selectedZone;
}
```

The time zone change is applied by adjusting `hours` based on the old and new offsets, with wrap-around logic for negative and overflow cases.

4.5 Timekeeping Using Timers

The system uses hardware timers to keep track of time. Timer1 is configured to generate an interrupt every second, which increments the seconds counter. When the seconds reach 60, the minutes counter is incremented, and similarly, when minutes reach 60, the hours counter is updated. The hours wrap around after 24 to maintain a 24-hour format.

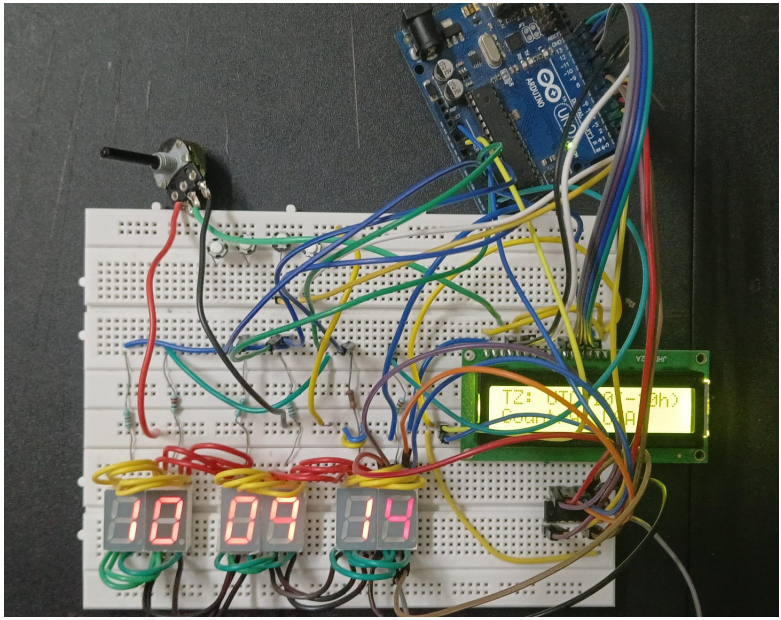

Fig. 3.1: Clock

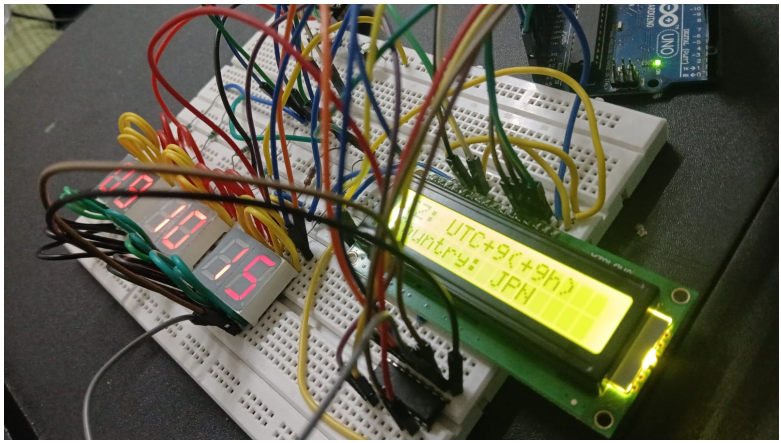

Fig. 3.2: Clock

5 Advatanges of AVR-GCC over other

- Optimized Code for Real-Time Execution
- Offers optimized ISR vector mapping for fast execution with low latency.
- Supports direct manipulation of AVR hardware registers, which results in faster response times than higher-level abstraction layers.
- Direct register-level programming (e.g., TCCR0A, OCR0A, TIMSK1) is completely compatible with different AVR devices.
- Precise Handling of Button Debouncing
- Highly optimized AVR-Libc functions, which minimize RAM and Flash usage.
- Efficient register usage, ensuring that conversions and LCD updates do not introduce significant computational delays.
- Robust Bitwise Operations for Fast Multiplexing
  - Streamlined inline assembly translation with lower instruction cycles for enhanced performance.
  - Low RAM overhead due to bitwise operations' lack of memory allocations.

6 Code

Code for the Digital CLock

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

// TYPEDEFS
typedef uint8_t byte; // changed the name


// ------------------
//LCD DRIVER ROUTINES
//
// Routines:
// LCD_Init initializes the LCD controller
// LCD_Cmd sends LCD controller command
// LCD_Char sends single ascii character to display
// LCD_Clear clears the LCD display & homes cursor
```

```
// LCD_Integer displays an integer value
// LCD_Message displays a string
// PortB is used for data communications with the HD44780-controlled LCD.
// The following defines specify which port pins connect to the controller:
#define ClearBit(x,y) x &= ~_BV(y) // equivalent to cbi(x,y)
#define SetBit(x,y) x |= _BV(y) // equivalent to sbi(x,y)
#define LCD_RS 0 // pin for LCD R/S (eg PB0)
#define LCD_E 1 // pin for LCD enable
#define DAT4 2 // pin for d4
#define DAT5 3 // pin for d5
#define DAT6 4 // pin for d6
#define DAT7 5 // pin for d7
//// The following defines are controller commands
#define CLEARDISPLAY 0x01

#define A_PIN 4
#define B_PIN 5
#define C_PIN 6
#define D_PIN 7

// 7-Segment Common Pins
#define HH_1 PC0
#define HH_2 PC1
#define MM_1 PC2
#define MM_2 PC3
#define SS_1 PC4
#define SS_2 PC5

// Button Pins
#define TOGGLE_BUTTON 2
#define SELECT_BUTTON 3



void PulseEnableLine ()
{
 SetBit(PORTB,LCD_E); // take LCD enable line high
 _delay_us(40); // wait 40 microseconds
 ClearBit(PORTB,LCD_E); // take LCD enable line low
}
void SendNibble(byte data)
{
 PORTB &= 0xC3; // 1100.0011 = clear 4 data lines
 if (data & _BV(4)) SetBit(PORTB,DAT4);
 if (data & _BV(5)) SetBit(PORTB,DAT5);
 if (data & _BV(6)) SetBit(PORTB,DAT6);
 if (data & _BV(7)) SetBit(PORTB,DAT7);
 PulseEnableLine(); // clock 4 bits into controller
}
void SendByte (byte data)
{
 SendNibble(data); // send upper 4 bits
 SendNibble(data<<4); // send lower 4 bits
 ClearBit(PORTB,5); // turn off boarduino LED
}
void LCD_Cmd (byte cmd)
{
 ClearBit(PORTB,LCD_RS); // R/S line 0 = command data
 SendByte(cmd); // send it
}
void LCD_Char (byte ch)
{
 SetBit(PORTB,LCD_RS); // R/S line 1 = character data
 SendByte(ch); // send it
}
void LCD_Init()
{
 LCD_Cmd(0x33); // initialize controller
 LCD_Cmd(0x32); // set to 4-bit input mode
 LCD_Cmd(0x28); // 2 line, 5x7 matrix
 LCD_Cmd(0x0C); // turn cursor off (0x0E to enable)
 LCD_Cmd(0x06); // cursor direction = right
 LCD_Cmd(0x01); // start with clear display
 _delay_ms(3); // wait for LCD to initialize
}
void LCD_Clear() // clear the LCD display
{
 LCD_Cmd(CLEARDISPLAY);
 _delay_ms(3); // wait for LCD to process command
}

void LCD_Message(const char *text) // display string on LCD
{
 while (*text) // do until /0 character
 LCD_Char(*text++); // send char & update char pointer
}

void LCD_Integer(int data)
// displays the integer value of DATA at current LCD cursor position
```

```
{
 char st[8] = ""; // save enough space for result
 itoa(data,st,10); //
 LCD_Message(st); // display in on LCD
}

volatile int hours = 0, minutes = 0, seconds = 0;



// Timezone variables
int currentZone = 0;
const int timezoneOffsets[8] = {0, -5, -10, 3, 4, 9, -7, 1};
const char* timezoneNames[8] = {"IST", "UTC-5", "UTC-10", "UTC+3", "UTC+4", "UTC+9", "UTC-7", "UTC+1"};
const char* countryAbbreviations[8] = {"IND", "USA", "USA", "RUS", "UAE", "JPN", "USA", "EUR"};


// Display
void displayDigit(uint8_t commonPin, uint8_t digit) {
    PORTD = (PORTD & 0x0F) | ((digit & 0x0F) << 4);
    PORTC |= (1 << commonPin);
    _delay_us(500);
    PORTC &= ~(1 << commonPin);
}

ISR(TIMER0_COMPA_vect) {
    static uint8_t digitIndex = 0;
    uint8_t digits[6] = {hours / 10, hours % 10, minutes / 10, minutes % 10, seconds / 10, seconds % 10};
    uint8_t pins[6] = {HH_1, HH_2, MM_1, MM_2, SS_1, SS_2};

    displayDigit(pins[digitIndex], digits[digitIndex]);
    digitIndex = (digitIndex + 1) % 6;
}

void Timer0_Init() {
    TCCR0A |= (1 << WGM01);
    TCCR0B |= (1 << CS01) | (1 << CS00);
    OCR0A = 250;
    TIMSK0 |= (1 << OCIE0A);
}

// Timekeeping using Timer1
ISR(TIMER1_COMPA_vect) {
    seconds = ~seconds;
    if (seconds >= 60) {
        seconds = 0;
        minutes = ~minutes;
    }
    if (minutes >= 60) {
        minutes = 0;
        hours = ~hours;
    }
    if (hours >= 24) {
        hours = 0;
    }
}



void Timer1_Init() {
    TCCR1B |= (1 << WGM12) | (1 << CS12);
    OCR1A = 62500;
    TIMSK1 |= (1 << OCIE1A);
    sei();
}

// Button Handling
bool isButtonPressed(uint8_t pin) {
    if (!(PIND & (1 << pin))) { // Button is active LOW
        _delay_ms(200); // Debounce delay
        if (!(PIND & (1 << pin))) return true;
    }
    return false;
}


// Timezone Handling
void handleTimeZones() {
    uint8_t selectedZone = 0;
    LCD_Clear();

    while (1) {
        LCD_Clear();
        LCD_Message("TZ: ");
        LCD_Message(timezoneNames[selectedZone]);
        int8_t offset = timezoneOffsets[selectedZone];
        if (offset >= 0) LCD_Message("(+");
        else LCD_Char('(');
        LCD_Integer(offset);
```

```
        LCD_Message("h)");

        LCD_Cmd(0xC0);
        LCD_Message("Country:␣");
        LCD_Message(countryAbbreviations[selectedZone]);


        if (isButtonPressed(TOGGLE_BUTTON)) {
            selectedZone = (~selectedZone) & 7;
            LCD_Cmd(0xC0);
            LCD_Message("Changed!");
            int adjHours = hours - timezoneOffsets[currentZone] + timezoneOffsets[selectedZone];

        if (adjHours < 0) adjHours += 24;
        if (adjHours >= 24) adjHours -= 24;
            hours = adjHours;
            currentZone = selectedZone;
        }
            _delay_ms(500);
        }

        if (isButtonPressed(SELECT_BUTTON)) {

        _delay_ms(200);
    }
}


void enterISTTime() {
    LCD_Clear();
    LCD_Message("Enter␣Time:");
    _delay_ms(1000);

    for (uint8_t i = 0; i < 6; i++) {
        uint8_t value = 0;
        while (1) {
            LCD_Cmd(0xC0);
            LCD_Message((i < 2) ? "HH:␣" : (i < 4) ? "MM:␣" : "SS:␣");
            LCD_Integer(value);

            if (isButtonPressed(TOGGLE_BUTTON)) {
                value = (value + 1) % 10;
            }

            if (isButtonPressed(SELECT_BUTTON)) {
                break;
            }
        }

        switch (i) {
            case 0: hours += value * 10; break;
            case 1: hours += value; break;
            case 2: minutes += value * 10; break;
            case 3: minutes += value; break;
            case 4: seconds += value * 10; break;
            case 5: seconds += value; break;
        }
    }
}

int main() {
    DDRB = 0xFF; // Port B output for LCD
    DDRD = 0xF0; // Upper bits for 7-segment, lower for input
    PORTD |= (1 << TOGGLE_BUTTON) | (1 << SELECT_BUTTON); // Enable internal pull-ups
    DDRC = 0xFF; // Port C output for 7-segment selection

    LCD_Init();
    enterISTTime(); // User enters time

    Timer0_Init(); // 7-segment display multiplexing
    Timer1_Init(); // Timekeeping

    while (1) {
        // If SELECT_BUTTON is pressed, enter timezone selection mode
        if (isButtonPressed(SELECT_BUTTON)) {
            handleTimeZones();
        }
    }
}
```

7 Precautions

7.1 Hardware

1) Always use 220Ω resistors in series with the seven-segment display segments to avoid drawing too much current and possibly damaging the Arduino.
2) Double-check wiring to avoid short circuits by mistake, which may damage the microcontroller or display.
3) First of all, cut off the power supply to the Arduino before changing anything in the circuit to avoid any accidental destruction.

7.2 Software

1) Set the delay of the multiplexing loop such that it will not flicker or display illegible digits. A 1.5-5ms delay per digit is good.
2) Double-check before uploading code that the appropriate pins are connected to the segments and common anodes of the display.