Digital Clock

Hardware Project

EE1003 : Scientific Programming

GVV Sharma

Harshil Rathan Y EE24BTECH11064

Contents

1 Introduction

This project implements a real time 24-hour clock using an Arduino, 7-segment displays and a 7447 BCD decoder using boolean logic. The clock displays hours, minutes and seconds using the time keeping logic iin avr-gcc

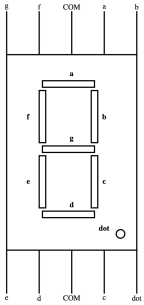
2 Components Required

1. Arduino UNO - 1
2. 7 Segment Display - 6
3. 7447 IC - 1
4. Breadboard - 2
5. Jumper cables
6. Push Buttons

3 Circuit Design

3.1 Pin Diagrams

The pin-out diagram of a 7 segement display is given below, in this project we used 6 displays for HH : MM : SS format



The pin-out diagram of a 7447 Decoder is below

3.2 Multiplexing

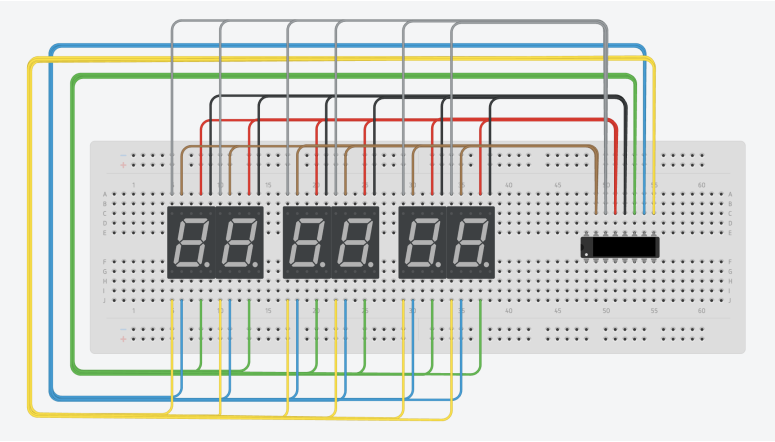
Why is Multiplexing used

- Multiplexing is a method of controlling multiple displays by activating only one display at a time while rapidly switching between them. This creates an illusion of all displays being lit simultaneously due to persistence of vision in human eyes.
- Here we used it to control 6 displays with just one 7447 BCD decoder since the Arduino has a limited number of pins.

Connections

- Connect all corresponding segment pins (a-g) of the 7-segment displays to their respective output pins (13, 12, 11, 10, 9, 15, 14) on the 7447 IC

Make connections accordingly



Follow thesd further connections

Connections	7447 Pins	Arduino Pin
BCD A	Pin 7	D2 (PD2)
BCD B	Pin 1	D3 (PD3)
BCD C	Pin 2	D4 (PD4)
BCD D	Pin 6	D5 (PD5)
H1	220Ω R	D6 (PD6)
H2	220Ω R	D7 (PD7)
M1	220Ω R	D8 (PB0)
M2	220Ω R	D9 (PB1)
S1	220Ω R	D10 (PB2)
S2	220Ω R	D11 (PB3)
VCC	Pin 16	5V
GND	Pin 8	GND

Connect one end of a 220Ω resistor to the displays common and its other end to the respective arduino pins

4 Working

Power up the arduino and upload the following code

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h> // Needed for _delay_ms()

// BCD Pins (A, B, C, D for 7447)
#define A PD2
#define B PD3
#define C PD4
#define D PD5

// Display Control Pins (H1 to S2 mapped to Digital 6-11)
#define H1 PD6 // Hours Tens (Digital Pin 6)
#define H2 PD7 // Hours Units (Digital Pin 7)
#define M1 PB0 // Minutes Tens (Digital Pin 8)
#define M2 PB1 // Minutes Units (Digital Pin 9)
#define S1 PB2 // Seconds Tens (Digital Pin 10)
#define S2 PB3 // Seconds Units (Digital Pin 11)

// Clock Variables (stored in BCD)
volatile uint8_t hours = 0b00010010; // 12 in BCD
volatile uint8_t minutes = 0b00000000; // 00 in BCD
volatile uint8_t seconds = 0b00000000; // 00 in BCD

void displayDigit(uint8_t digit) {
    PORTD = (PORTD & 0b11000011) | (digit << 0b00000010); // Set BCD bits on PD2-PD5
}

void displayTime() {
    uint8_t h1 = (hours >> 0b00000100) & 0b00001111; // Tens place of hours
    uint8_t h2 = hours & 0b00001111; // Units place of hours
    uint8_t m1 = (minutes >> 0b00000100) & 0b00001111;
    uint8_t m2 = minutes & 0b00001111;
    uint8_t s1 = (seconds >> 0b00000100) & 0b00001111;
    uint8_t s2 = seconds & 0b00001111;

    // Multiplexed Display Updates
    PORTD |= (0b00000001 << H1); displayDigit(h1); _delay_ms(0b00000101); PORTD &= ~(0
        b00000001 << H1);
    PORTD |= (0b00000001 << H2); displayDigit(h2); _delay_ms(0b00000101); PORTD &= ~(0
        b00000001 << H2);
    PORTB |= (0b00000001 << M1); displayDigit(m1); _delay_ms(0b00000101); PORTB &= ~(0
        b00000001 << M1);
    PORTB |= (0b00000001 << M2); displayDigit(m2); _delay_ms(0b00000101); PORTB &= ~(0
        b00000001 << M2);
    PORTB |= (0b00000001 << S1); displayDigit(s1); _delay_ms(0b00000101); PORTB &= ~(0
        b00000001 << S1);
    PORTB |= (0b00000001 << S2); displayDigit(s2); _delay_ms(0b00000101); PORTB &= ~(0
        b00000001 << S2);
}

// Timer Interrupt (1 Second)
ISR(TIMER1_COMPA_vect) {
    seconds += 0b00000001; // Increment seconds

    // Handle BCD carry
```

```
    if ((seconds & 0b00001111) > 0b1001) {
        seconds = (seconds & 0b11110000) + 0b00010000; // Carry to tens
    }
    if (seconds >= 0b01100000) { // If seconds = 60 (BCD)
        seconds = 0b00000000;
        minutes += 0b00000001;
    }
    if ((minutes & 0b00001111) > 0b1001) {
        minutes = (minutes & 0b11110000) + 0b00010000; // Carry to tens
    }
    if (minutes >= 0b01100000) { // If minutes = 60 (BCD)
        minutes = 0b00000000;
        hours += 0b00000001;
    }
    if ((hours & 0b00001111) > 0b1001) {
        hours = (hours & 0b11110000) + 0b00010000; // Carry to tens
    }
    if (hours >= 0b00100000) { // If hours = 24 (BCD)
        hours = 0b00000000;
    }
}

// Timer1 Setup
void timer1_init() {
    TCCR1B |= (0b00000001 << WGM12) | (0b00000001 << CS12) | (0b00000001 << CS10); //
        CTC mode, Prescaler 1024
    OCR1A = 0b0011110011111000; // Compare match value for 1 second (16MHz / 1024 / 1Hz
         - 1)
    TIMSK1 |= (0b00000001 << OCIE1A); // Enable Timer1 compare interrupt
    sei(); // Enable global interrupts
}

int main(void) {
    // Configure BCD output pins
    DDRD |= (0b00000001 << A) | (0b00000001 << B) | (0b00000001 << C) | (0b00000001 <<
        D);

    // Configure Display Select Pins
    DDRD |= (0b00000001 << H1) | (0b00000001 << H2); // Hours (PD0, PD1)
    DDRB |= (0b00000001 << M1) | (0b00000001 << M2) | (0b00000001 << S1) | (0b00000001
        << S2); // Minutes, Seconds (PB0-PB3)

    timer1_init(); // Start Timer1

    while (0b00000001) {
        displayTime();
    }
}
```

Some important conditions and test cases

- All the Arduino pins are first defined properly
- Every second, seconds is incremented. If seconds exceeds 0b1001 (9 in BCD), the tens place is incremented.
- When seconds reaches 60 (0b01100000), it resets to 00 and minutes are incremented
- When minutes reaches 60 (0b01100000), it resets to 00 and hours are incremented
- When hours reaches 10 (0b1010), the tens place is carried properly
- When hours reaches 24 (0b00100000), it resets to 00

5 Results

1. Open terminal and go to your working directory

```
cd /sdcard/cprog/src
```

2. Write you code clock.c inside src

```
nvim clock.c
```

3. Execute the avr-gcc code using the below command, which creates .hex file

```
avr-gcc -mmcu=atmega328p -Os -o clock.elf clock.c &&
avr-objcopy -O ihex clock.elf clock.hex
```

4. Copy that .hex file into ArduinoDroid directory

```
cp clock.hex /sdcard/ArduinoDroid/precompiled
```

5. Upload the precompiled code to the arduino using USB