-standard embedded system diploma
-Eng : Mohamed Tarek

Standard Embedded Diploma -Mini Project 2
(Stopwatch)

Project Title:
Stopwatch with Dual Mode (Increment and Countdown) Using ATmega32 and Seven-Segment Display.

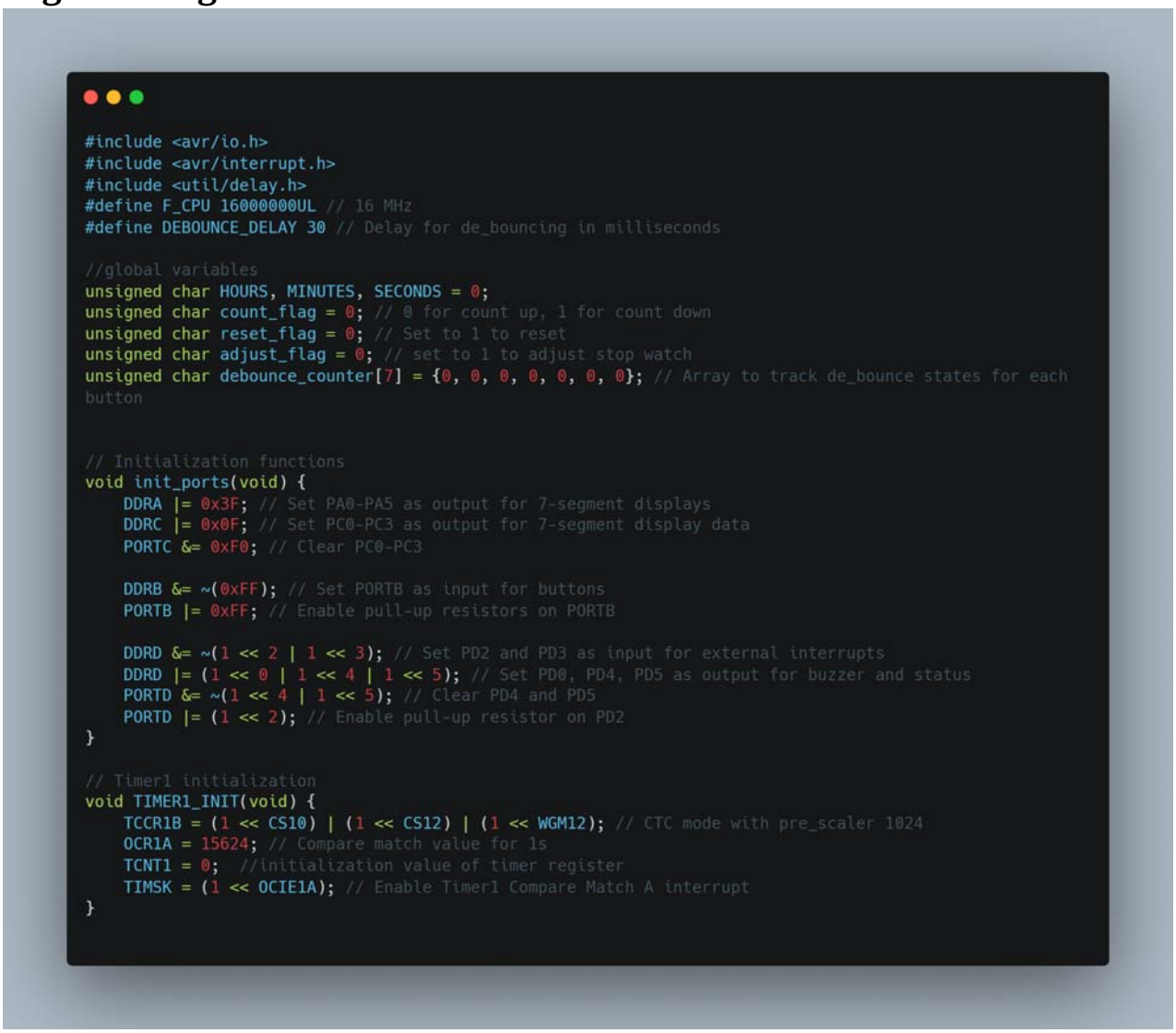
Operation Description:
• Increment Mode: By default, the stopwatch starts in increment mode, where the time counts up continuously from zero. The time is displayed across six seven-segment displays, showing hours, minutes, and seconds. The red LED on PD4 is turned on to indicate increment mode.
• Countdown Mode: When the mode toggle button is pressed, the stopwatch switches to countdown mode. In this mode, the user can set the desired countdown time using the hour, minute, and second adjustment buttons. The countdown begins once the start/resume button is pressed. A yellow LED on PD5 is turned on to indicate countdown mode. An alarm is triggered when the countdown reaches zero.

System Requirements:
1. Use ATmega32 Microcontroller.
2. System Frequency: 16Mhz
3. Configure Timer1 in ATmega32 with CTC mode to count the stopwatch time.
4. Connect the six 7-segments in the project using the multiplexed technique. You should use one 7447 decoders for all 7-segments and control the enable/disable for each 7-segement using a NPN BJT transistor connect to one of the MCU pins.

5. We can connect more than one 7-segment display by using the multiplexing method. In this method, at a time one 7-segment display is driven by the Microcontroller and the rest are OFF. It keeps switching the displays using transistors. Due to the persistence of vision, it appears as a

normal display. Use first 6-pins in PORTA as the enable/disable pins for the six 7- segments.
6. Stopwatch counting should start once the power is connected to the MCU.
7. Configure External Interrupt INT0 with falling edge. Connect a push button with the internal pull-up resistor. If a falling edge detected the stopwatch time should be reset.
8. Configure External Interrupt INT1 with raising edge. Connect a push button with the external pull-down resistor. If a raising edge detected the stopwatch time should be paused.
9. Configure External Interrupt INT2 with falling edge. Connect a push button with the internal pull-up resistor. If a falling edge detected the stopwatch time should be resumed.
10. Countdown Mode Setup: To operate the stopwatch in countdown mode, follow these steps:
a. Pause the Timer: Ensure that the timer is paused. This can be done by pressing the pause button connected to PD3 (INT1).
b. Toggle to Countdown Mode: Use the mode toggle button connected to PB7 to switch the stopwatch from the default increment mode to countdown mode.
c. Adjust the Countdown Start Time: Set the desired start time for the countdown using the adjustment buttons.
d. Resume the Countdown: Once the desired countdown time is set, press the resume button connected to PB2 (INT2) to start the countdown.
e. Buzzer Activation: When the countdown reaches zero, the buzzer connected to the alarm circuit will be triggered, alerting the user that the countdown has finished.

Programming code



```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#define F_CPU 16000000UL // 16 MHz
#define DEBOUNCE_DELAY 30 // Delay for de_bouncing in milliseconds

//global variables
unsigned char HOURS, MINUTES, SECONDS = 0;
unsigned char count_flag = 0; // 0 for count up, 1 for count down
unsigned char reset_flag = 0; // Set to 1 to reset
unsigned char adjust_flag = 0; // set to 1 to adjust stop watch
unsigned char debounce_counter[7] = {0, 0, 0, 0, 0, 0, 0}; // Array to track de_bounce states for each
button


// Initialization functions
void init_ports(void) {
    DDRA |= 0x3F; // Set PA0-PA5 as output for 7-segment displays
    DDRC |= 0x0F; // Set PC0-PC3 as output for 7-segment display data
    PORTC &= 0xF0; // Clear PC0-PC3

    DDRB &= ~(0xFF); // Set PORTB as input for buttons
    PORTB |= 0xFF; // Enable pull-up resistors on PORTB

    DDRD &= ~(1 << 2 | 1 << 3); // Set PD2 and PD3 as input for external interrupts
    DDRD |= (1 << 0 | 1 << 4 | 1 << 5); // Set PD0, PD4, PD5 as output for buzzer and status
    PORTD &= ~(1 << 4 | 1 << 5); // Clear PD4 and PD5
    PORTD |= (1 << 2); // Enable pull-up resistor on PD2
}

// Timer1 initialization
void TIMER1_INIT(void) {
    TCCR1B = (1 << CS10) | (1 << CS12) | (1 << WGM12); // CTC mode with pre_scaler 1024
    OCR1A = 15624; // Compare match value for 1s
    TCNT1 = 0;  //initialization value of timer register
    TIMSK = (1 << OCIE1A); // Enable Timer1 Compare Match A interrupt
}
```

```c
// Reset interrupt initialization
void RESET_INIT(void) {
    GICR |= (1 << INT0); // Enable external interrupt INT0
    MCUCR |= (1 << ISC01); // Falling edge trigger
}

// Pause interrupt initialization
void PAUSE_INIT(void) {
    GICR |= (1 << INT1); // Enable external interrupt INT1
    MCUCR |= (1 << ISC10) | (1 << ISC11); // Rising edge trigger
}

// Resume interrupt initialization
void RESUME_INIT(void) {
    GICR |= (1 << INT2); // Enable external interrupt INT2
    MCUCSR &= ~(1 << ISC2); // Falling edge trigger
}

// Decoder control function
void display(unsigned char num) {
    PORTC = (PORTC & 0xF0) | (num & 0x0F); // Send BCD value to decoder
}

// Function to display time on 6 multiplexed 7-segment displays
void display_time(void) {
    unsigned char h10 = HOURS / 10; // Tens digit of hours
    unsigned char h1 = HOURS % 10; // Units digit of hours
    unsigned char m10 = MINUTES / 10; // Tens digit of minutes
    unsigned char m1 = MINUTES % 10; // Units digit of minutes
    unsigned char s10 = SECONDS / 10; // Tens digit of seconds
    unsigned char s1 = SECONDS % 10; // Units digit of seconds

    // Display seconds
    PORTA = 0b100000; // Enable display 1
    display(s1); // Units digit of seconds
    _delay_ms(2);

    PORTA = 0b010000; // Enable display 2
    display(s10); // Tens digit of seconds
    _delay_ms(2);

    // Display minutes
    PORTA = 0b001000; // Enable display 3
    display(m1); // Units digit of minutes
    _delay_ms(2);

    PORTA = 0b000100; // Enable display 4
    display(m10); // Tens digit of minutes
    _delay_ms(2);

    // Display hours
    PORTA = 0b000010; // Enable display 5
    display(h1); // Units digit of hours
    _delay_ms(2);

    PORTA = 0b000001; // Enable display 6
    display(h10); // Tens digit of hours
    _delay_ms(2);
}

// Function to reset the stop watch
void reset_stopwatch(void) {
    PORTD &= ~(1 << 0);
    count_flag=0;
    HOURS = 0;
    MINUTES = 0;
    SECONDS = 0;
    display_time();
}
```

```c
// Increment time
void increment_time(void) {
    SECONDS++;
    if (SECONDS == 60) {
        SECONDS = 0;
        MINUTES++;
    }
    if (MINUTES == 60) {
        MINUTES = 0;
        HOURS++;
    }
    if (HOURS == 24) {
        reset_stopwatch();
    }
}

// Decrement time
void decrement_time(void) {
    if (SECONDS > 0) {
        SECONDS--;
    } else {
        if (MINUTES > 0) {
            SECONDS = 59;
            MINUTES--;
        } else {
            if (HOURS > 0) {
                SECONDS = 59;
                MINUTES = 59;
                HOURS--;
            } else {
                // Count down reached zero, trigger buzzer
                SECONDS = 0;
                MINUTES = 0;
                HOURS = 0;
                PORTD |= (1 << 0); // Activate buzzer on PD0
            }
        }
    }
}

// Function to stop Timer1
void stop_timer1() {
    TCCR1B &= ~(1 << CS12 | 1 << CS11 | 1 << CS10); // Clear prescaler bits
}

// Function to start Timer1
void start_timer1() {
    TCCR1B |= (1 << CS12) | (1 << CS10); // Set prescaler to 1024
}

// Functions to adjust hours, minutes, and seconds
void increment_hours() {
    if (HOURS < 23) {
        HOURS++;
    }
}
```

```
void decrement_hours() {
    if (HOURS > 0) {
        HOURS--;
    }
}

void increment_minutes() {
    if (MINUTES < 59) {
        MINUTES++;
    } else {
        MINUTES = 0;
    }
}

void decrement_minutes() {
    if (MINUTES > 0) {
        MINUTES--;
    } else {
        MINUTES = 59;
    }
}

void increment_seconds() {
    if (SECONDS < 59) {
        SECONDS++;
    } else {
        SECONDS = 0;
    }
}

void decrement_seconds() {
    if (SECONDS > 0) {
        SECONDS--;
    } else {
        SECONDS = 59;
    }
}

// De_bouncing function
void debounce(void) {
    _delay_ms(DEBOUNCE_DELAY); // Delay to de_bounce
}

// Adjustment function
void adjust_counter(void) {
    // Check button 0 (decrement hours)
    if (!(PINB & (1 << 0))) {
        if (debounce_counter[0] == 0) {
            decrement_hours();
            adjust_flag = 1;
            debounce_counter[0] = DEBOUNCE_DELAY; // Set de_bounce timer
        }
    }

    // Check button 1 (increment hours)
    if (!(PINB & (1 << 1))) {
        if (debounce_counter[1] == 0) {
            increment_hours();
            adjust_flag = 1;
            debounce_counter[1] = DEBOUNCE_DELAY;
        }
    }

    // Check button 3 (decrement minutes)
    if (!(PINB & (1 << 3))) {
        if (debounce_counter[2] == 0) {
            decrement_minutes();
            adjust_flag = 1;
            debounce_counter[2] = DEBOUNCE_DELAY;
        }
    }

    // Check button 4 (increment minutes)
    if (!(PINB & (1 << 4))) {
        if (debounce_counter[3] == 0) {
            increment_minutes();
            adjust_flag = 1;
            debounce_counter[3] = DEBOUNCE_DELAY;
        }
    }

    // Check button 5 (decrement seconds)
    if (!(PINB & (1 << 5))) {
        if (debounce_counter[4] == 0) {
            decrement_seconds();
            adjust_flag = 1;
            debounce_counter[4] = DEBOUNCE_DELAY;
        }
    }

    // Check button 6 (increment seconds)
    if (!(PINB & (1 << 6))) {
        if (debounce_counter[5] == 0) {
            increment_seconds();
            adjust_flag = 1;
            debounce_counter[5] = DEBOUNCE_DELAY;
        }
    }
}
```

```
// Interrupt Service Routines
ISR(INT0_vect) {
    reset_stopwatch();
}

ISR(INT1_vect) {
    stop_timer1(); // Pause the counter
}

ISR(INT2_vect) {
    start_timer1(); // Resume the counter
}

ISR(TIMER1_COMPA_vect) {
    if (count_flag == 0) {
        increment_time(); // Count up mode
        PORTD |= (1 << 4); // Status indicator
        PORTD &= ~(1 << 5);
        display_time();
    } else {
        decrement_time(); // Count down mode
        display_time();
        PORTD &= ~(1 << 4); // Clear status indicator
        PORTD |= (1 << 5); // Status indicator
    }
}

// Main function
int main() {
    init_ports();
    reset_stopwatch();
    SREG |= (1 << 7); // Enable global interrupts
    TIMER1_INIT();
    RESET_INIT();
    PAUSE_INIT();
    RESUME_INIT();
    start_timer1();

    while (1) {
        if (!(PINB & (1 << 7))) {
            count_flag = 1; // Set flag to logic 1 to activate count down mode
        }

        adjust_counter(); // Check and adjust stop watch timer

        // Decrease de_bounce counters for each button, if non-zero
        for (int i = 0; i < 6; i++) {
            if (debounce_counter[i] > 0) {
                debounce_counter[i]--;
            }
        }

        // Display the time continuously
        display_time();
    }
}
```

Code sinnpet explaination:

This code implements a digital stopwatch system using the ATmega32 microcontroller and six multiplexed 7-segment displays, with functionality for both count-up and countdown modes. The system also allows for manual adjustments of hours, minutes, and seconds, as well as pausing and resetting the stopwatch.

Key Components:

1. **Global Variables**:
    - `HOURS`, `MINUTES`, `SECONDS`: Track the current time in hours, minutes, and seconds.
    - `count_flag`: Indicates whether the stopwatch is in count-up (0) or countdown (1) mode.
    - `reset_flag`: Used to reset the stopwatch when necessary.
    - `adjust_flag`: Signals whether the stopwatch is in adjustment mode.
    - `debounce_counter[]`: Array used for debouncing the buttons to avoid erroneous multiple presses.
2. **Port Initialization (`init_ports`)**:
    - Configures ports for 7-segment display control, button inputs, and external interrupts.
    - PORTA controls which 7-segment display is active.
    - PORTC sends BCD values to the 7-segment decoder.
    - Buttons are connected to PORTB, with internal pull-up resistors enabled.
    - External interrupts are connected to PD2 (reset), PD3 (pause), and PD2 (resume).
3. **Timer Initialization (`TIMER1_INIT`)**:
    - Sets up Timer1 in CTC (Clear Timer on Compare Match) mode with a prescaler of 1024.
    - Timer1 triggers an interrupt every 1 second using the compare match value of 15624.
4. **External Interrupt Initialization**:
    - INT0 (reset), INT1 (pause), and INT2 (resume) interrupts are enabled with specific edge triggers.
    - Reset is triggered by a falling edge, pause by a rising edge, and resume by a falling edge.
5. **Display Control (`display`, `display_time`)**:
    - The function `display` sends the binary-coded decimal (BCD) value to the decoder.

- `display_time` controls the multiplexing of six 7-segment displays, showing the time by switching between each display rapidly.

6. **Timer Counting**:
    - `increment_time`: Increases the seconds, minutes, and hours as needed, resetting the stopwatch when it reaches 24 hours.
    - `decrement_time`: Decreases the seconds, minutes, and hours in countdown mode, triggering a buzzer when the countdown reaches zero.
7. **Button Adjustment**:
    - The `adjust_counter` function checks if any button for adjusting hours, minutes, or seconds has been pressed.
    - A debounce mechanism is implemented to avoid multiple erroneous button presses by introducing a delay between valid presses.
8. **Debouncing (`debounce`)**:
    - A simple delay is added to avoid multiple readings of a single button press. Each button has a debounce counter that prevents immediate successive activations.
9. **Interrupt Service Routines (ISRs)**:
    - `ISR(INT0_vect)`: Resets the stopwatch.
    - `ISR(INT1_vect)`: Pauses the stopwatch.
    - `ISR(INT2_vect)`: Resumes the stopwatch.
    - `ISR(TIMER1_COMPA_vect)`: Called every 1 second; increments or decrements the time based on `count_flag`.

    - **Operation Flow:**

1. The system starts with a reset, and the stopwatch begins counting up by default.
2. Users can toggle to countdown mode by pressing button PB7.
3. Users can adjust hours, minutes, and seconds using buttons PB0-PB6.
4. The stopwatch can be paused or resumed via external interrupts (INT1 and INT2).
5. The current time is continuously displayed on the 7-segment displays.
6. The stopwatch resets after reaching 24 hours or when the countdown reaches zero.

Summary

Your code effectively handles a digital stopwatch with functionality for counting up and down, adjusting the time, and displaying it on 7-segment displays. It uses Timer1 for timekeeping and external interrupts for control actions like reset, pause, and resume. Debouncing is implemented to ensure reliable button presses.

If you have any specific questions or need further clarification on any part, feel free to ask!

SOME NOTES:

The delay function (_delay_ms()) blocks the entire execution of the program for the specified time. In a system like yours, where the display is multiplexed and needs to be refreshed frequently, this blocking can cause interruptions. The refresh rate for multiplexed displays needs to be fast enough that the human eye perceives a continuous display. If you introduce delays, like in the button debouncing process, it slows down the refresh rate, making the display flicker or blink.

In summary, blocking delays interfere with time-sensitive tasks such as display updates because the processor is locked up waiting for the delay to finish. That's why delays in your button handling code are causing the display to blink — the display isn't being refreshed while the delay is occurring.

The array debounce_counter[] is used to store independent counters for each button, helping with non-blocking debouncing. Instead of using _delay_ms() for debouncing, which halts the program, we use these counters to track how long each button should be ignored after a press.

This array is a set of 7 counters (one for each button you're debouncing). Each index corresponds to a specific button. The value of each array element represents the time remaining for debouncing that button.

For example:

debounce_counter[0] is for the first button (PB0).
debounce_counter[1] is for the second button (PB1), and so on.
Using the Array for Debouncing:

When a button is pressed, its corresponding array element (debounce_counter[x]) is set to a debounce value (e.g., 20 ms).
Each time through the main loop, the counter is decremented by 1. Until the counter reaches zero, no further actions are taken for that button. This ensures that repeated presses within the debounce period are ignored.
The button can be acted upon again only when its debounce counter reaches zero, avoiding the need for a blocking delay.

Key Advantages:
Non-blocking: The program can continue performing tasks (like refreshing the display) while debouncing is handled in the background.
Independent counters: Each button has its own debounce counter, ensuring that button presses are handled independently and without interference.
This non-blocking method ensures that display updates occur smoothly, and debouncing is handled without interrupting the display refresh process.

PROTEUS SIMULATION: