Project report on

Stop Watch

Submitted by:

Stop Watch

Group Members:

1. Sachin Yadav: BT21ECE032
2. Ayush Shukla: BT21ECE017

A report submitted for the partial fulfillment of the requirements of the course
ECL-303 Hardware Description Languages

Submission Date: 20/11/2023

Under the guidance of:
Dr. Mayank B. Thacker
Department of Electronics and Communication Engineering

भारतीय सूचना प्रौद्योगिकी संस्थान, नागपुर
Indian Institute of Information Technology, Nagpur

Table of Contents:

Chapter No.	**Particular**	**Page No.**
1	Introduction	2
2	Code	4-9
3	Working	11-12
4	References	12

Chapter 1: Introduction

Board and Layout:

Our project is implemented on DE10, a versatile hardware platform that provides the necessary resources for FPGA development. The layout of our design includes modular components representing seconds, minutes, hours, and milliseconds, each contributing to the overall functionality of the stopwatch.

Key Components:

Clock Input (clk): Serves as the heartbeat of our system, determining the timing of state transitions and overall operation.

Control Signals (start, stop, reset): External signals that trigger state transitions and control the stopwatch's behavior.

Segment Display Output (seg): The 7-segment display output visually represents the elapsed time in a user-friendly format.

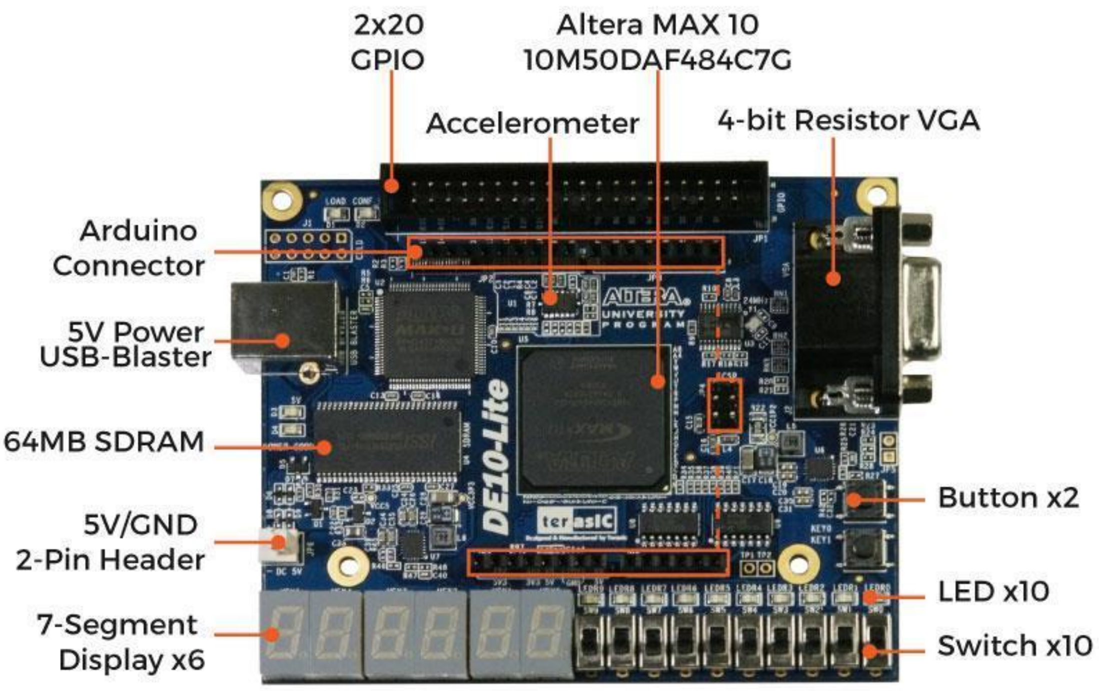

Fig. 1: ESP32 board

Chapter 2: Code

```
module stopwatch(
    input wire clk,
    input wire reset,
    input wire start,
    input wire stop,
    output wire [6:0]ssd5,ssd4,ssd3,ssd2,ssd1,ssd0


);
wire seconds;
delay d22(clk,seconds);
display dp5(min_tens,ssd5);
display dp4(min_ones,ssd4);
display dp3(sec_tens,ssd3);
display dp2(sec_ones,ssd2);
display dp1(ms_tens,ssd1);
display dp0(min_ones,ssd0);


reg [3:0] sec_tens;
reg [3:0] sec_ones;
reg [3:0] min_tens;
reg [3:0] min_ones;
reg [3:0] hour_tens;
reg [3:0] hour_ones;
reg [3:0] ms_tens;
reg [3:0] ms_ones;

reg [3:0] sec_tens_next;
reg [3:0] sec_ones_next;
reg [3:0] min_tens_next;
reg [3:0] min_ones_next;
reg [3:0] hour_tens_next;
reg [3:0] hour_ones_next;
reg [3:0] ms_tens_next;
reg [3:0] ms_ones_next;

reg [3:0] state;
reg [3:0] state_next;

always @(posedge seconds or posedge reset) begin
    if (reset) begin
        sec_tens <= 4'b0;
        sec_ones <= 4'b0;
        min_tens <= 4'b0;
        min_ones <= 4'b0;
        hour_tens <= 4'b0;
        hour_ones <= 4'b0;
        ms_tens <= 4'b0;
        ms_ones <= 4'b0;
        state <= 4'b0000;
    end else begin
        sec_tens <= sec_tens_next;
        sec_ones <= sec_ones_next;
        min_tens <= min_tens_next;
        min_ones <= min_ones_next;
        hour_tens <= hour_tens_next;
        hour_ones <= hour_ones_next;
        ms_tens <= ms_tens_next;
        ms_ones <= ms_ones_next;
        state <= state_next;
    end
end

always @(state or start or stop) begin
    case (state)
        4'b0000: begin // Idle state
            if (start) begin
                state_next = 4'b0001; // Transition to counting state
            end else begin
                state_next = 4'b0000; // Stay in idle state
            end
        end
        4'b0001: begin // Counting state
            if (stop) begin
```

```
                state_next = 4'b0010; // Transition to paused state
            end else begin
                state_next = 4'b0001; // Stay in counting state
            end
        end
        4'b0010: begin // Paused state
            if (start) begin
                state_next = 4'b0001; // Transition to counting state
            end else if (reset) begin
                state_next = 4'b0000; // Transition to idle state
            end else begin
                state_next = 4'b0010; // Stay in paused state
            end
        end
        default: begin
            state_next = 4'b0000; // Default to idle state
        end
    endcase
end

always @(posedge seconds) begin
    case (state)
        4'b0000: begin // Idle state
            sec_tens_next = sec_tens;
            sec_ones_next = sec_ones;
            min_tens_next = min_tens;
            min_ones_next = min_ones;
            hour_tens_next = hour_tens;
            hour_ones_next = hour_ones;
            ms_tens_next = ms_tens;
            ms_ones_next = ms_ones;
        end
        4'b0001: begin // Counting state
            ms_ones_next = ms_ones + 1;
            if (ms_ones_next == 10) begin
                ms_ones_next = 0;
                ms_tens_next = ms_tens + 1;
                if (ms_tens_next == 10) begin
                    ms_tens_next = 0;
                    sec_ones_next = sec_ones + 1;
                    if (sec_ones_next == 10) begin
                        sec_ones_next = 0;
                        sec_tens_next = sec_tens + 1;
                        if (sec_tens_next == 6) begin
                            sec_tens_next = 0;
                            min_ones_next = min_ones + 1;
                            if (min_ones_next == 10) begin
                                min_ones_next = 0;
                                min_tens_next = min_tens + 1;
                                if (min_tens_next == 6) begin
```

```
                            min_tens_next = 0;
                            hour_ones_next = hour_ones + 1;
                            if (hour_ones_next == 10) begin
                                hour_ones_next = 0;
                                hour_tens_next = hour_tens + 1;
                                if (hour_tens_next == 10) begin
                                    hour_tens_next = 0;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
4'b0010: begin // Paused state
    sec_tens_next = sec_tens;
    sec_ones_next = sec_ones;
    min_tens_next = min_tens;
    min_ones_next = min_ones;
    hour_tens_next = hour_tens;
    hour_ones_next = hour_ones;
    ms_tens_next = ms_tens;
    ms_ones_next = ms_ones;
```

```
        end
        default: begin
            sec_tens_next = sec_tens;
            sec_ones_next = sec_ones;
            min_tens_next = min_tens;
            min_ones_next = min_ones;
            hour_tens_next = hour_tens;
            hour_ones_next = hour_ones;
            ms_tens_next = ms_tens;
            ms_ones_next = ms_ones;
        end
    endcase
end
endmodule
```

Code for Delay

```
module delay (clk,seconds);
output reg seconds;
input clk;
reg [26:0] count;

always @(posedge clk)
begin
           if (count == 27'd100_000) begin
        count   = 0;
```

```
                  seconds=~seconds;
    end else begin
        count   <= count + 1'b1;
    end
end
endmodule
```

Code for Display

```
module display(in,out);
input [3:0]in;
output reg [6:0]out;

always @(in)
begin
          case(in)
                    4'b0000: out = 7'b0000001;
                    4'b0001: out = 7'b1001111;
                    4'b0010: out = 7'b0010010;
                    4'b0011: out = 7'b0000110;
                    4'b0100: out = 7'b1001100;
                    4'b0101: out = 7'b0100100;
                    4'b0110: out = 7'b0100000;
                    4'b0111: out = 7'b0001111;
                    4'b1000: out = 7'b0000000;
```

```
        4'b1001: out = 7'b0000100;
        4'b1010: out = 7'b0001000;
        4'b1011: out = 7'b1100000;
        4'b1100: out = 7'b0110001;
        4'b1101: out = 7'b1000010;
        4'b1110: out = 7'b0110000;
        4'b1111: out = 7'b0111000;
        default: out = 7'b1111111;
    endcase
end

endmodule
```

Working:

Initialization:

When the system is initialized or reset (reset signal asserted), all timekeeping values are set to zero, and the state is set to Idle.

State Machine:

In the Idle state, the stopwatch waits for the start signal. If start is asserted, it transitions to the Counting state.

In the Counting state, the milliseconds are incremented with each clock cycle. When the milliseconds reach 10, they reset to zero, and the tens of milliseconds (ms_tens) are incremented. This process continues for seconds, minutes, and hours, rolling over as needed.

If the stop signal is asserted, the system transitions to the Paused state.

In the Paused state, the stopwatch retains the current time values. It can transition back to the Counting state if the start signal is asserted or return to the Idle state if the system is reset.

7-Segment Display:

The seg output is a 7-bit vector representing the milliseconds and the individual digits of seconds, minutes, and hours. This vector can be used to drive a 7-segment display for visual representation.

The overall functionality of this Verilog module is to implement a stopwatch with the ability to start, stop, reset, and display the time on a 7-segment display. The time is measured in hours, minutes, seconds, and tenths of a second. The state machine controls the transitions between idle, counting, and paused states. The counting logic increments the time units based on the clock signal.

References:

https://www.chipverify.com/tutorials/verilog

https://www.instructables.com/How-to-use-Verilog-and-Basys-3-to-do-stop-watch/