Project report on

STOPWATCH

Submitted by:

Verilog Timekeeper

Group Members:

1. **NISARG JAIN BT21ECE024**
2. **ARYAN RANJAN BT21ECE048**
3. **HRITIK GUPTA BT21ECE049**

A report submitted for the partial fulfilment of the requirements of the course

ECL-303 Hardware Description Languages

Submission Date: 17/11/2023

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 1: Introduction

- The project's goal is to design and simulate the stopwatch using Verilog HDL, involving testing the design and verifying results on an FPGA board.
- As part of the project, the elapsed time will be displayed on the FPGA board, showcasing practical applications of the implemented stopwatch.
- In the field of Verilog, the stopwatch becomes a fascinating topic for exploration and implementation using Verilog HDL.
- A stopwatch, functioning as a digital chronometer, is a fundamental time-measuring instrument designed for precise and reliable measurement of elapsed time intervals.
- It operates through a combination of digital circuits and user interfaces, enabling precise start, stop, and reset functions for timing operations.
- Its simplicity and versatility make it an essential tool for measuring time intervals in various contexts, including sports, scientific experiments, and more.

Chapter 2: Description

- The stopwatch algorithm utilizes three fundamental timers: millisecond, second, and minute counters.
- Initially, The Verilog module, "millisecond_delay," serves the purpose of generating a delay of a predetermined duration, relying on a 50 MHz clock input.
- Specifically, the objective is to establish a delay of 500,000 clock cycles, translating to a 10 ms delay when operating with a 50 MHz clock frequency.
- This module's output is invoked multiple times, precisely 100 times, to accumulate and achieve a cumulative delay of 1 second.
- The utilization of this Verilog module within the stopwatch project ensures precise timing control, allowing for accurate time measurements and synchronization with other components of the system.
- Then all counters are set to zero, establishing the baseline for time measurement.
- The millisecond timer starts counting when the start button is pressed, incrementing until it reaches 100.
- When the millisecond counter hits 100, it resets to 0, and the second counter begins counting.

- Once the second counter reaches 59, it resets and increments the minute counter.

- If the minute counter reaches 99, all three counters reset, and the stopwatch stops.

- Pressing the stop button halts all counters, displaying the stopped time.

- Pressing the reset button resets all counters to zero, preparing the stopwatch for a new measurement cycle.

- The calculated time is displayed in the format (minutes:seconds:milliseconds).

- Two additional modules enhance the functionality of this stopwatch project: one module for converting binary to BCD (Binary-Coded Decimal) and another for BCD to 7-segment display.

- The binary to BCD module is instrumental in converting the digital output from the stopwatch counters into a format suitable for driving the 7-segment display.

- This conversion is crucial for accurately presenting the measured time in a human-readable format on the display.

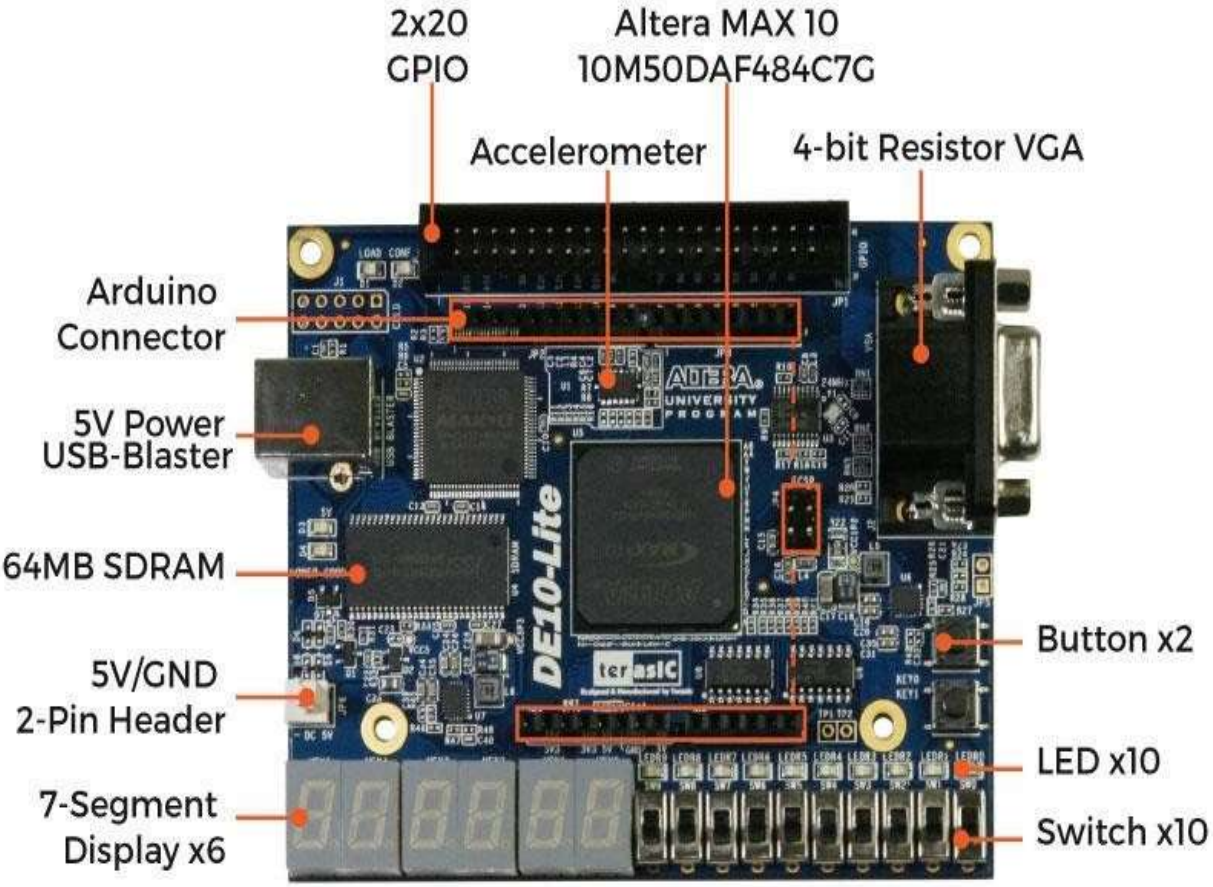

Fig. 1: ESP32 board

Chapter 3: Code

Stopwatch module

```
module millisecond_delay(
    input wire clk, // 50 MHz clock input
    output wire delay_done
);
reg [31:0] counter;

always @(posedge clk)
        begin
    if (counter < 500_000) // 10 millisecond at 50 MHz
    counter <= counter + 1'b1;
else counter <= 32'b0;
end

    assign delay_done = (counter == 500_000);
endmodule

    module stopwatch(
        output [7:0] millicount,
        output [7:0] mincount,
        output [7:0] seccount,
        input clk, start, stop, reset,
        output delay,
        output wire [6:0] HEX0,
        output wire [6:0] HEX1,
        output wire [6:0] HEX2,
        output wire [6:0] HEX3,
        output wire [6:0] HEX4,
        output wire [6:0] HEX5

    );
reg [7:0] millicount_reg;
reg [7:0] seccount_reg;
```

```
reg [7:0] mincount_reg;
reg counting;

millisecond_delay(clk, delay);
always @(reset, start, stop) begin
   if (reset) begin

   counting<= 1'b0;
end else if (start) begin
       counting <= 1'b1;
end else if (stop) begin
       counting <= 1'b0;

end
   end
      always @(posedge delay or posedge reset)
         begin
   if (reset)
      begin
   millicount_reg = 8'd0;
mincount_reg = 8'd0;
seccount_reg = 8'd0;
end else begin

   if (counting) begin
      millicount_reg = millicount_reg + 1;
if (millicount_reg >= 8'd99)
   begin
      seccount_reg = seccount_reg + 1;
millicount_reg = 8'd0;
end if (seccount_reg == 8'd59) begin
   mincount_reg = mincount_reg + 1;
seccount_reg = 8'd0;
end if (mincount_reg == 8'd59) begin
   mincount_reg = 8'd0;
seccount_reg = 8'd0;
millicount_reg = 8'd0;
end
```

```
    end
        end
            end
                assign millicount = millicount_reg;
assign seccount = seccount_reg;
assign mincount = mincount_reg;

wire [11:0] bcd_mili, bcd_sec, bcd_min;
wire [3:0] BCD0, BCD1, BCD2, BCD3, BCD4, BCD5;

hex2bcd h1(.bin(millicount_reg), .bcd(bcd_mili));
hex2bcd h2(.bin(seccount_reg), .bcd(bcd_sec));
hex2bcd h3(.bin(mincount_reg), .bcd(bcd_min));

assign BCD0 = bcd_mili [3:0];
assign BCD1 = bcd_mili [7:4];
assign BCD2 = bcd_sec  [3:0];
assign BCD3 = bcd_sec  [7:4];
assign BCD4 = bcd_min [3:0];
assign BCD5 = bcd_min [7:4];

bcd27seg b1(.bcd(BCD0), .HEX(HEX0));
bcd27seg b2(.bcd(BCD1), .HEX(HEX1));
bcd27seg b3(.bcd(BCD2), .HEX(HEX2));
bcd27seg b4(.bcd(BCD3), .HEX(HEX3));
bcd27seg b5(.bcd(BCD4), .HEX(HEX4));
bcd27seg b6(.bcd(BCD5), .HEX(HEX5));

endmodule
```

hex2bcd module

```
module hex2bcd(
  input [7:0] bin,
  output reg [11:0] bcd
  );

integer i;

always @(bin) begin
   bcd=0;
   for (i=0;i<8;i=i+1) begin
  if (bcd[3:0] >= 5) bcd[3:0] = bcd[3:0] + 3;
       if (bcd[7:4] >= 5) bcd[7:4] = bcd[7:4] + 3;
       if (bcd[11:8] >= 5) bcd[11:8] = bcd[11:8] + 3;
       bcd = {bcd[10:0],bin[7-i]};
   end
end
endmodule
```

bcd27seg module

```
module bcd27seg(
   bcd,
   HEX
);
   input       [3:0] bcd;
   output reg  [6:0] HEX;


   always @ (bcd) begin
      case (bcd)

                4'd0 : HEX = 7'b0000001;
                4'd1 : HEX = 7'b1001111;
                4'd2 : HEX = 7'b0010010;
                4'd3 : HEX = 7'b0000110;
                4'd4 : HEX = 7'b1001100;
                4'd5 : HEX = 7'b0100100;
                4'd6 : HEX = 7'b0100000;
                4'd7 : HEX = 7'b0001111;
                4'd8 : HEX = 7'b0000000;
                4'd9 : HEX = 7'b0000100;
                4'd10: HEX = 7'b0001000;
                4'd11: HEX = 7'b1100000;
                4'd12: HEX = 7'b0110001;
                4'd13: HEX = 7'b1000010;
                4'd14: HEX = 7'b0110000;
                4'd15: HEX = 7'b0111000;


          default: HEX = 7'b0000110;
      endcase
   end

endmodule
```

References:

1. https://en.wikipedia.org/wiki/Double_dabble