1. Objectives

The main objectives of the lab are:

- Designing a sequential digital system given a specification of its function.
- Implementing hierarchy to build complex synchronous sequential logic circuits connecting together multiple simpler elements.
- Implementing and debugging complex sequential logic circuits

2. Introduction

In this lab we focused on the design of a stopwatch circuit using a top-down approach, where we started by understanding the design process by which we clearly define the problem to be solved, outlined the functions of a desired circuit, and then combined digital building blocks to realize the desired function of the circuit.At first we designed a block diagram of stopwatch using various combinational and sequential circuits with the required circuit specifications.

We created a RTL,Simulation, and Constraints and copied seven segment controller code from Lab 7 along with a constraint file to implement it on the Nexys A7 FPGA board.We created a module for the single pulser, switch debouncer, and BCD counter that was provided to us.Likewise we created module for clk divider , register, finite state machine module file. Then we created a top-level file named stopwatch.sv that contains the top-level stopwatch module with all the modules used in stopwatch. And we created a stimulus-only testbenches to make sure that these modules work properly.After that we created a constraints file to connect the input and output ports of the stopwatch module to the appropriate switch inputs and display outputs.Then we synthesized the code , generated the bitstream and test your circuit on the Nexys A7 board.

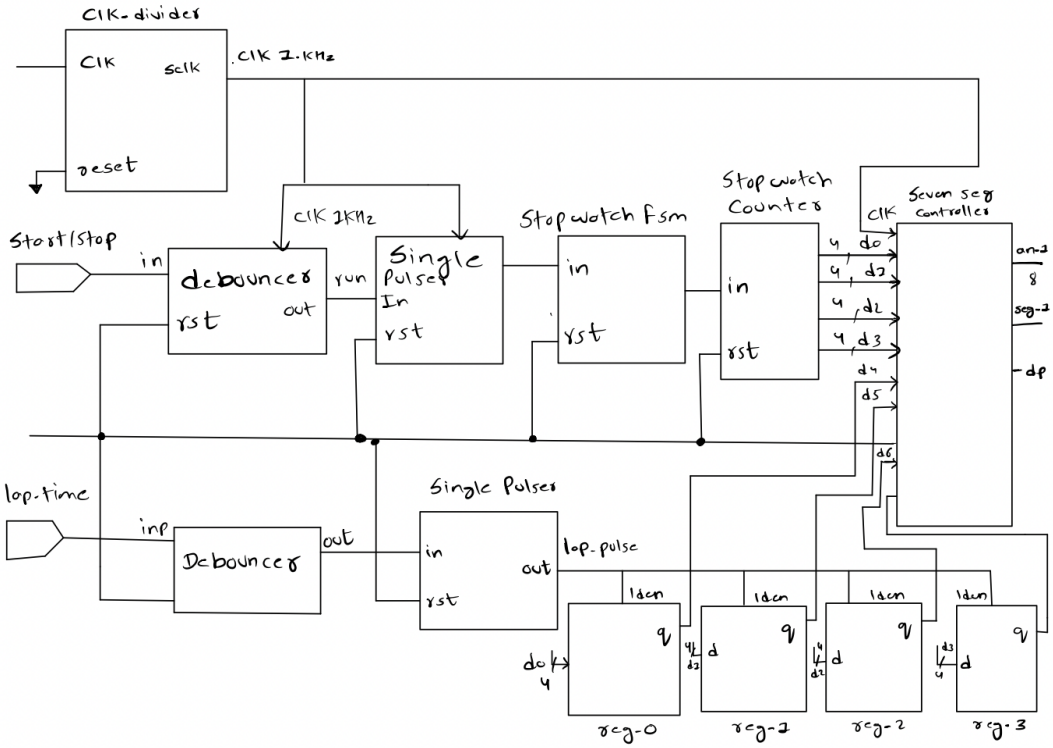

Figure 1: Stopwatch block diagram

3 . Result

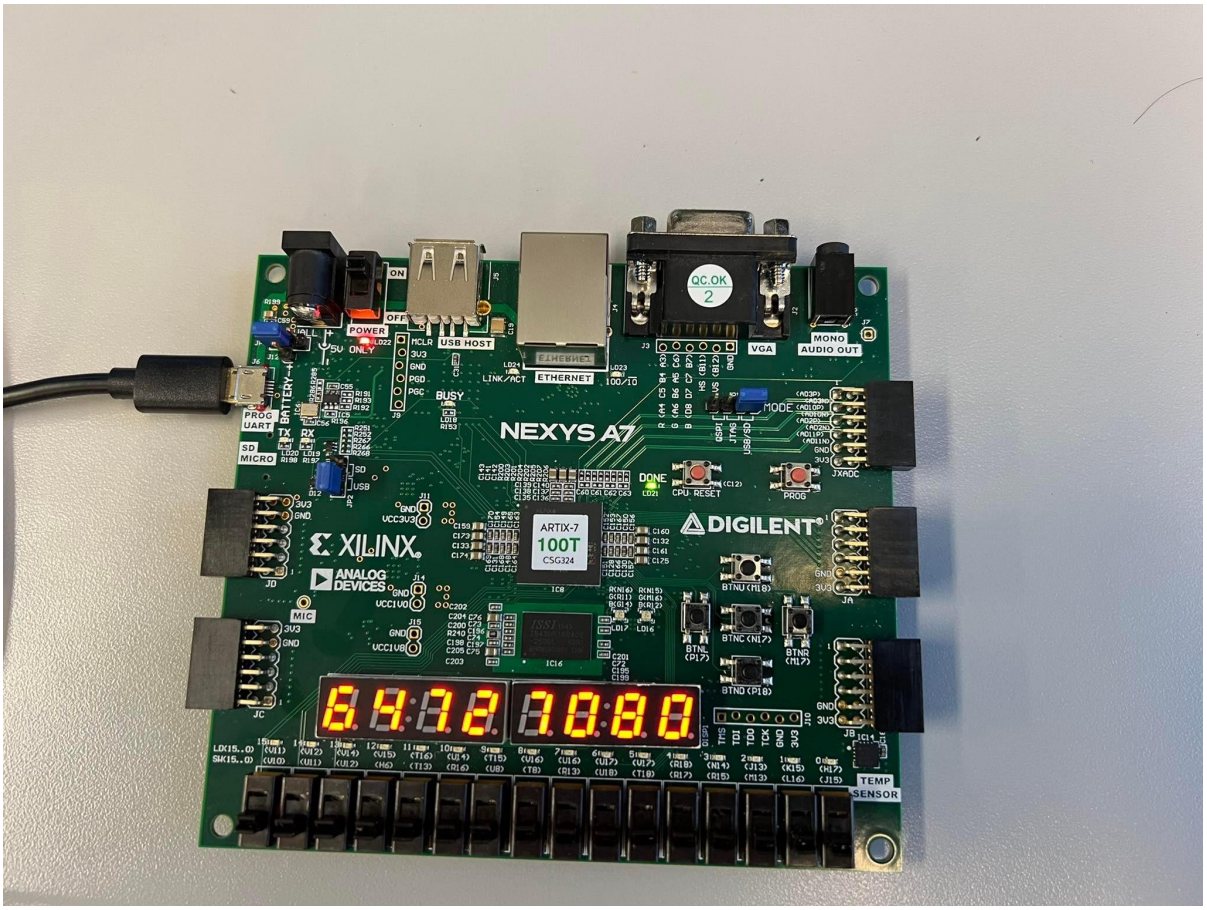

Figure 2: Working StopWatch Diagram

Code Listings

System Verilog File for clk divider

```
module clkdiv(input logic clk, input logic reset, output logic
sclk);
   parameter DIVFREQ = 100;  // desired frequency in Hz (change as
needed)
   parameter DIVBITS = 26;   // enough bits to divide 100MHz down
to 1 Hz
   parameter CLKFREQ = 100_000_000;
   parameter DIVAMT = (CLKFREQ / DIVFREQ) / 2;

   logic [DIVBITS-1:0] q;

   always_ff @(posedge clk)
     if (reset) begin
```

```
          q <= 0;
          sclk <= 0;
      end
      else if (q == DIVAMT-1) begin
          q <= 0;
          sclk <= ~sclk;
      end
      else q <= q + 1;
endmodule
```

System Verilog File for Counter

```
module count_3bit ( input logic clk, rst,
                    output logic [2:0]q ); //declare inputs and
outputs

    always_ff @ (posedge clk)  //initialize clock
        if (rst) q <= 3'd0;    //Condition 1: Check if reset is
positive, it will return 3'd0
         else q <= q + 3'd1;   //else it will return 3'd1

endmodule
```

System Verilog File for BCD Counter

```
module counter_bcd(input logic clk, rst, enb,
                   output logic [3:0] q,
                output logic      carry);

assign carry = ((q == 4'd9) && enb);

  always_ff @( posedge clk )
    begin
       if (rst || carry) q <= 0;
       else if (enb) q <= q + 1;
    end

endmodule
```

System Verilog File for Debouncer

```
module debounce(input logic clk, pb, output logic pb_debounced);
   parameter CLKFREQ = 1000;    // clock frequency in Hertz
```

```
   parameter DEBOUNCE_MS = 10;  // desired debounce delay in
   milliseconds
   parameter CTRBITS = $clog2(DEBOUNCE_MS*CLKFREQ/1000);

   logic pb_q1, pb_q2, pb_edge, carry;
   logic [CTRBITS:0] count;  // use extra bit as carry out

   assign carry = count[CTRBITS];

   always_ff @(posedge clk)
     begin
      pb_q1 <= pb;
      pb_q2 <= pb_q1;
      if (carry) pb_debounced <= pb_q2;
     end

   assign pb_edge = pb_q1 ^ pb_q2;  // rising or falling edge on
pb

   always_ff @(posedge clk)
     if (pb_edge) count <= '0;
     else if (!carry) count <= count + 1;

endmodule // debounce
```

System Verilog code for 3_8decoder

```
module dec_3_8(input logic [2:0] a,
output logic [7:0] y_1);

logic [7:0]y;
always_comb
    begin
        case (a)
        3'd0: y = 8'b00000001;
        3'd1: y = 8'b00000010;
        3'd2: y = 8'b00000100;
        3'd3: y = 8'b00001000;
        3'd4: y = 8'b00010000;
        3'd5: y = 8'b00100000;
        3'd6: y = 8'b01000000;
```

```
        3'd7: y = 8'b10000000;

        default: y = 4'b00000000;
        endcase
    end
  assign y_1=~y;
endmodule
```

System Verilog Top level code for Multiple Counter

```
module multiple_bcd(input logic en, reset,clk,
                    output logic [3:0] d1,d2,d3,d4
                    );
                     logic carry;
                    logic carry1;
                    logic carry2;
                    logic carry3;
                    logic carryd;
                    logic [3:0]d5;

counter_bcd M_INST1 (.rst(reset), .enb(en), .clk(clk), .q(d5),
.carry(carry));
counter_bcd M_INST2 (.rst(reset), .enb(carry), .clk(clk), .q(d1),
.carry(carry1));
counter_bcd M_INST3 (.rst(reset), .enb(carry1), .clk(clk), .q(d2),
.carry(carry2));
counter_bcd M_INST4 (.rst(reset), .enb(carry2), .clk(clk), .q(d3),
.carry(carry3));
counter_bcd M_INST5 (.rst(reset), .enb(carry3), .clk(clk), .q(d4),
.carry(carryd));
endmodule
```

System Verilog code for Mux

```
module mux_8_1(input logic [3:0] d0, d1, d2, d3, d4, d5, d6, d7,
               input logic [2:0] s,
               output logic [3:0] y );

    always_comb
        case (s)
```

```
        3'b000: y = d0;
        3'b001: y = d1;
        3'b010: y = d2;
        3'b011: y = d3;
        3'b100: y = d4;
        3'b101: y = d5;
        3'b110: y = d6;
        3'b111: y = d7;
        default: y = 3'd0;

    endcase
endmodule
```

System Verilog code for Register

```
module register(input logic clk, en,
             input logic [3:0] d,
             output logic [3:0] q);
always_ff @(posedge clk)
     begin
        if (en)q<=d;
     end

endmodule
```

System Verilog code for Sevenseg_controller

```
module sevenseg_control(input logic [3:0] d0,d1,d2,d3,d4,d5,d6,d7,
                        input logic clk, rst,
                        output logic [7:0] an_1,
                        output logic [6:0] segs_1,
                        output logic DP
                        );

                        logic [2:0] q;
                        logic [3:0] y;

  count_3bit U_INST3 ( .clk(clk), .rst(rst), .q(q));
  mux_8_1 U_INST4 (.d0(d0), .d1(d1), .d2(d2), .d3(d3), .d4(d4),
  .d5(d5), .d6(d6), .d7(d7), .s(q), .y(y));
  dec_3_8 U_INST2 ( .a(q), .y_1(an_1) );
  sevenseg_hex U_INST1  ( .data(y), .segs_1(segs_1) );

    always_comb
```

```
  begin
  if (q==2 || q==6)
    DP = 1;
    else DP = 0;
  end
endmodule
```

System Verilog Sevenseg_hex code

```
module sevenseg_hex(input logic [3:0] data,
output logic [6:0] segs_1);

always_comb
    begin
        case (data)

    4'd0: segs_1 = 7'b0000001;
    4'd1: segs_1 = 7'b1001111;
    4'd2: segs_1 = 7'b0010010;
    4'd3: segs_1 = 7'b0000110;
    4'd4: segs_1 = 7'b1001100;
    4'd5: segs_1 = 7'b0100100;
    4'd6: segs_1 = 7'b0100000;
    4'd7: segs_1 = 7'b0001111;
    4'd8: segs_1 = 7'b0000000;
    4'd9: segs_1 = 7'b0001100;
    4'd10: segs_1 = 7'b0001000;
    4'd11: segs_1 = 7'b1100000;
    4'd12: segs_1 = 7'b1110010;
    4'd13: segs_1 = 7'b1000010;
    4'd14: segs_1 = 7'b0110000;
    4'd15: segs_1 = 7'b0111000;

    default: segs_1 = 7'b0000000;
        endcase
    end


endmodule
```

System Verilog single pulser code

```
module single_pulser(input logic clk, din,
```

```
                    output logic d_pulse);
   logic dq1, dq2;

   always_ff @(posedge clk)
     begin
        dq1 <= din;
        dq2 <= dq1;
     end

   assign d_pulse = dq1 & ~dq2;
endmodule // single_pulser
```

System Verilog code for Finite State Machine

```
module stopwatch_fsm(input logic ststop,
                     input logic clk,
                     input logic reset,
                     output logic run);


  typedef enum logic [1:0] {
          START = 2'b00,
          STOP = 2'b01
  } state_t;

  state_t p_s, n_s;

    always_ff @(posedge clk)
    begin
      if (reset)
      p_s <= START;
      else p_s <= n_s;
    end

  always_comb
    begin

       case (p_s)
        START:
         begin
         run  = 1;
            if(ststop == 1)
                n_s = STOP;
```

```
            else
                 n_s = START;
         end
        STOP:
        begin
            run = 0;
            if(ststop == 1 )
            n_s = START;
            else
            n_s = STOP;
          end
        default:
            begin
                run = 0;
                n_s = START;
             end

      endcase
  end

endmodule
```

System Verilog Top Module code for StopWatch

```
module lab10_top(input logic clk_100MHz,
                 input logic reset,
                 input logic startstop,
                 input logic lap_time,
                 output logic [7:0] an_1,
                 output logic [6:0] segs_1,
                 output logic DP
                 );

                 logic clk_1KHz;
                 logic run_deb;
                 logic run2;
                 logic run3;
                 logic [3:0] d1,d2,d3,d4,d5,d6,d7,d8;
                 logic lap_deb;
                 logic lap_pulse;
```

```
    clkdiv #(.DIVFREQ(1000)) U_CLKDIV( .clk(clk_100MHz),
.reset(1'b0), .sclk(clk_1KHz) );

    debounce U_INST1 (.clk(clk_1KHz),
.pb(startstop),.pb_debounced(run_deb));

    single_pulser U_INST4 (.clk(clk_1KHz), .din(run_deb),
    .d_pulse(run2) );
    debounce U_INST2 (.clk(clk_1KHz),
    .pb(lap_time),.pb_debounced(lap_deb));
    single_pulser U_INST7 (.clk(clk_1KHz), .din(lap_deb),
    .d_pulse(lap_pulse) );
    stopwatch_fsm U_INST5 (.clk(clk_1KHz), .reset(reset),
    .ststop(run2), .run(run3));
    multiple_bcd U_INST6 (.clk(clk_1KHz), .en(run3),
.reset(reset),
    .d1(d1), .d2(d2), .d3(d3), .d4(d4));
    sevenseg_control U_INST3(.clk(clk_1KHz), .rst(reset), .d0(d1),
    .d1(d2), .d2(d3), .d3(d4), .d4(d5), .d5(d6), .d6(d7), .d7(d8),
    .an_1(an_1), .segs_1(segs_1), .DP(DP)  );

    register U_INST8 (.clk(clk_1KHz), .d(d1), .en(lap_pulse),
.q(d5)
    ,.rst(reset));
    register U_INST9 (.clk(clk_1KHz), .d(d2), .en(lap_pulse),
    .q(d6), .rst(reset));
    register U_INST10 (.clk(clk_1KHz), .d(d3), .en(lap_pulse),
    .q(d7), .rst(reset));
    register U_INST11 (.clk(clk_1KHz), .d(d4), .en(lap_pulse),
    .q(d8), .rst(reset));
endmodule
```

StopWatch Constraint File

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk_1KHz }]; #IO_L12P_T1_MRCC_35 Sch=clk_1KHz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports {clk_1KHz}];
```

```
##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { startstop }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 }
[get_ports { reset }]; #IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 }
[get_ports { lap_time }]; #IO_L10N_T1_D15_14 Sch=btnr

##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 }
[get_ports { segs_1[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 }
[get_ports { segs_1[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 }
[get_ports { segs_1[2]}]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 }
[get_ports { segs_1[3]}]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 }
[get_ports { segs_1[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 }
[get_ports { segs_1[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 }
[get_ports { segs_1[6] }]; #IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 }
[get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 }
[get_ports { an_1[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 }
[get_ports { an_1[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 }
[get_ports { an_1[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 }
[get_ports { an_1[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 }
[get_ports { an_1[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 }
[get_ports {an_1[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 }
[get_ports { an_1[6]}]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 }
[get_ports { an_1[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

Test Bench File for Multiple BCD

```
module multiple_bcd_tb;
            logic en, reset,clk;
            logic [3:0] d1,d2,d3,d4;
            logic carry, carry1, carry2;


            multiple_bcd DUV( .en(en), .reset(reset),
.clk(clk), .d1(d1), .d2(d2), .d3(d3), .d4(d4));
            //, .carry(carry), .carry1(carry1),
.carry2(carry2)

parameter CLK_PD = 10;
always
  begin
   clk = 1'b0; #(CLK_PD/2);
   clk = 1'b1; #(CLK_PD/2);
  end

    initial begin
    en = 0;
    reset = 0;
    clk = 0;
    @(posedge clk) #1;
        reset=1;
    repeat (2) @(posedge clk); #1;
     en = 1;
     reset = 0;

    repeat (3) @(posedge clk); #1;

     reset = 0;

    repeat (10000) @(posedge clk); #1;

     reset = 0;

$stop;
end
```

```
endmodule
```

Stopwatch Finite State Machine Test Bench File

```
module stopwatch_fsm_tb;
            logic ststop, clk, reset;
            logic run;

           stopwatch_fsm U_INST5( .ststop, .clk, .reset,
.run);

parameter CLK_PD = 10;
always
  begin
   clk = 1'b0; #(CLK_PD/2);
   clk = 1'b1; #(CLK_PD/2);
  end

    initial begin
    ststop = 0;
    clk = 0;
    reset = 0;
    @(posedge clk) #1;
        reset=1;
    repeat (2) @(posedge clk); #1;
        ststop = 0;
        clk = 1;
        reset = 0;
    repeat (3) @(posedge clk); #1;
        ststop = 1;
        clk = 0;
        reset = 0;
    repeat (4) @(posedge clk); #1;
        ststop = 0;
        clk = 0;
        reset = 1;
$stop;
end
endmodule
```

4. Conclusion:

In this lab, we've designed a sequential digital system given a specification of its function.Using the concept of hierarchy we were able to build complex synchronous sequential logic circuits connecting together multiple simpler elements.We also learned how to debug complex sequential logic circuits.