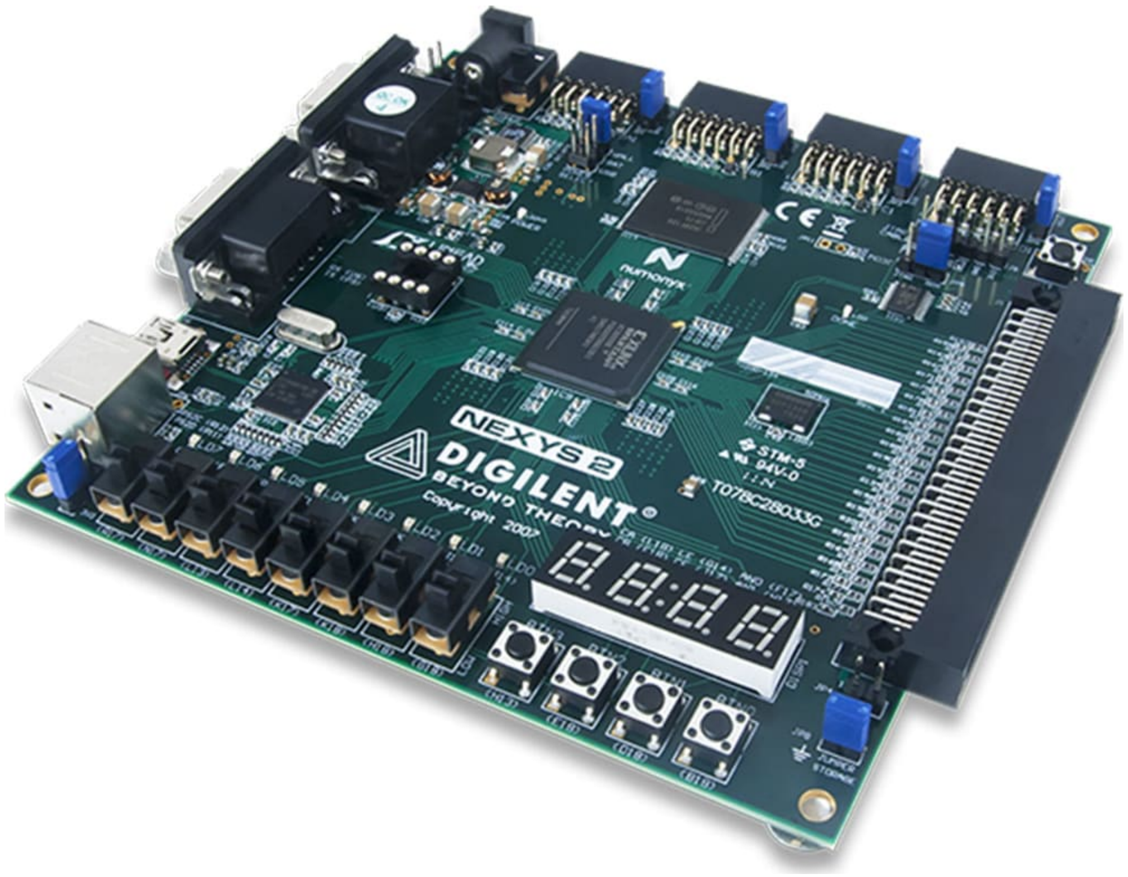
LSIC Project



Andrei LAZAROV

CTI En, Year 3

1st semester

2022-2023

Table of contents

Overview

At the LSIC laboratory meetings, we learned to program an FPGA board by writing Verilog code. The trainer board was a Digilent Nexys 2.

Project description

After uploading the generated programming file, the board will start the counter and display the seconds passed:

- in **binary** with the 8 LEDs
- in **hexadecimal** on the 4-digit seven-segment display

The user has 3 controls available:

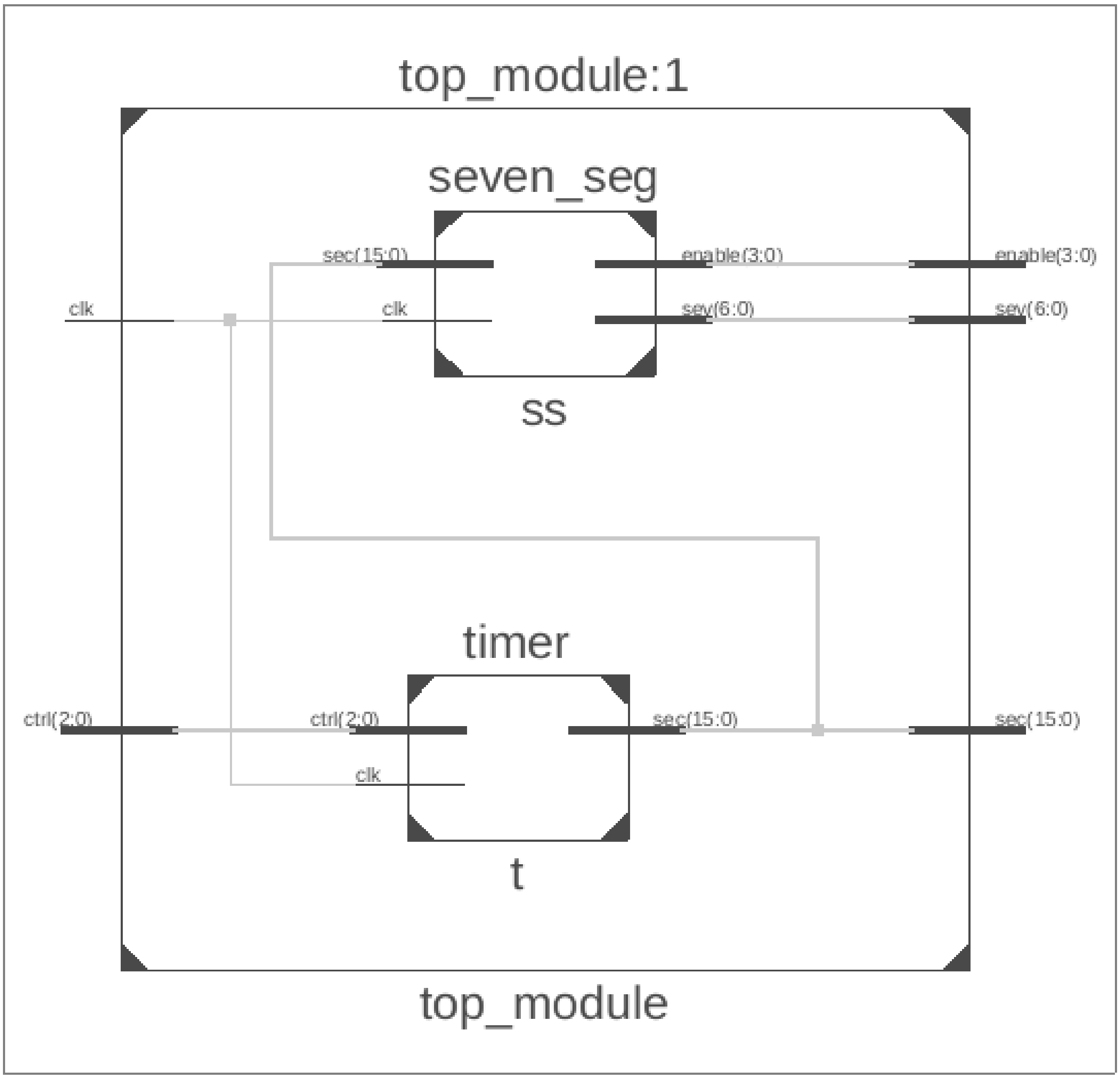
1. **Start** - first switch (R17)
2. **Stop** - second switch (N17)
3. **Pause** - third switch (L13)

To send a control signal - toggle the switch on and back off. (use them as push buttons)

Milestones

- 🟢 Blinking an LED
  Getting used to the board and the tools;
- 🟢 Time measurement
  Count clock ticks and store seconds passed based on frequency;
  Display time in binary;
- 🟢 7-segment display output
  Display time in hexadecimal;
- 🟢 States / opcodes
  Start, stop and pause the timer with buttons/switches;
- 🔴 Memories
- 🔴 Wishbone

Schematic


top_module:1
seven_seg
sec(15:0)
enable(3:0)
enable(3:0)
clk
clk
sev(6:0)
sev(6:0)
ss
timer
ctrl(2:0)
ctrl(2:0)
sec(15:0)
sec(15:0)
clk
t
top_module

Files

The code is written in 4 files:

1. **top_module.v**

   The top level module - instantiates the other two modules.

```
`timescale 1ns / 1ps

module top_module(
      input clk,
      input [2:0] ctrl,
      output [15:0] sec,
      output [3:0] enable,
      output [6:0] sev
      );

      timer t (
      .clk (clk),
      .ctrl (ctrl),
      .sec (sec)
      );

      seven_seg ss (
      .clk (clk),
      .sec (sec),
      .sev (sev),
      .enable (enable)
      );

endmodule
```

2. timer.v

Contains timer module - responsible with time measurement and states

```
`timescale 1ns / 1ps

`define ST_IDLE   3'b000      //idle
`define ST_START  3'b001      //start
`define ST_STOP   3'b010      //stop
`define ST_PAUSE  3'b100      //pause


module timer(
    input         clk,
    input   [2:0] ctrl,
    output [15:0] sec
    );

      reg  [2:0] state_reg, state_nxt;
      reg [31:0] cnt_reg, cnt_nxt;
      reg [15:0] sec_reg, sec_nxt;

      assign sec = sec_reg;

      always @ (posedge clk) begin

            cnt_reg <= cnt_nxt;
            sec_reg <= sec_nxt;
            state_reg <= state_nxt;
      end

      always @ (*) begin

            cnt_nxt = cnt_reg;
            sec_nxt = sec_reg;
            state_nxt = state_reg;

            case (state_reg)

                  `ST_IDLE: begin

                        if (ctrl[0]==1) state_nxt = `ST_START;
                        else if (ctrl[1]==1) state_nxt = `ST_STOP;
                        else if (ctrl[2]==1) state_nxt = `ST_PAUSE;
                  end
                  `ST_START: begin

                        cnt_nxt = cnt_reg + 1;

                        if (cnt_reg == 49_999_999) begin
                                sec_nxt = sec_reg + 1;
                                cnt_nxt = 0;
                        end

                        if (ctrl[0]==1) state_nxt = `ST_START;
                        else if (ctrl[1]==1) state_nxt = `ST_STOP;
                        else if (ctrl[2]==1) state_nxt = `ST_PAUSE;
                  end

                  `ST_STOP: begin

                        cnt_nxt = 0;
                        sec_nxt = 0;
                        if (ctrl[0]==1) state_nxt = `ST_START;
                        else if (ctrl[1]==1) state_nxt = `ST_STOP;
                        else if (ctrl[2]==1) state_nxt = `ST_PAUSE;
                  end

                  `ST_PAUSE: begin

                        cnt_nxt = cnt_reg;
                        if (ctrl[0]==1) state_nxt = `ST_START;
                        else if (ctrl[1]==1) state_nxt = `ST_STOP;
                        else if (ctrl[2]==1) state_nxt = `ST_PAUSE;
                  end
            endcase
      end
endmodule
```

3. seven_seg.v

Contains seven_seg module - writes the output to the seven segment display

```
module seven_seg(
      input              clk,
      input [15:0]       sec,
      output [3:0]       enable,
      output [6:0]       sev
   );

      reg [3:0] enable_reg;
      assign enable = enable_reg;

      reg [6:0] sev_reg;
      assign sev = sev_reg;

      reg [3:0] hex_digit;

      reg [1:0] select_digit_reg, select_digit_nxt;
      reg [31:0] cnt_reg, cnt_nxt;

      always @ (posedge clk) begin
            select_digit_reg <= select_digit_nxt;
            cnt_reg <= cnt_nxt;
      end

      always @ (*) begin

            if (select_digit_reg == 3 & cnt_reg == 199_999)
                  select_digit_nxt = 0;
            else if (cnt_reg == 199_999)
                  select_digit_nxt = select_digit_reg + 1;
            else
                  select_digit_nxt = select_digit_reg;

            cnt_nxt = cnt_reg + 1;

            if (cnt_reg == 199_999)
                  cnt_nxt = 0;

            case (select_digit_reg)
                  0 : begin
                        enable_reg = 4'b1110;
                        hex_digit = sec[3:0];
                  end
                  1 : begin
                        enable_reg = 4'b1101;
                        hex_digit = sec[7:4];
                  end
                  2 : begin
                        enable_reg = 4'b1011;
                        hex_digit = sec[11:8];
                  end
                  3 : begin
                        enable_reg = 4'b0111;
                        hex_digit = sec[15:12];
                  end
            endcase
      end

   always @(*)
      case (hex_digit)
          4'b0001 : sev_reg = 7'b1111001;   // 1
          4'b0010 : sev_reg = 7'b0100100;   // 2
          4'b0011 : sev_reg = 7'b0110000;   // 3
          4'b0100 : sev_reg = 7'b0011001;   // 4
          4'b0101 : sev_reg = 7'b0010010;   // 5
          4'b0110 : sev_reg = 7'b0000010;   // 6
          4'b0111 : sev_reg = 7'b1111000;   // 7
          4'b1000 : sev_reg = 7'b0000000;   // 8
          4'b1001 : sev_reg = 7'b0010000;   // 9
          4'b1010 : sev_reg = 7'b0001000;   // A
          4'b1011 : sev_reg = 7'b0000011;   // b
          4'b1100 : sev_reg = 7'b1000110;   // C
          4'b1101 : sev_reg = 7'b0100001;   // d
          4'b1110 : sev_reg = 7'b0000110;   // E
          4'b1111 : sev_reg = 7'b0001110;   // F
          default : sev_reg = 7'b1000000;   // 0
      endcase

endmodule
```

4. fdsf.ucf

User constraint file - maps inputs and outputs to physical pins.

- The 8 LEDs display the seconds passed in binary (LED on = 1, off = 0) from the *sec* output.
- *clk* gets the real clock signal
- *sev* is mapped to the seven segments of a digit
- *enable* selects the digit
- *ctrl* gets the input switches states

```
NET "sec[7]" LOC = "R4";
NET "sec[6]" LOC = "F4";
NET "sec[5]" LOC = "P15";
NET "sec[4]" LOC = "E17";
NET "sec[3]" LOC = "K14";
NET "sec[2]" LOC = "K15";
NET "sec[1]" LOC = "J15";
NET "sec[0]" LOC = "J14";

NET "clk"    LOC = "B8";

NET "sev[0]" LOC = "L18";     //a
NET "sev[1]" LOC = "F18";     //b
NET "sev[2]" LOC = "D17";     //c
NET "sev[3]" LOC = "D16";     //d
NET "sev[4]" LOC = "G14";     //e
NET "sev[5]" LOC = "J17";     //f
NET "sev[6]" LOC = "H14";     //g

NET "enable[0]" LOC = "F17";  //AN0
NET "enable[1]" LOC = "H17";  //AN1
NET "enable[2]" LOC = "C18";  //AN2
NET "enable[3]" LOC = "F15";  //AN3

NET "ctrl[0]" LOC = "R17"; //START
NET "ctrl[1]" LOC = "N17"; //STOP
NET "ctrl[2]" LOC = "L13"; //PAUSE
```