Aeronautics Institute of Technology – ITA
Computer Science Division – IEC

EEA-25 Programmable Digital Systems
Prof. André F. Ponchet
Teaching assistant: Felipe Ferreira

Laboratory 04
Last document update: September 4, 2023

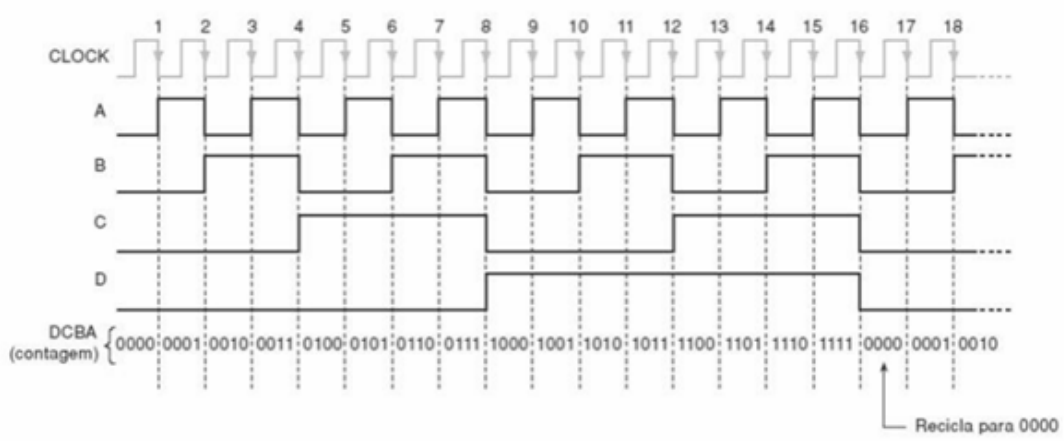
1 Objectives

1.1 General Objectives

- Familiarize yourself with the Lattice iCE40 development environment;
- Understand circuit design process using HDLs and FPGA implementation;
- Simulate and verify results with real FPGA boards.

1.2 Specific Objectives

- Implement a frequency divider (something like the image below);
- Understand behavioral/structural design approaches using Verilog;
- Use a 7-segment display to show proper working.



Frequency divider.

2 Lab Development

2.1 Circuit analysis

In this lab, we are going to implement a frequency divider using two counters. This circuit receives a clock or similar type of input and emits said signal divided by a certain value as output.

2.2 HDL coding and simulation

1. Create a folder named "LAB-4";
2. Implement the following code in Verilog, named "frequency_divider.v";

```
module frequency_divider (

     // Inputs
     input clk,
     input rst_btn,

     // Outputs
     output reg[3:0] out_clk
);
     wire rst;
     reg [31:0] count;
     localparam [31:0] max_count = (1) - 1;

      // Reset is the inverse of the reset button
     assign rst = ~rst_btn;

     // Clock divider
     always @(posedge clk or posedge rst) begin
          if (rst) begin
               out_clk <= 4'b0;
               count <= 32'b0;
          end
          else if (count == max_count) begin
                    count <= 32'b0;
                    out_clk <= out_clk + 1'b1;
               end
               else begin
                         count <= count + 1;
               end
     end

endmodule
```

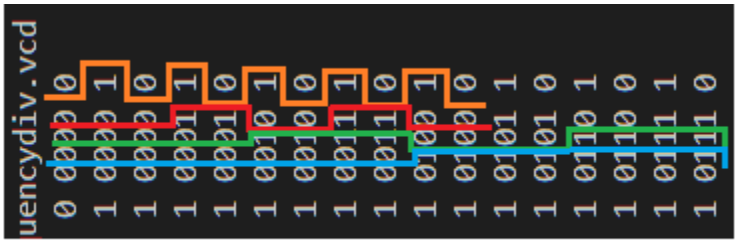
3. Create the following testbench code "frequency_divider_tb.v" inside the folder:

```
`timescale 10ns/1ps
`include "frequency_divider.v"

module frequency_divider_tb;

     output reg clk, rst;
     input wire[3:0] out_clk;

     frequency_divider uut(clk, rst, out_clk);

     initial
          clk = 1'b0;

     always
          #1 clk = ~clk;

     initial begin
          $monitor($time,"clk = %b, rst = %b, out_clk = %b",
                                          clk, rst, out_clk);

          rst = 0;
          #1 rst = 1;
          #31 $finish;
     end

     initial begin
```

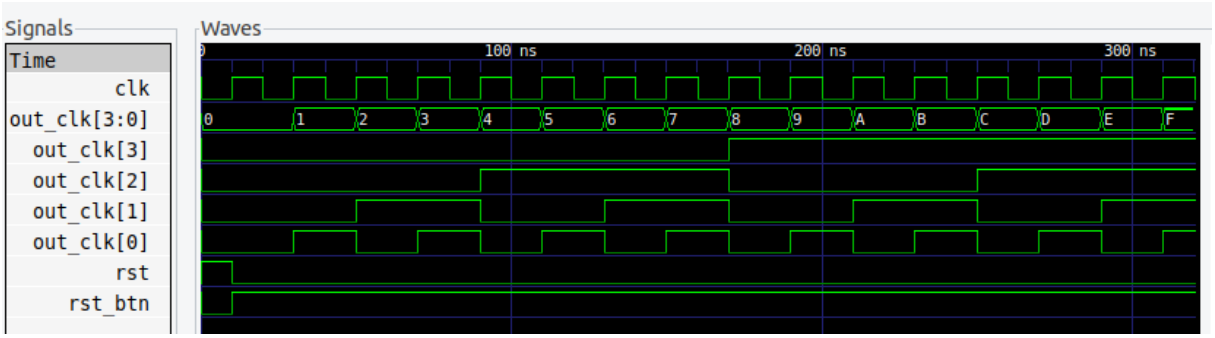
```
            $dumpfile ("frequency_divider_tb.vcd");
            $dumpvars (0, frequency_divider_tb);
        end

endmodule
```

4. In the same folder, enter the following commands using Linux terminal. Open "frequency_divider_tb.vcd" file with gtkwave and compare your results with the next image:

```
$ iverilog frequency_divider_tb.v
$ vvp a.out
```



Frequency divider vvp.



Frequency divider simulation.

- **For more:** Verilog provides some system functions and tasks specifically for us to generate input and output to help with verification. **Monitor** is an example of a system task. System tasks are not used, they are ignored by the synthesis tools.
- The monitor instruction is not as powerful as the waveform graphical tools, but it is useful in many cases and provides little idea of the circuit's behavior.
- **For more:** The always keyword takes two arguments (known as sensitivity list). Looking at the Always block, we are activating it on each rising edge of clock or reset signals.

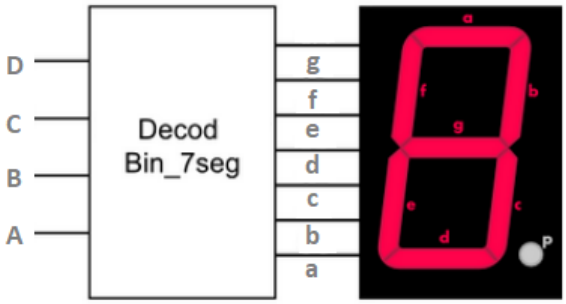
3 Exercises

1. What value should the **max_count** parameter be initialized to convert the 12 MHz clk signal to a 1 Hz signal in **out_clk**?
2. Implement using Verilog and behavioral design approach, a circuit that converts an input BCD number to the representation on a common anode 7-segment display (following image).
   - The code snippet below is incomplete, based on the truth table, try to get the correct description that represents the decoder for the display.

```
        module decod_bcd (
        input [3:0],
        output [6:0]
```

```
        );
        always @* begin
        case()

        endcase
        end
        endmodule
```

- Try implementing testbench to validate your circuit.

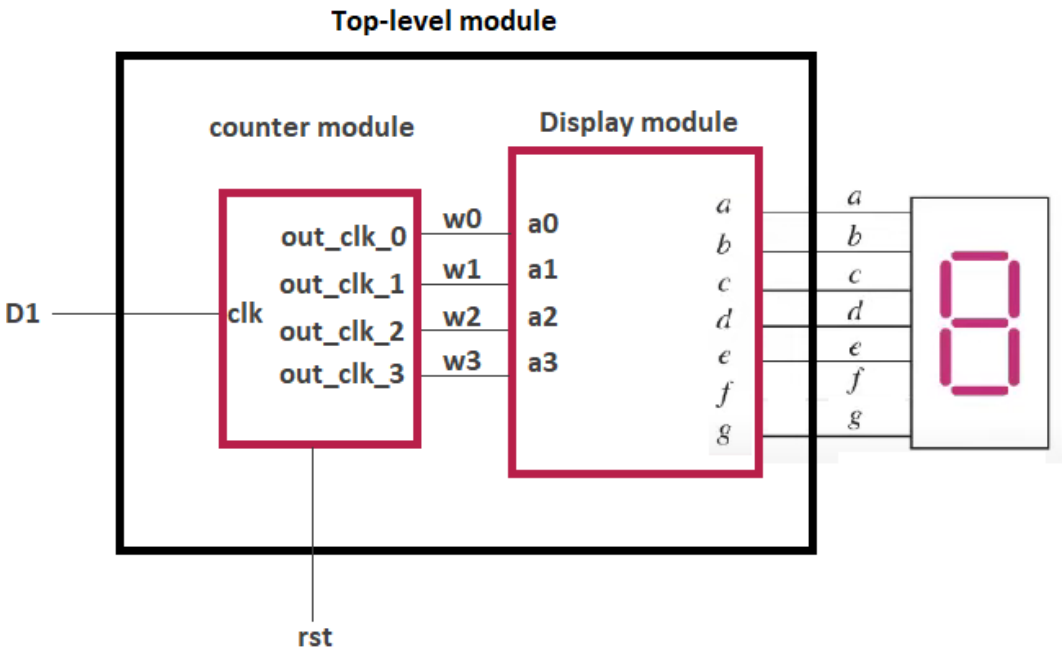


Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

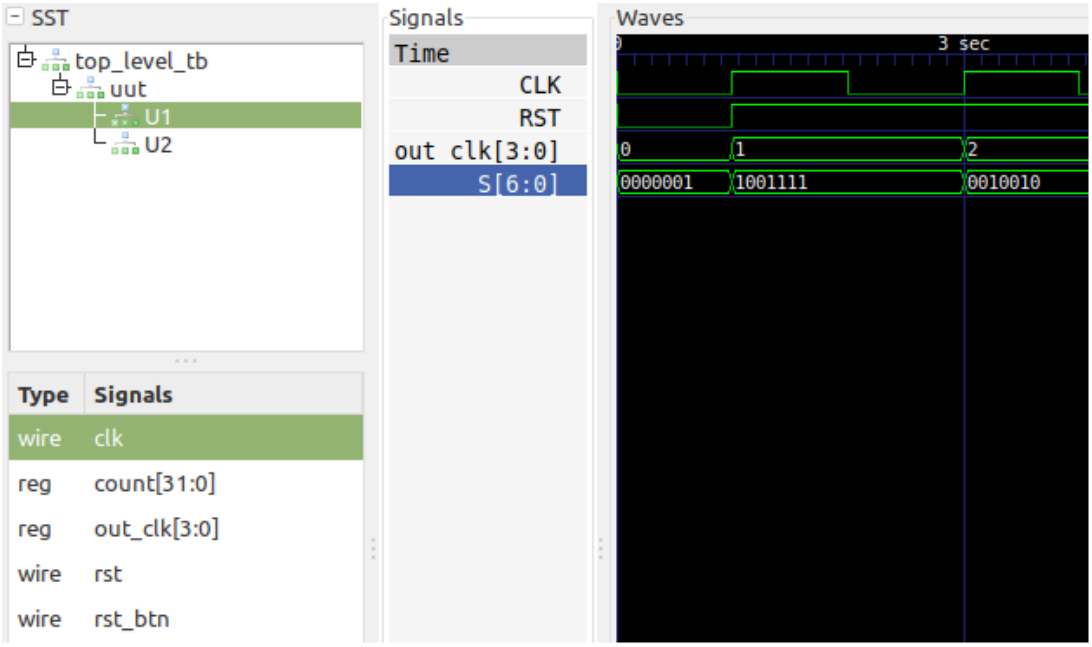
Decoder for 7-segment display.

- One of the other possible designs is the hierarchical or mixed type. This type of structure is interesting because we reuse and organize small modules of different designs into top levels.
- Based on the image below, try to create a top level that reuses the counter and display drive that were developed in this lab. Propose a testbench to validate your circuit.

- You should get a simulation similar to the image ahead.



Top level design structure.



Simulation top level design.

3.1 Board implementation

1. Create the pin constraint file "top_level.pcf" inside the folder:

```
set_io s[6] E2
set_io s[5] B1
set_io s[4] C5
set_io s[3] C6
set_io s[2] E3
set_io s[1] C2
set_io s[0] B4
set_io clk_d1 D1
set_io -pullup yes rst A1
set_io -nowarn PMOD1 D1
set_io -nowarn PMOD1 A1
```

```
set_io -nowarn PMOD2 E2
set_io -nowarn PMOD1 B1
set_io -nowarn PMOD2 C5
set_io -nowarn PMOD2 C6
set_io -nowarn PMOD1 E3
set_io -nowarn PMOD2 C2
set_io -nowarn PMOD2 B4
```

Note that the clock signal is assigned to the pin D1, it takes advantage on the board oscillator connected directly to D1 whose frequency is 12MHz.

2. Use the makefile you created previously to perform: synthesis, PnR, package, and write to the iCESugar-nano.
3. After simulation/debugging, use the available breadboard to show the actual circuit working.

4 Grading

This lab does not require a report. When finishing it, make sure you enter the line for a direct assessment conducted by the professors or TAs during lab timeframe.