**Aeronautics Institute of Technology – ITA**
Computer Science Division – IEC

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

**Laboratory 03**
Last document update: August 21, 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 digital comparator;
- Exercise the design process seen on the last laboratory;
- Describe a circuit through continuous expressions in Verilog;
- Build a 4-bit comparator and complete the challenge.

## 2 Lab Development

### 2.1 Circuit analysis

A digital comparator is a combinational circuit that compares two binary or digital values and determines which one is greater/lesser than the other or if they are equal. The circuit has two inputs: A and B, and three outputs, accordingly to each conditions: A $<$ B, A $>$ B and A $=$ B.

### 2.2 HDL coding and simulation

1. Create a folder named "LAB-3";
2. Implement the following code in Verilog, named "magnitude_comparator.v":

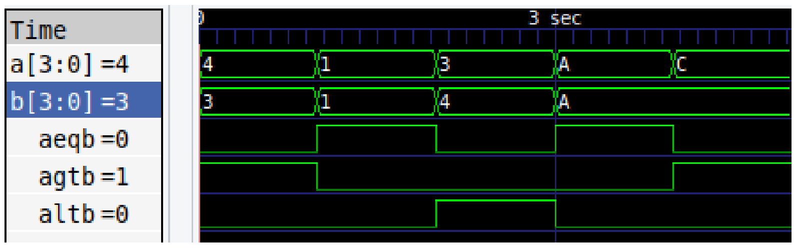
```verilog
module magnitude_comparator (
      // Inputs
      input [3:0] a, b,
      // Outputs
      output aeqb, agtb, altb
);
      assign aeqb = (a == b) ? 1'b1 : 1'b0;
      assign agtb = (a > b)  ? 1'b1 : 1'b0;
      assign altb = (a < b)  ? 1'b1 : 1'b0;
endmodule
```

3. Create the following testbench code "magnitude_comparator_tb.v" inside the folder, remember to map each port respecting the Unit Under Test (UUT) order:

```
‘include "magnitude_comparator.v"

module magnitude_comparator_tb;

    // Inputs
    reg [3:0] data_a;
    reg [3:0] data_b;

    // Outputs
    wire aeqb, agtb, altb;

   // Instantiate the Unit Under Test (UUT)
    magnitude_comparator uut (data_a, data_b, aeqb, agtb, altb);

    initial begin
        $dumpfile ("magnitude_comparator_tb.vcd");
        $dumpvars (0, magnitude_comparator_tb);
    end

    initial
        $monitor ($time," data_a = %d, data_b = %d,
                                 aeqb = %b, agtb = %b, altb = %b",
                                 data_a, data_b, aeqb, agtb, altb);

    initial begin
        data_a = 4; data_b = 3;  #1
        data_a = 1; data_b = 1;  #1
        data_a = 3; data_b = 4;  #1
        data_a = 10; data_b = 10; #1
        data_a = 12; data_b = 10; #1;
        $finish;
    end
endmodule
```

4. Enter the following commands using Linux terminal and open the file "magnitude_comparator_tb.vcd" with **gtkwave**. Compare the results with the following image.

```
iverilog magnitude_comparator_tb.v
vvp a.out
gtkwave magnitude_comparator_tb.vcd
```

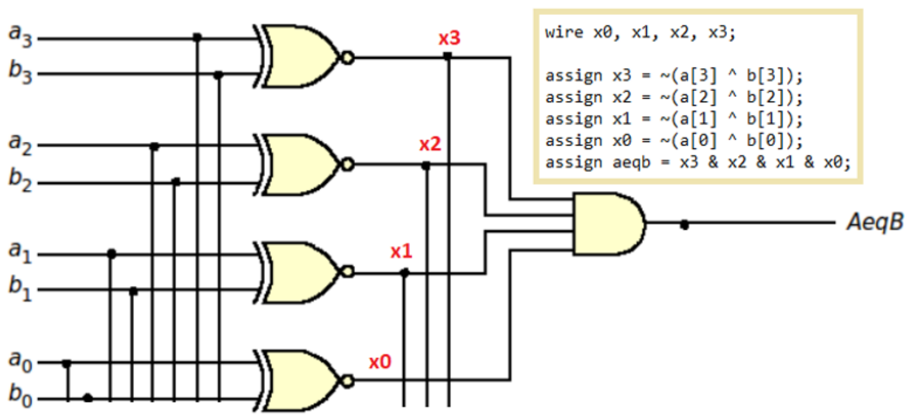


Comparator simulation.

## 3 Challenge

- Now that you have been induced to validate the circuit's behavior using ternary operators, we will explore other possibilities to describe it.
- Exploring the concepts seen in digital circuits, we are able to get the logic expression that represents a digital circuit or get a digital circuit from a logic expression, right?
- It is essential to interpret the logic expressions of a circuit to synthesize it or obtain a circuit through its HDL equivalent.

- Your challenge will be to be able to put these concepts into practice!
- The image below represents the circuit responsible for the function A == B.



```
wire x0, x1, x2, x3;

assign x3 = ~(a[3] ^ b[3]);
assign x2 = ~(a[2] ^ b[2]);
assign x1 = ~(a[1] ^ b[1]);
assign x0 = ~(a[0] ^ b[0]);
assign aeqb = x3 & x2 & x1 & x0;
```

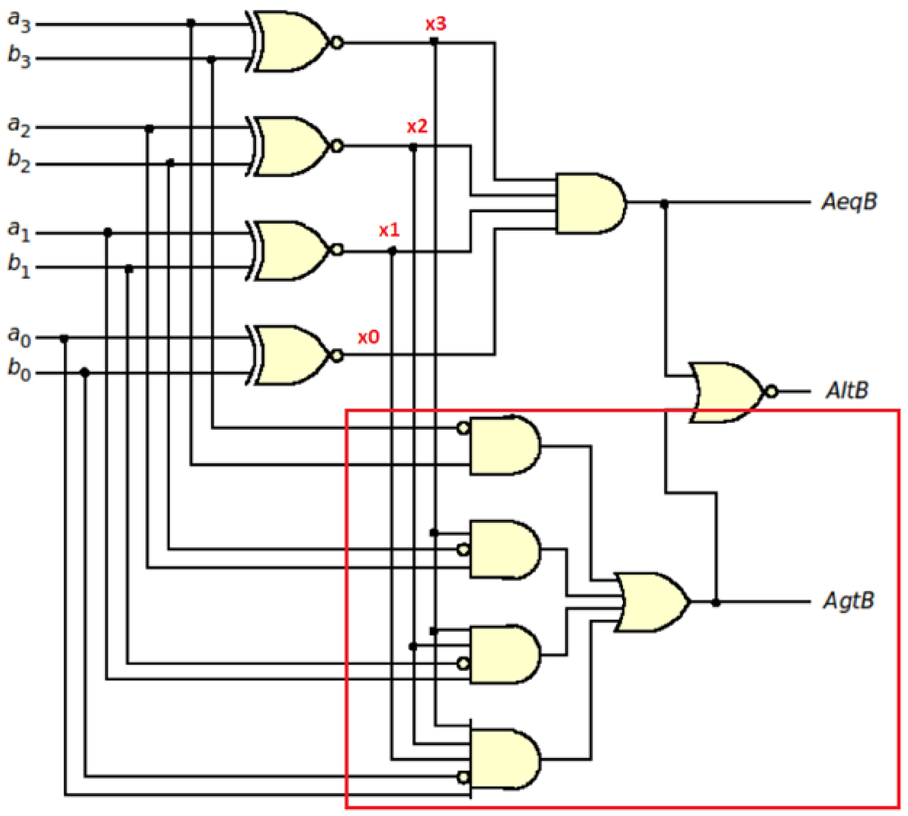
Incomplete 4-bit comparator.

```
module magnitude_comparator (

    // Inputs
    input [3:0] a, b,

    // Outputs
    output aeqb, agtb, altb
);
    wire x0, x1, x2, x3;

    assign x3 = ~(a[3] ^ b[3]);
    assign x2 = ~(a[2] ^ b[2]);
    assign x1 = ~(a[1] ^ b[1]);
    assign x0 = ~(a[0] ^ b[0]);

    assign aeqb = x3 & x2 & x1 & x0;

    assign agtb = (a > b) ? 1'b1 : 1'b0;
    assign altb = (a < b) ? 1'b1 : 1'b0;
endmodule
```

- Notice that we modified our Verilog file to match the equivalent logic function of the circuit.
- **Your challenge in this lab will be to continue the remaining HDL to get the parts responsible for A $>$ B and A $<$ B, based on the example above.**

- **Get the equivalent expression that matches the AgtB output, i.e, A $>$ B.**

```
module magnitude_comparator (

    // Inputs
    input [3:0] a, b,

    // Outputs
    output aeqb, agtb, altb
);
    wire x0, x1, x2, x3;

    assign x3 = ~(a[3] ^ b[3]);
```



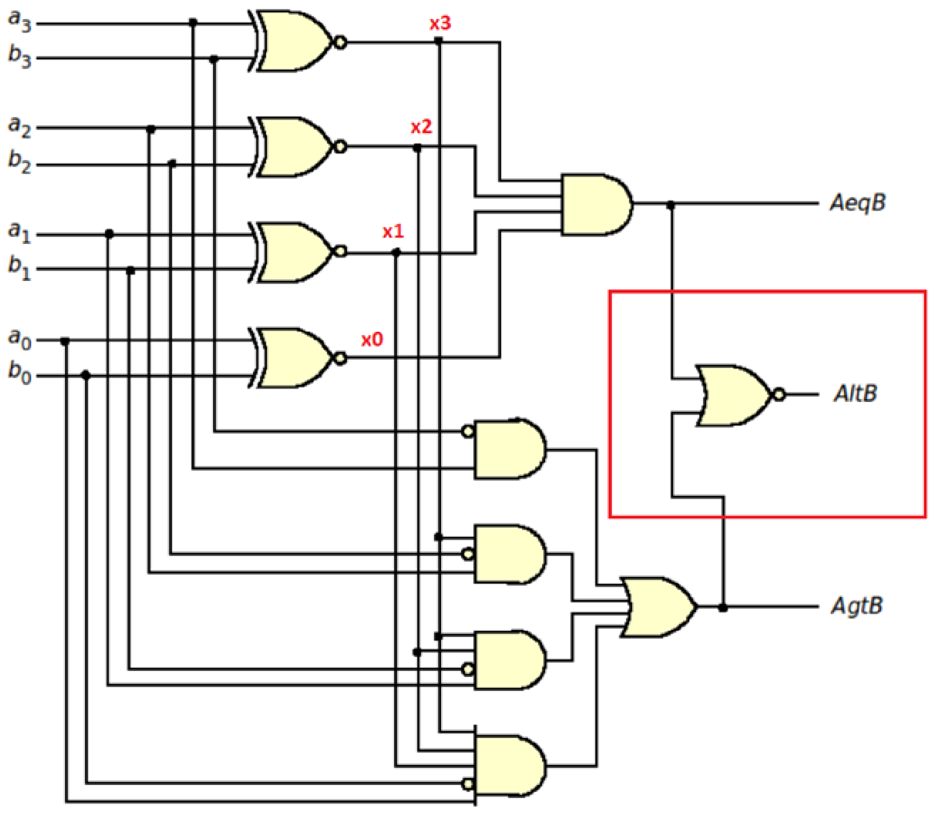
Challenge comparator schematic A > B.

```
    assign x2 = ~(a[2] ^ b[2]);
    assign x1 = ~(a[1] ^ b[1]);
    assign x0 = ~(a[0] ^ b[0]);

    assign aeqb = x3 & x2 & x1 & x0;

    assign agtb = a[3]&(~b[3]) | (...)
    (...)
endmodule
```

- The snippet above corresponds to the beginning of the expression you should get, try to complete it.
- **Get the equivalent expression that matches the AltB output, i.e, A < B.**
- **Note that you can use the previous expressions as a resource.**

## 4 Exercises

- What differences did you observe between using the data flow approach (functional or logical) and the structural approach?
- Implement the missing functions with the proposed challenge and simulate the circuit using the testbench already provided.
- Propose the description that represents the 4x1 multiplexer circuit below and create a testbench to validate your circuit.
- Desirable: Use data flow approach (logical description).

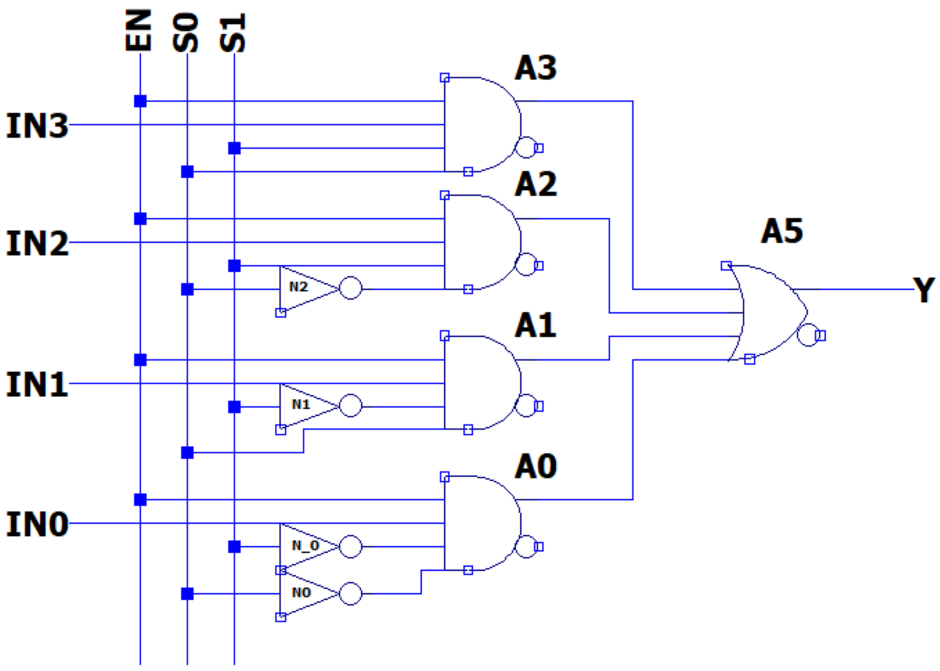


Challenge comparator schematic A < B.

```verilog
`include "mux_4_1_logic.v"
`timescale 1ns/1ps // time unit / simulation precision

module mux_4_1_tb;

    reg [3:0] IN;
    reg [1:0] S;
    reg EN;
    wire Y;

    mux_4_1_logic uut (EN, IN, S, Y);

    initial begin
        $dumpfile("mux_4_1_tb.vcd");
        $dumpvars(0, mux_4_1_tb);
    end

    initial begin

    end

endmodule
```

## 5 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.



4x1 multiplexer.