Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

LOGIC DESGIN WITH HDL
Practical session - Semester 222

WEEK 1

Introduction &
Structural Model

1 Introduction

1.1 Aims

- Get familiar with Vivado software and the FPGA development flow.
- Get familiar with FPGA Arty-Z7 board.
- Practice in designing simple digital logic circuits with Verilog.
- Understand the hierarchical design principle.
- Practice in writing test benches for a designed module.

1.2 Preparation

- Read the laboratory materials before class.
- Revise chapter 0-3 about Verilog basic.
- Each group prepare at least one laptop with Vivado software installed.

1.3 Documents and lab materials

- M. Morris Mano, Michael D. Ciletti, Digital System with an Introduction to the Verilog HDL, VHDL, and SystemVerilog, Pearson Education, Inc, 2017
- Lecture slides
- *Arty-Z7-20-Master.xdc*: Arty-Z7 constraint file.
- *Guide_for_Installing_Vivado.pdf* : Guide for installing Vivado and getting started with Vivado andArty-Z7.
- *dec1to2.v*, *mux2to1.v*: 2-to-1 multiplexer module and its sub-module.
- *mux2to2_tb.v*: test bench to simulate the module mux2to1.

1.4 Procedure

For each exercise (also for further labs):

- Read the requirements, then determine the input/output signals of your circuits.
- Make design idea of the circuit then using Verilog to model the circuit.
- Analysis & Synthesis the circuit with Vivado software.
- Write test bench to simulate the circuit on Vivado Simulator.
- Generate the bitstream and program the Arty-Z7 to evaluate the circuit.

1.5 Report requirements

- Lab exercises will be reviewed directly in class.
- Write report (with circuit/simulation screenshots inserted) in pdf.
- Must have group ID, group member's names and student IDs in the report.
- Compress the report with code files (only .v files) in only one .zip file, name the .zip the group ID (for example: Group1.zip).
- Submit on BK-elearning by deadline.

2 Exercises

2.1 Exercise 1

a. Design a *1-to-2 decoder* using structure model as the following circuit:

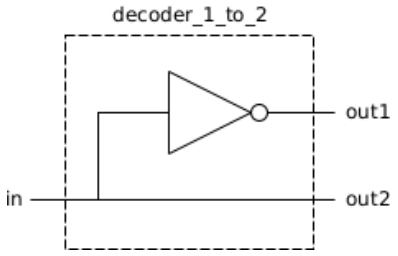


Figure 1: 1-to-2 decoder

- Dựa vào mạch ta thấy có một đầu vào (IN) và hai đầu ra (OUT 1, OUT2).
- Bảng chân trị

IN	OUT 1	OUT2
0	1	0
1	0	1

b. Design a *2-to-1 multiplexer* using structure model and hierarchical design (reuse the module decoder 1 to 2) as following circuit:

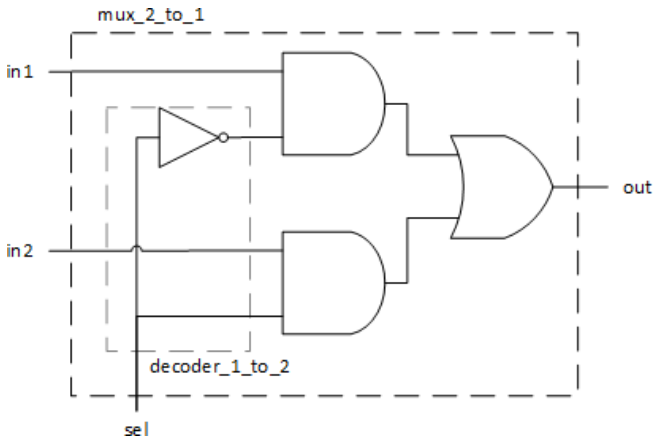


Figure 2: 2-to-1 multiplexer

Check the schematic in RTL Analysis to see the result circuit.

```
module mux_2_to_1(in1, in2, sel, out);
output out;
input in1, in2, sel;
wire T1, T2, Sbar;
and (T1, in2, sel), (T2, in1, Sbar);
not (Sbar, sel);
or (out, T1, T2);
endmodule
```

2.2 Exercise 2

a. Write a test bench for the *2-to-1 Multiplexer* in *Exercise 1* then use Vivado Simulator to simulate the design, students can use the given example source code. Let's analyse the structure of a test bench then point out the differences between an RTL code and a test bench code.

Change the **Radix**, **Format** of signals and use zoom tool to evaluate the waveform.

Check the **Tcl console** window to see output of **$monitor** command.

b. Then, perform the Synthesis, compare the Synthesis's Schematic and the RTL Analysis's schematic.

c. After that, run the Implementation, check the Utilization report in Project Summary for used resources.

d. Add the Arty-Z7 constraint file to the project, assign pin for the design as follow:

- in1: btn[0], in2: btn[1]
- sel: sw[0]
- out: led[0]

then, generate bitstream file and program the FPGA to test the implemented circuit on board.

```
11  module mux_2_to_1_tb;
12          wire out;
13          reg in1, in2, sel;
14
15          mux_2_to_1 uut (.in1(in1), .in2(in2), .sel(sel), .out(out));
16          initial begin
17                  in1=1'b0;
18                  in2=1'b0;
19                  sel=1'b0;
20                  #100 $finish;
21          end
22          always #40 in1=~in1;
23          always #20 in2=~in2;
24          always #10 sel=~sel;
25          always@(in1 or in2 or sel)
26          $monitor("At time = %t, Output = %d", $time, out);
27  endmodule;
```

```
At time =                    0, Output = 0
At time =                   10, Output = 0
At time =                   20, Output = 0
At time =                   30, Output = 1
At time =                   40, Output = 1
At time =                   50, Output = 0
At time =                   60, Output = 1
At time =                   70, Output = 1
At time =                   80, Output = 0
At time =                   90, Output = 0
```

2.3 Exercise 3

a. Design a half-adder circuit using structural model.

a. Half-adder

```
1  module half_adder(
2      input a,
3      input b,
4      output s,
5      output c
6      );
7      and(c,a,b);
8      xor(s,a,b);
9  endmodule
```

testbench:

```
1  module half_adder_tb;
2  //inputs
3          reg a;
4          reg b;
5  //outputs
6          wire s;
7          wire c;
8
9          half_adder uut(.a(a),.b(b),.s(s),.c(c));
10         initial begin
11         // Initialize Inputs
12                 a = 1'b0;
13                 b = 1'b0;
14
15                 #2 a = 1'b0; b = 1'b1;
16                 #2 a = 1'b1; b = 1'b0;
17                 #2 a = 1'b1; b = 1'b1;
18         end
19         initial $monitor("$time=%g,s=%b,c=%b,a=%b,b=%b",$time,s,c,a,b);
20         initial #10 $finish;
21 endmodule
22
```

```
$time=0,s=0,c=0,a=0,b=0
$time=2,s=1,c=0,a=0,b=1
$time=4,s=1,c=0,a=1,b=0
$time=6,s=0,c=1,a=1,b=1
```

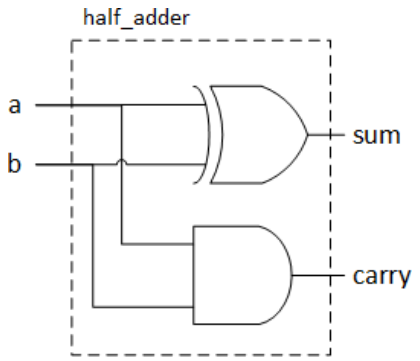


Figure 3: Half adder

b. Design a full-adder circuit using structural model. Reuse the half-adder module.

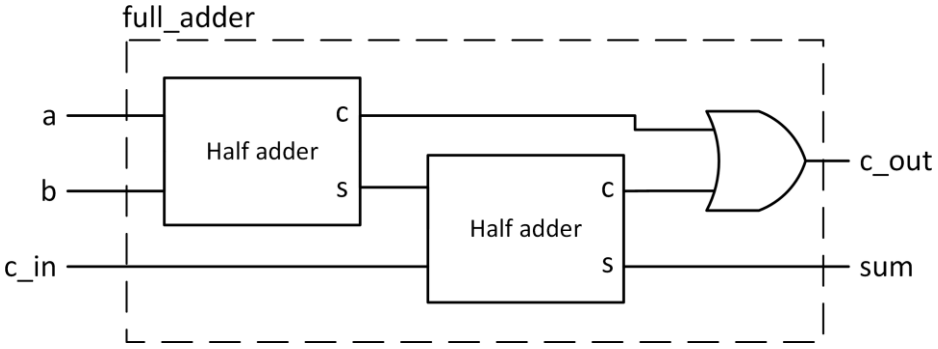


Figure 4: Full adder

Write a test bench to evaluate operations of the implemented full-adder circuit.

Test the implemented circuit on Arty-Z7 board using switches/buttons and LEDs.

Bonus: show the operands and sum on 7-seg LEDs.

b.Full adder

```
1   module full_adder(a,b,cin,sum,cout);
2     input a,b,cin;
3     output sum,cout;
4     wire x,y,z;
5
6   // instantiate building blocks of full adder
7     half_adder h1(a,b,x,y);
8     half_adder h2(x,cin,sum,z);
9     or o1(cout,y,z);
10  endmodule
```

testbench

```
module full_adder_tb;
  reg a,b,cin;
  wire sum,cout;

  full_adder uut(.a(a),.b(b),.cin(cin),.sum(sum),.cout(cout));

// insert all the inputs
  initial begin a=1'b1;  #4; a=1'b0;#10 $stop();end
  initial begin b=1'b1; forever #2 b=~b;end
  initial begin cin=1'b1;forever #1 cin=~cin; #10 $stop();end

// monitor all the input and output ports at times
// when any of the input changes its state

 initial begin $monitor(" time=%0d a=%b b=%b cin=%b sum=%b cout=%b",$time,a,b,cin,sum,cout);end
 endmodule
```

```
time=0 a=1 b=1 cin=1 sum=1 cout=1
 time=1 a=1 b=1 cin=0 sum=0 cout=1
 time=2 a=1 b=0 cin=1 sum=0 cout=1
 time=3 a=1 b=0 cin=0 sum=1 cout=0
 time=4 a=0 b=1 cin=1 sum=0 cout=1
time=5 a=0 b=1 cin=0 sum=1 cout=0
 time=6 a=0 b=0 cin=1 sum=1 cout=0
 time=7 a=0 b=0 cin=0 sum=0 cout=0
 time=8 a=0 b=1 cin=1 sum=0 cout=1
 time=9 a=0 b=1 cin=0 sum=1 cout=0
 time=10 a=0 b=0 cin=1 sum=1 cout=0
 time=11 a=0 b=0 cin=0 sum=0 cout=0
 time=12 a=0 b=1 cin=1 sum=0 cout=1
 time=13 a=0 b=1 cin=0 sum=1 cout=0
```

c. Design a 4-bit ripple carry adder using structural model. Reuse the implemented full-adder. Write a test bench to simulate the implemented circuit.

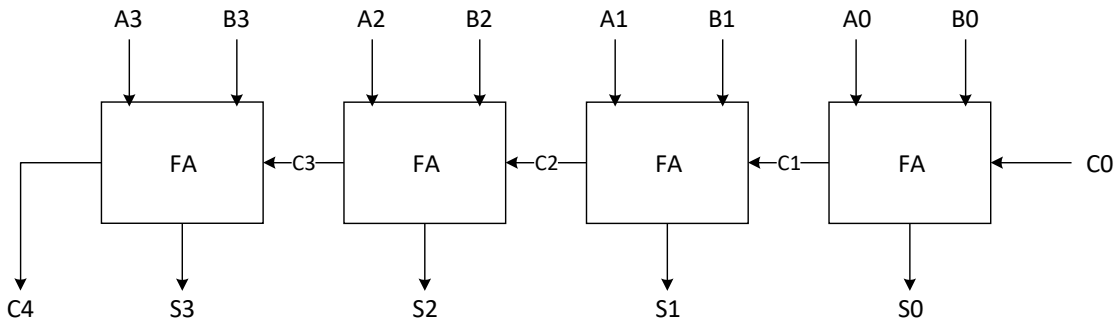


Figure 5: 4-bit ripple carry adder

c. 4-bit ripple carry adder

```
module ripple_carry_adder(x,y,cin,s,cout);
        input [3:0] x,y;
        input cin;
        output [3:0] s;
        output cout;
        wire w1,w2,w3;
        full_adder f1(x[0], y[0], cin, s[0], w1);
        full_adder f2(x[1], y[1], w1, s[1], w2);
        full_adder f3(x[2], y[2], w2, s[2], w3);
        full_adder f4(x[3], y[3], w3, s[3], cout);
endmodule
```

Testbench

```
module ripple_carry_adder_tb;
        reg [3:0] x,y;
        reg cin;
        wire [3:0] s;
        wire cout;

        ripple_carry_adder uut(.x(x),.y(y),.cin(cin),.s(s),.cout(cout));
        initial begin
                x = 0;
                y = 0;
                cin = 0;
                // Wait 100 ns for global reset to finish
                #100;
                // Add stimulus here
                x=4'b0001; y=4'b0000; cin=1'b0;
                #10 x=4'b1010; y=4'b0011; cin=1'b0;
                #10 x=4'b1101; y=4'b1010; cin=1'b1;
        end
        initial begin
                $monitor("time=%g, x=%b y=%b cin=%b : s=%b cout=%b",$time,x,y,cin,s,cout);
        end
endmodule
```

```
time=0, x=0000 y=0000 cin=0 : s=0000 cout=0
```

2.4 Exercise 4

Give the 2-bit comparator circuit as Figure 6 with $A = \{A1, A0\}$ and $B = \{B1, B0\}$ are 2 2-bit input numbers, A_gt_B is active if $A > B$, A_lt_B is active if $A < B$ and A_eq_B is active if $A = B$.

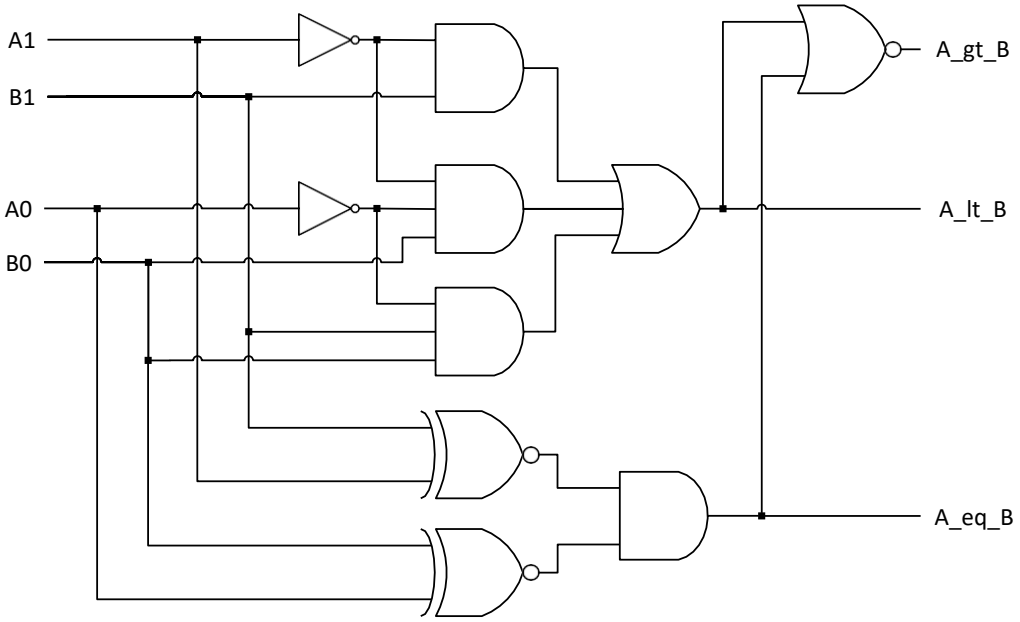


Figure 6: 2-bit comparator

Let's design a 4-bit comparator following below steps:

- Analyse the functions of each output of the 2-bit comparator, then determine the functions of 4-bit comparator outputs.

  - Mạch so sánh 2 bit được hiển thị trong hình 6 so sánh hai số đầu vào 2 bit, là A và B. Các đầu ra của bộ so sánh như sau:

  + A_eq_B: Đầu ra này hoạt động khi và chỉ khi A=B.

  + A_qt_B: Đầu ra này hoạt động khi và chỉ khi A>B.

  + A_lt_B: Đầu ra này hoạt động khi và chỉ khi A<B.

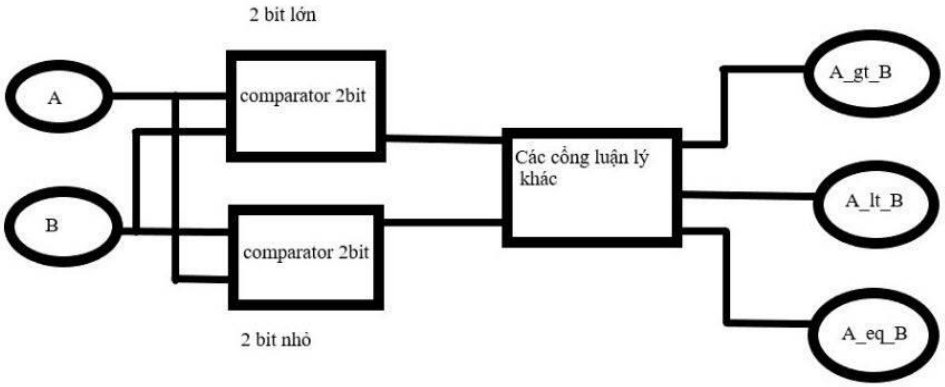
  - Các chức năng của đầu ra so sánh 4 bit có thể được xác định bằng cách mở rộng cùng một logic được sử dụng trong bộ so sánh 2 bit. Chúng ta có thể xem xét 4 bit đầu vào của A và B một cách riêng biệt, cụ thể như sau:

  + A_eq_B: Đầu ra này hoạt động khi và chỉ khi A=B. Trong một bộ sao sánh 4 bit, điều này có nghĩa là cả 4 bit đầu vào của A và B phải bằng nhau.

  + A_gt_B: Đầu ra này hoạt động khi và chỉ khi A>B. Trong một bộ so sánh 4 bit, điều này có nghĩa là A lớn hơn B theo bit quan trọng nhất (MSB) hoặc MSB bằng nhau và các bit quan trọng nhất

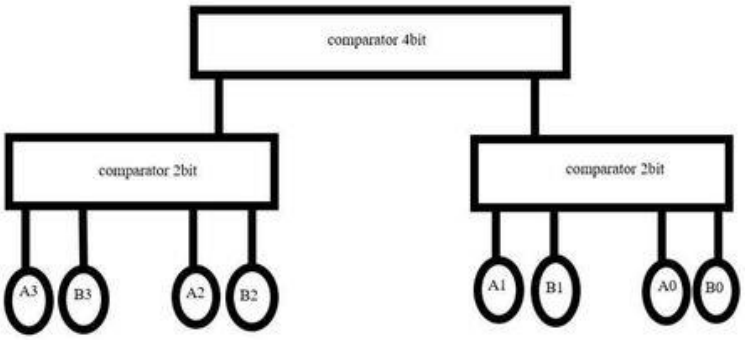
tiếp theo của A lớn hơn các bit tương ứng của B.

- Conceptualize the design of 4-bit comparator from 2-bit comparators (the 2-bit comparator can be partitioned into smaller blocks). Draw a block diagram that describes the idea.
  - Ý tưởng của bài toán so sánh này 4 bit là chúng ta sẽ chia thành 2 bit lớn và 2 bit nhỏ rồi đem chúng so sánh, rồi ta sử dụng các cổng luận lý gồm cổng and, cổng or để tổng hợp kết quả từ hai bộ sao sánh 2 bit, ta có thể tận dụng mạch so sánh 2 bit có sẵn trong bài toán này.
  - Sơ đồ khối cho thiết kế này có thể được hiển thị như sau:



  - Biểu đồ khối này bao gồm 2 bộ phận so sánh 2 bit. So sánh 2 bit lớn nhất của A và B và sao sánh 2 bit nhỏ nhất của A và B.
  - Các đầu ra của hai bộ so sánh 2 bit sau đó được tổng hợp lại bằng các cổng luận lý gồm cổng and, cổng or và cho ra các kết quả.

- Draw a diagram to shows the designed circuit hierarchy.
  - Dưới đây là một sơ đồ phân cách mạch ví dụ cho thấy thiết kế của bộ so sánh 4 bit sử dụng các bộ so sánh 2 bit làm khối xây dựng. Biểu đồ này cho thấy cấu trúc của bộ so sánh ở mức cao và không bao gồm các chi tiết về mạch bên trong của các bộ so sánh 2 bit.



  - Trong sơ đồ này, bộ so sánh 4 bit được hiển thị ở cấp cao nhất, với 2 bộ so sánh 2 bit. Các bộ so sánh 2 bit này được sử dụng để só sánh 4 cặp bit tương ứng trong các số đầu vào 4 bit.

- Các bộ so sánh 2 bit là các khối xây dựng của thiết kế này và được tổ chức theo thứ bậc trong bộ so sánh. Các đầu vào cho mỗi bộ so sánh 2 bit là 2 cặp bit tương ứng trong các số đầu vào. Các đầu ra của các bộ so sánh 2 bit được kết hợp để tạo ra đầu ra cuối cùng của bộ sao sánh 4 bit.
- Các cấu trức phân cấp này giúp quản lý sự phức tạp của thiết kế bộ so sánh, giúp việc thực hiện và gỡ lỗi dễ dàng hơn.

- Implement the designed circuit using Verilog HDL structural model.

```
module comparator_2bit(
input [1:0] A,
input [1:0] B,
output A _gt_B,
output A _lt_B,
output A_eq_B
);
wire va1, va2, va3, xnol, xno2;
and (va1, ~A[1], B[1]),
(va2, ~A[1), ~A[0], B[0]),
(va3, ~A[0], B[1], B[0]),
(A_eq_B, xno1, xno2);
xnor (xno1, A[1], B[1]),
      (xno2, A [0], B[0]);
Or (A_lt_B, va1, va2, va3);
nor (A_gt_B, A_lt_B, A _eq_B);
endmodule
module comparator_4bit(
input [3:0] A,
input [3:0] B,
output A_gt_B,
output A_lt_B,
output A_eq_B
);
wire gt1, gt2, lt1, lt2, eq1, eq2, va1, va2;
```

- Write a test bench to simulate the implemented circuit.

```
'timescale 1ns / 1ps
module comparator_4bit_tb();
reg [3:0] A;
reg [3:0] B;
```

```
wire A_gt_B;
wire A_lt_B;
wire A_eg_B;
comparator_4bit comp4bit(A, B, A_gt_B, A_lt_B, A_eg_B);
initial $monitor("time %t: A=%d, B= %d, A _gt_B=%b, A_lt_B=%b, A_eg_B=%b\n", $time, A, B,
A_gt_B, A_lt_B, A_eg_B);
initial begin
      {A, B} = 8'b0000_0001;
#10 {A, B} = 8'b1001 1001;
#10 {A, B} = 8'b1011_1110;
#10 {A, B} = 8'b1101_0100;
#10 {A, B} = 8'b1111 1110;
#10 {A, B} = 8'b1011_0111;
#10 {A, B} = 8'b1001_0010;
#10 {A, B} = 8'b0010_0100;
#10 {A, B} = 8'b1011_0110;
#10 {A, B} = 8'b0110_0110;
#10 $stop;
end
endmodule
```