LABORATORY # 5

In lab5, you will:

Practice the structure model.

Build a project with structure design, which is a 4-bit lookahead adder

Write a testbench to test your design.

Analysis the simulation result.

1. **4-bit lookahead adder**

An N-bit adder adds two N-bit numbers plus a carry-in bit, resulting in an N-bit sum and a carry-out bit. A block diagram of a 4-bit adder appears in **Figure 1**.

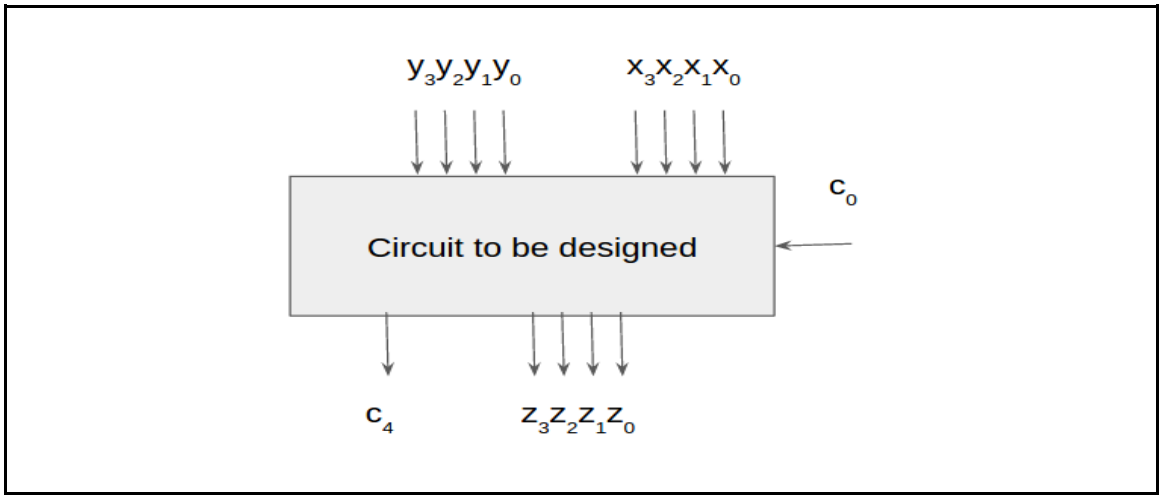

Figure L1. Block Diagram of 4-bit adder

Although we could design a 4-bit adder's circuit using the combination logic design process, the resulting circuit would be rather large. Let's assume that we are adding two n-bit numbers $x_{n-1} .. x_1 \quad x_0$ and $y_{n-1} .. y_1 y_0$. The result is $z_{n-1} .. z_1 z_0$. A single-bit full adders will add x_i, y_i and c_i and get as result z_i and c_{i+1}. The equations for these quantities are as follows

$c_{i+1} = (x_i \& y_i) \quad | (x_i \& c_i) | (yi \& ci)$

$z_i = (x_i \wedge y_i \wedge c_i)$

With simple boolean algebra, we can rewrite c_{i+1} as

$c_{i+1} = (x_i \& y_i) \quad | c_i(x_i | y_i) ;$

2. Carry bits c

It takes N full-adder delays for the carry to propagate through the carry-ripple adder. To avoid this, we can use a different design approach which targets speed.

Here the carry bits (c_n .. c_2 c_1) are pre-calculated using a separate module and fed into each full-adder. The full-adder inturn just calculates the result bits (z_{n-1} .. z_1 z_0)

Note, from the previous equation c_{i+1} can be rewritten as $c_{i+1} = g_i + p_i c_i$ where $g_i = (x_i \,\&\, y_i)$ and $p_i = x_i \mid y_i$. As result c_1 and c_2 can be written as

$c_1 = g_0 + p_0 c_0$

$c_2 = g_1 + p_1 c_1 = g_1 + p_1(g_0 + p_0 c_0) = g_1 + p_1 g_0 + p_1 p_0 c_0$

$c_3 = g_2 + p_2 c_2 = g_2 + ?$

$c_4 = g_3 + p_3 c_3 = g_3 + ?$

Likewise, derive the equations for c3 and c4. You will be needing these later.

3. Output bits z

At this point we have equations to compute all c_i. Now, to compute z_i we can use the equation

$z_i = (x_i \,\hat{}\, y_i \,\hat{}\, c_i)$

where the c_i's are as above.

Now connect four full-adders to create a 4-bit adder, as shown in **Figure 2**. The figure does not show all the connections of the inputs and outputs to the full-adders, but you should be able to determine those connections easily.

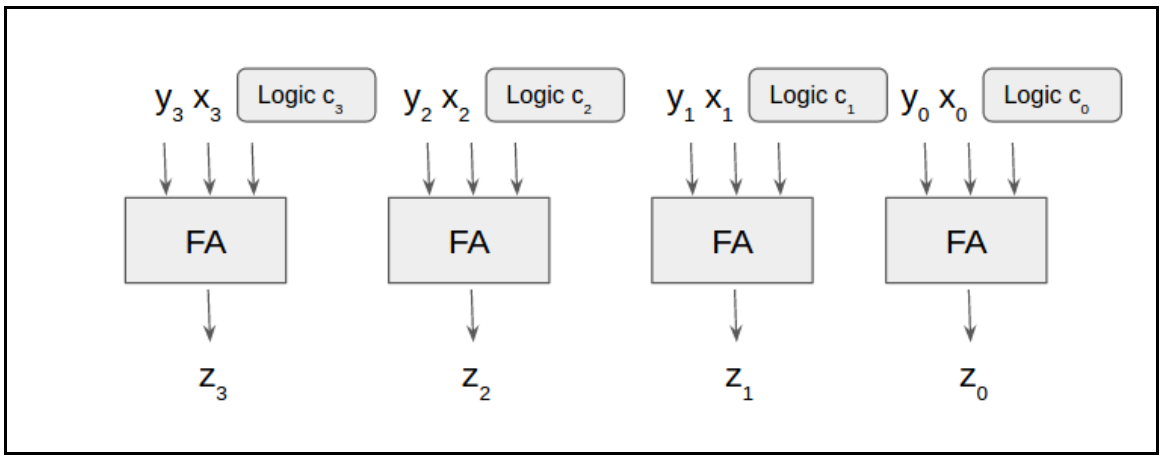

Figure 2. General structure of a 4-bit carrylookahead adder

4.Hardware implementation (not required)

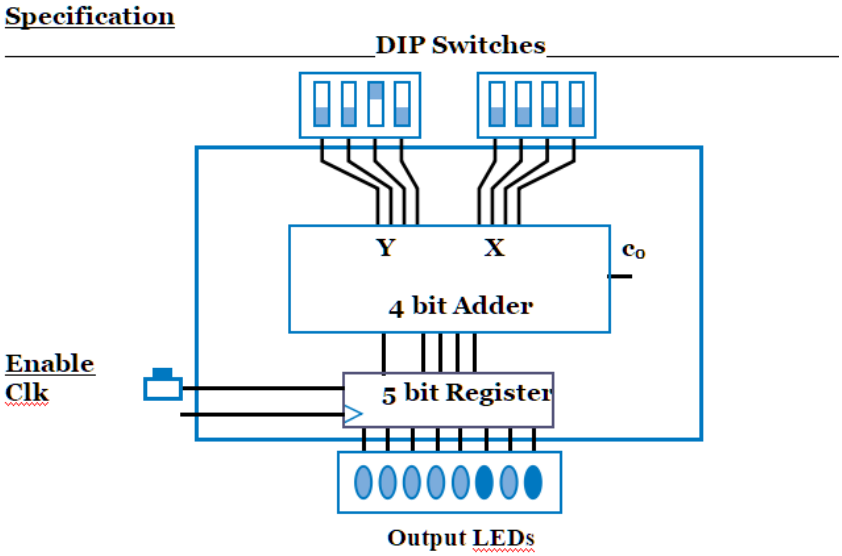

Figure 3. Adder Structure and Basys Board implementation hint

Build a new project with behavior design:

1. Open Xilinx ISE "32-bit Project Navigator" and click File>New Project

3.. Enter a "Project name" and select a "Project location" for your project.

4.. Select "Schematic" as your "Top-level source type" and click next.

Your next window should look EXACTLY like this EXCEPT for Top-Level Source Type

Property Name	Value
Top-Level Source Type	HDL
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3E
Device	XC3S100E
Package	TQ144
Speed	-5
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values

5. Click "New Source," select "verilog module," enter a "File name", delete the .sch file.

Write verilog module

6. Create and test a Full adder using structural verilog. In order to simplify your design we suggest creating four components.

7. One component to implement the logic of a full adder (code given) and one for an N-bit register (code given) . Another component to implement the logic of the carry unit (part of the code given). Finally, create a 4 bit carrylookahead_st module that uses the components already created (set this as your top level module if you are creating separate Verilog files for each module). The following are the interfaces of the modules we suggest.

```
module carrylogic(
   output [3:0] cout ,
   input cin,
   input [3:0] x,
   input [3:0] y       );
   // Computing all gx
   wire g0, g1, g2, g3 ;
   assign g0 = x[0] & y[0] ;
   assign g1 = x[1] & y[1] ;
   assign g2 = x[2] & y[2] ;
   assign g3 = x[3] & y[3] ;
   // Computing all px
   wire p0, p1, p2, p3 ;
   assign p0 = x[0] + y[0] ;
   assign p1 = x[1] + y[1] ;
   assign p2 = x[2] + y[2] ;
   assign p3 = x[3] + y[3] ;
   // Computing all carries , your code here
   assign cout[0] =
   assign cout[1] =
   assign cout[2] =
   assign cout[3] =
endmodule
```

```
module falogic(
   output r,
   input x,
   input y,
   input cin );

//your code here

endmodule
```

```
module carrylookahead_st(
   input clk ,
   input enable ,
   input cin,
   input [3:0] x,
   input [3:0] y,
   output cout,
```

```
  output [3:0] r            );
  wire [3:0] c;
  wire [3:0] ir1 ;
  wire [4:0] ir2 ;
  // Compute Carries
  carrylogic cx1 ( c, cin, x, y ) ;
  // Compute output ir1
  falogic cx6 ( ir1[0], x[0], y[0], cin     ) ;
  falogic cx7 ( ir1[1], x[1], y[1], c[0]    ) ;
  falogic cx8 ( ir1[2], x[2], y[2], c[1]    ) ;
  falogic cx9 ( ir1[3], x[3], y[3], c[2]    ) ;
  // Register
  register_logic cx10 ( clk, enable, {c[3],ir1}, ir2 )   ;
  // Results
  assign r = ir2[3:0] ;
  assign cout = ir2[4] ;
endmodule
```

8. Write a testbench to test the adder.

Deliverables:

Question1: Is it possible for two 4-bit numbers and a carry-in to result in a number too big to represent using 4 sum bits and a carry-out bit?

Question2: Why our 4-bit lookahead adder has better delay performance than combination of 4 single bit adder?

Question3: Please paste your verilog code and testbench code.

Question 4: Please paste your simulation waveform screenshot.

Please submit a .pdf file. Don’t compress to any other forms.Thank you.