Experiment 6



EXPERIMENT 6

Binary Adders

OBJECTIVES:

- Design a 1-bit full adder based on its truth table.
- Demonstrate modular design and hierarchy.
- Use Xilinx simulation tools to test combinational circuits.

MATERIALS:

- Xilinx Vivado software, student or professional edition V2018.2 or higher.
- IBM or compatible computer with Pentium III or higher, 128 M-byte RAM or more, and 8 G-byte Or larger hard drive.
- BASYS 3 Board.

DISCUSSION:

Addition and subtraction are two essential arithmetic functions performed by computers and other digital systems. It is therefore important to understand how to design a circuit to perform such functions. However, since subtraction is done by adding the 2s complement of a number, we will only need to design one adder circuit to perform both operations. An adder can be 1 or more bits. A 4-bit adder can add two 4-bit unsigned binary numbers. If larger binary numbers are to be added, an adder with more bits is needed. Let's observe what happens when adding two 4-bit binary numbers with pencil and paper:



The symbols [A3 A2 Al A0] and [B3 B2 B1 B0] represent addend and minuend, respectively. C0 is the carry bit generated by adding bits A0 and B0. C1 is the carry bit generated from the addition of C0, Al, and B1. C2 and C3 are generated in the same manner, with C3 the carry-out. The column containing A0 and B0 (the least significant bits of addend and minuend) allows for a carry-in from a previous addition, for this example we set it to 0. Each column adds three bits. The implementation of the above process in hardware called a full adder.

When we perform this addition, we will start from the least significant bit, and then push the process left one bit at a time. This means that a 1-bit full adder is the basic element of a 4-bit adder and four such elements are needed to construct a 4-bit adder.

The 1-Bit Full Adder

From the discussion above we know that a 1-bit full adder should have three inputs: carry input (Cin), addend (A), and minuend (B). We can determine the number of output bits by looking at any column in the addition process, say, the column containing CO, A1 and B1. Assume all three bits are 1. Then the result is 3 which, in binary, are 11. The sum requires two bits but S1 can be only one bit, so there must be a carry to the next column. Each column will produce a sum bit and a carry output to the next more significant bit position. So the circuit for the 1-bit full adder should have two outputs: sum bit (S) and carry output (Cout). Table 7.1 shows the truth table for the 1-bit full adder:

	Inputs	Outputs		puts
Α	В	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Using Boolean algebra, we can derive the following two equations for the sum bit and the carry output bit:

$$\mathbf{S} = \mathbf{A} \oplus \mathbf{B} \oplus \mathbf{C}$$
$$\mathbf{C}_{out} = \mathbf{C}_{in}(\mathbf{A} + \mathbf{B}) + \mathbf{A}\mathbf{B}$$

Page 2 | 19

The above two equations can be implemented using a 3-input XOR gate, two 2-input AND gates, and two 2-input OR gates.

The 4-Bit Adder

Once we have the 1-bit full-adder (FA), we can use it as a building block in any design that needs to do addition, such as the multi-bit adder in a CPU. In a multi-bit adder, the carry-in of the least significant bit (LSB) must be connected to 0 since there is no previous stage. The carry output from the LSB stage should feed into the second least significant bit. The carry output of second least significant stage feeds into the next more significant stage as carry input, and so on. The last carry output is the most significant bit of the sum. The 4-bit adder block diagram with interconnections between the FA modules is shown in figure below:



PROCEDURE: Section I. The 1-Bit Full Adder 1. Open Xilinix Vivado.

<text>

Logic Design Lab EEL3712I	Experiment 6
2. In the Xilinx-	Project Navigator window, Quick start, New Project.
HLX Editions	Create a New Vivado Project This wizard will guide you through the creation of a new project. To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.
E XILINX.	≪Back Next > Einish Cance
 Name the pro A New Project Project Name Enter a name for your proj 	ect and specify a directory where the project data files will be stored.
Project name: project Project location: C:/Xili	t S
3	< Back Next > Einish Cancel
	Page 4 19

- 4. Choose "RTL Project" and check the "Do not specify sources at this time" as we will configure all the settings manually through the navigator from inside the project.
- 5. Select New Source... and the New window appears. In the New window, choose Schematic, type your file name (such as *source_1*) in the File Name editor box, click

Project Type	
ew Project	
d Sources cify HDL, netlist, Block Design on disk and add it to your proje	, and IP files, or directories containing those files, to add to your project. Create a new source ect. You can also add and create sources later.
+, - +	
	Use Add Files, Add Directories or Create File buttons below
	Add Files Add Directories Create File
Scan and add RTL <u>i</u> nclude f	Add Files Add Directories Create File
Scan and add RTL include f	Add Files Add Directories Create File
] Scan and add RTL include f] Copy <u>s</u> ources into project] Add so <u>u</u> rces from subdirect arget language:	Add Files Add Directories <u>C</u> reate File iles into project ories Simulator language: VHDL ~
Scan and add RTL include f Copy <u>s</u> ources into project Add so <u>u</u> rces from subdirect arget language: VHDL	Add Files Add Directories <u>C</u> reate File iles into project ories ✓ Simulator language: VHDL ✓
Scan and add RTL include f Copy <u>s</u> ources into project Add so <u>u</u> rces from subdirect rarget language: VHDL	Add Files Add Directories Create File iles into project ories ✓ Simulator language: VHDL ✓ < Back
Scan and add RTL include f Copy sources into project Add sources from subdirect arget language: VHDL	Add Files Add Directories Create File iles into project ories ✓ Simulator language: VHDL ✓ < Back
Scan and add RTL include f Copy gources into project Add sources from subdirect arget language: VHDL	Add Files Add Directories Lites into project ories Simulator language: VHDL ~ < Back
Scan and add RTL include f Copy sources into project Add sources from subdirect arget language: VHDL on OK, and then cl	Add Files Agd Directories iles into project ories Simulator language: VHDL < Back

- Category: "General Purpose"
- Family: "Artix-7"
- Package: "cpg236"
- Speed: "-1"
- Choose "xc7a35tcpg236-1" that corresponds to the board we are using.

Page 5 | 19

🖕 New Projec	:t		×
Default Pa Choose a de	rt fault Xilinx pa	nt or board for your project. This can be changed later.	*
Parts	Boards Filters		
Category:	General Pu	rrpose V Package: cpg236 V Temperature: I	~
Family:	Artix-7	V Speed: -1L V	
<u>S</u> earch:	Q⊤ xc7a35t	icpg 💿 🗸 (1 match)	
Part xc7a35ti	cpg236-1L	I/O Pin Count Available IOBs LUT Elements FlipFlops Block RAMs Ultra RAMs DSPs 236 106 20800 41600 50 0 90	Gt 2
?		<back next=""> Einish C</back>	ancel
👆 New Proje	ct		×
		New Project Summary	×
New Project		New Project Summary A new RTL project named 'project' will be created.	×
	ct	New Project Summary A new RTL project named 'project' will be created. A source file will be added.	×
New Projec	ditions	 New Project Summary A new RTL project named 'project' will be created. 1 source file will be added. No constraints files will be added. Use Add Sources to add them later. 	×
New Projec	ct OOA ditions	 New Project Summary A new RTL project named 'project' will be created. 1 source file will be added. No constraints files will be added. Use Add Sources to add them later. The default part and product family for the new project: Default Part: xc7a35ticpg236-1L Product: Artix-7 Family: Artix-7 Package: cpg236 Speed Grade: -1L 	×
New Projec	et	 New Project Summary A new RTL project named 'project' will be created. 1 source file will be added. No constraints files will be added. Use Add Sources to add them later. The default part and product family for the new project: Default Part: xc7a35ticpg236-1L. Product: Artix-7 Package: cpg236 Speed Grade: -1L. 	×
		 New Project Summary A new RTL project named 'project' will be created. 1 source file will be added. No constraints files will be added. Use Add Sources to add them later. The default part and product family for the new project: Default Part xor335ticpg236-1L Product: Attx-7 Family: Artix-7 Package: cpg238 Speed Grade: -1L 	×
New Project	LINX.	New Project Summary A new RTL project named 'project' will be created. 1 source file will be added. No constraints files will be added. Use Add Sources to add them later. No constraints files will be added. Use Add Sources to add them later. The default part and product family for the new project. Default Part: xc7a35ticpg236-1L. Product: Artix-7 Package: cpg236 Speed Grade: -1L To create the project, click Finish Back Next> Enish 	Cancel

Page 7 | 19

7. The Define Module Window that will appear, we will choose the input and output labels for the gates under investigation in this experiment. In this experiment, we are investigating De Morgan's Theorem and we use 4 inputs to get 2 outputs. Under "Port Name", add "A", "B", and "Cin" as inputs and add "S", "Cout" as outputs and select OK.

MSE Por	e a module ; ach port spe B and LSB v ts with blank	and spe cified: alues v c name	ecify I/ vill be s will i	O Ports ignore not be	s to add d unless written.	to your s s its Bus	ource file. column is ch	ecked.	
Modu	Ile Definition	ı							
E	ntity name:		src1						6
Architecture name:			Beha	vioral					8
I/C	D Port Defin	itions 1	ŧ.						
1	Port Name	Direc	ction	Bus	MSB	LSB			
/	A	in	~		0	0			
	В	in	~		0	0			
	Cin	in	~		0	0			
	Cout	out	~		0	0			
	oour		~		0	0			
•	S	out							

8. In the "source_1.vhd" created file, type the gates equivalent VHDL code for the S and Cout between the "begin" and "end Behavioral" as follows and then save the file.

```
library IEEE;
1
2
         use IEEE.STD_LOGIC_1164.ALL;
3
4 Θ
         entity full_adder_vhdl_code is
5
         Port ( A : in STD LOGIC;
6
         B : in STD LOGIC;
7
         Cin : in STD LOGIC;
         S : out STD LOGIC;
8
         Cout : out STD LOGIC);
9
10 🛆
         end full adder vhdl code;
11 ;
12 🖯
         architecture gate_level of full_adder_vhdl_code is
13
14
         begin
15
     ○ S <= A XOR B XOR Cin ;</p>
16
17
     Ο
         Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;
18
19 🖨
         end gate_level;
```

- 9. Next, we need to add a constraint file with the".xdc" extension, as following: Go to "Flow Navigator" and from "Project Manager" select "Add Sources" then "Add or create constraints". Next, choose "Create File" and enter the file name "lab_6" then "OK" followed by "Finish".
- 10. Then, we need to get a template xdc file that is going to be edited according to the different experiments. Google "basys 3 xdc file" and choose the "xilinix" link that appears (<u>https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2015x/Basys3/Supporting%20Material/Basys3_Master.xdc</u>). Copy the whole file and paste it into the "lab_2.xdc" that you have just created in the last step. Then uncomment and edit the input Switches and the output LEDs as in the next step.
- 11. Uncomment (by deleting the # sign) the ones you are going to use as following:

```
10
 11 ! ## Switches
 12 set property PACKAGE PIN V17 [get ports {A}]
 13 :
         set property IOSTANDARD LVCMOS33 [get ports {a}]
 14 | set property PACKAGE_PIN V16 [get ports {B}]
 15
        set property IOSTANDARD LVCMOS33 [get ports {B}]
 16 | set property PACKAGE_PIN W16 [get ports {Cin}]
 17
         set property IOSTANDARD LVCMOS33 [get ports {Cin}]
 18 | #set property PACKAGE PIN W17 [get ports {sw[3]}]
45
46 ## LEDs
47 | set property PACKAGE PIN U16 [get ports {S}]
       set property IOSTANDARD LVCMOS33 [get ports {S}]
48 .
49 | set property PACKAGE_PIN E19 [get ports {Cout}]
    set property IOSTANDARD LVCMOS33 [get ports {Cout}]
50 i
51 ; #set property PACKAGE PIN U19 [get ports {led[2]}]
52
    #set property IOSTANDARD LVCMOS33 [get ports {led[2]}]
53 | #set property PACKAGE PIN V19 [get ports {led[3]}]
54
      #set property IOSTANDARD LVCMOS33 [get ports {led[3]}]
55 #set property PACKAGE PIN W18 [get ports {led[4]}]
56
      #set property IOSTANDARD LVCMOS33 [get ports {led[4]}]
```



13. The implementation errors window will appear if any or the successfully completed window. From this window select "Generate Bitstream" and then OK. This will make the software generate ".bin" file to be used in programing the hardware BAYAS 3.

mplementation Completed	×
Implementation successfully completed.	
Open Implemented Design	-
Generate Bitstream	
◯ <u>V</u> iew Reports	
Don't show this dialog again	
OK Cancel	

14. The next window will appear in which choose "Open Hardware Manger", connect the Hardware Kit to the USB port and then press OK.

Logic Design Lab EEL3712l		Experiment 6
	Bitstream Generation Completed	×
	Bitstream Generation successfully completed.	
	Next	
	Open Implemented Design	
	◯ <u>V</u> iew Reports	
	Open <u>H</u> ardware Manager	
	<u>Generate Memory Configuration File</u>	
	Don't show this dialog again	
	OK Cancel	
	CAPET -	

- 15. A green tab will appear in the top of the Vivado window, from which choose "open target" to program the hardware.
- 16. From the window appears, select the ".bin" file from the Project you create by browsing for the generated ".bit file" under the ".runs" folder and program the board then press OK.

Specify Bits	tream File	
ook in:	impl_1	✓ ↑ ☆ 果 ± A B X C Ⅱ
.Xil	_	Recent Directories
👫 source_1	bit	C:/Xilinx/Vivado/2018.2
		File Preview
		File: source_t_bit Directory: CS/MinV/Wado/2018.2/project/project.runs/impl_1 Created: Today at 16:31 PM Accessed: Today at 16:31 PM Modified: Today at 16:31 PM Size: 2.1 MB Type: Bitsteam file Owner: ECE-3865-832F/mabde030
ile <u>n</u> ame:	source_1.bit	
iles of type:	Bitstream Files (bit bin, rbt)	
		OK Car
		Page 10 19

Logic Design Lab EEL3712

17. Notice that the 7-segment on the hardware is counting up from 0 to 9 frequently until you download the program and it will stop.



18. Test the program on your board by going through all the input combinations and observing the two outputs. Fill the truth table.

	Inputs		Out	puts
Α	В	Cin	Cout	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

- 19. Are the two output the same? If they are, you have proved the Boolean distributive law. If not, figure it out.
- 20. Then you can use the simulation tools to verify the Boolean distributive law. For simulation, we need to create a simulation source file as following:

- 21. "Flow Navigator" → "Project Manager" → "Add Sources" → "Add or create simulation sources" → Name it "TB" (Test Bench) → "VHDL" → No need for switches and leds assignments as we will not be working on board. → "OK".
- 22. After that, implement your simulation as similarly:

```
1
         LIBRARY ieee;
2 |
         USE ieee.std_logic_1164.ALL;
 3 ¦
 4 🖯
         ENTITY Testbench_full_adder IS
 5 🖨
         END Testbench_full_adder;
 6
7 🖯
         ARCHITECTURE behavior OF Testbench_full_adder IS
 8
 9
          -- Component Declaration for the Unit Under Test (UUT)
                                                                         40 i
10
                                                                         41 :
                                                                                    -- Stimulus process
11 Ö
          COMPONENT full_adder_vhdl_code
                                                                         42 🖯
                                                                                    stim proc: process
12
          PORT (
13
          A : IN std logic;
                                                                         43
                                                                                    begin
          B : IN std logic;
14
                                                                         44
                                                                                    -- hold reset state for 100 ns.
          Cin : IN std logic;
15
                                                                               0
                                                                                    wait for 100 ns;
                                                                         45
16
          S : OUT std logic;
                                                                         46
17
          Cout : OUT std logic
                                                                         47
                                                                                    -- insert stimulus here
18
          );
                                                                               0:
                                                                         48
                                                                                   A <= '1';
19 🛆
          END COMPONENT;
                                                                               0 1
                                                                                   B <= '0';
20
                                                                         49 i
21
          --Inputs
                                                                               0
                                                                         50
                                                                                    Cin <= '0';
22
          signal A : std logic := '0';
                                                                         51
                                                                               Ο
                                                                                    wait for 10 ns;
23
          signal B : std logic := '0';
                                                                         52
24
          signal Cin : std logic := '0';
                                                                               ○ : A <= '0';</p>
                                                                         53
25
                                                                               0
                                                                         54 i
                                                                                    B <= '1';
26
          --Outputs
                                                                         55 ¦
                                                                               0
                                                                                    Cin <= '0';
27
          signal S : std_logic;
28
          signal Cout : std logic;
                                                                               0
                                                                                    wait for 10 ns;
                                                                         56 i
29
                                                                         57
30
         BEGIN
                                                                               0
                                                                         58
31
                                                                              ○ A <= '1';
                                                                         59 !
32
          -- Instantiate the Unit Under Test (UUT)
                                                                              0
                                                                                    B <= '1';
                                                                         60
33 🖯
          uut: full_adder_vhdl_code PORT MAP (
                                                                              0
                                                                         61 i
                                                                                    Cin <= '1';
34
          A \Rightarrow A,
35
          B \Rightarrow B,
                                                                         62
                                                                                    wait for 10 ns;
36
          Cin => Cin,
                                                                               0
                                                                         63 i
37
          S => S,
                                                                         64 A O
                                                                                  end process;
38
          Cout => Cout
                                                                               0
                                                                         65
39 🗀
          );
                                                                         66 🔶 🔘 END;
40
```

- **23.** In the "initialization" section change the simulation variables according to your needs.
- 24. You should see similar output:

ogic Design	Lad EEL3	/ 121				Experiment 6
full_adder.vh	d x cons	6.xdc × sim6.vhd	× Untitled 4	×		
Q 💾	Ð. Q.	s - r I∢ ►l	te er +	Γ⇔ ⊨Γ	[+→	
				275.862 ns		
Name	Value	0 ns	200 ns		400 ns	
1 8 A	1					
le B	0					
Cin	0					
li S	1	┝────┘ 빌닏				
le Cout	0					

Section II. Building a 4-bit Adder Using Full-Adder (FA) Modules

In this part of the experiment, we will show how to perform modular design by building the 4-bit adder using four 1-bit full adder modules. We will start with making a symbol for a 1-bit full adder and add it as a module to the project library.

1. Create a new source file called four_bit_adder under the same project. Write the following code:



2. Then create the simulation as you did before and write the following code for simulation.

);

```
Experiment 6
```

```
1
         LIBRARY ieee;
 2
         USE ieee.std logic_1164.ALL;
 3
         ENTITY Tb_Ripple_Adder IS
 4
         END Tb_Ripple_Adder;
 5
 6
 7
         ARCHITECTURE behavior OF Tb_Ripple_Adder IS
8
9
         -- Component Declaration for the Unit Under Test (UUT)
10
         COMPONENT Ripple Adder
11
12
         PORT (
13
         (A : IN std logic vector(3 downto 0);
14
         B : IN std logic vector(3 downto 0);
         Cin : IN std logic;
15
         S : OUT std logic vector(3 downto 0);
16
         Cout : OUT std logic
17
18
         );
         END COMPONENT;
19
20
21
         --Inputs
22
         signal A : std logic vector(3 downto 0) := (others => '0');
23
         signal B : std_logic_vector(3 downto 0) := (others => '0');
24
         signal Cin : std logic := '0';
25
         --Outputs
26
27
         signal S : std logic vector(3 downto 0);
28
         signal Cout : std logic;
29
         BEGIN
30
31
32
         -- Instantiate the Unit Under Test (UUT)
33
         uut: Ripple_Adder PORT MAP (
34
         A => A,
35
         B => B,
         Cin => Cin,
36
37
         S => S,
38
         Cout => Cout
39
```

Page 15 | 19

Experiment 6

40	
41	Stimulus process
42	stim_proc: process
43	begin
44	hold reset state for 100
45	Wait for 100 ns;
46	○ A <= "0110";
47	○ B <= "1100";
48	
49	wait for 100 ns;
50	○ 'A <= "1111";
51	○ B <= "1100";
52	
53	Wait for 100 ns;
54	○ A <= "0110";
55	○ ˈB <= "0111";
56	
57	Wait for 100 ns;
58	○ A <= "0110";
59	○ B <= "1110";
60	
61	Wait for 100 ns;
62	○ A <= "1111";
63	○ B <= "1111";
64	
65	O wait;
66	
67	end process;
68	
69	END;
70	

3. Example output of simulation:

Name	Value	0 ns	<u> </u>	200 ns		400	ns 	 600 ns		800 ns	
> • A[3.0]	0 7		×	<u>f</u>	×		=		f		
Cin	0	<u> </u>	^		^ 		<u> </u>		1		
> 😼 S[3:0]	d	0	2	ь	X 1		4		e		
🔓 Cout	0										

ns.

Switches(Input)										LEDs(Output)					
1	2	3	4	imal	5	6	7	8	mal	1	2	3	4	5	mal
A 3	A 2	A 1	\mathbf{A}_0	Deci	B 3	B2	Bı	Bo	Deci	Cout	S 3	S2	S1	S0	Deci
0	0	0	0	0	0	0	0	0	0						
0	0	0	1	1	0	0	0	0	0						
0	0	1	0	2	0	0	0	1	1						
0	0	1	1	3	0	0	1	0	2						
0	1	0	0	4	0	0	1	1	3						
0	1	0	1	5	0	0	1	1	3						
0	1	1	1	7	0	0	1	0	2						
1	0	0	0	8	0	0	1	0	2						
0	1	0	0	4	1	0	0	1	9						
0	1	0	1	5	1	1	0	0	12						
0	1	1	1	7	1	1	1	0	14						
1	0	1	0	10	0	1	1	1	7						
1	1	0	0	12	1	0	1	1	11						
1	1	1	1	15	1	1	1	1	15						

4. Fill the table accordingly by changing your initialization part of the code:

Page 17 | 19

QUESTIONS

1. Find the ADD4 symbol in the symbol library. Draw a schematic diagram in the space provided below and show how to make an 8-bit adder using the ADD4s.

2. If you are asked to build and simulate the above design, what types of I/O buffers (symbols) would be convenient to use? How many sum bits would you expect to have?

3. What is a carry look-ahead adder? Why is it preferred over a regular adder? (Refer to your textbook and diagrams of Ripple Carry and Carry Lookahead in the following.)

Page 18 | 19

