An-Najah National University
Faculty of Engineering and IT

جامعة النجاح الوطنية
كلية الهندسة وتكنولوجيا المعلومات

Computer Engineering Department

Course Name: Digital design Lab 2 **Number: 10636391**

Lab Report Grading Sheet

Instructor: Dr Ashraf Armoush	Experiment #: 1
Academic Year: 2020/2021	Experiment Name: 4-bit Adder with Structural and Behavioral Implementation
Semester: Summer	

Students	
1-Ashraf Habromman	2-Ra'ed Khwayerh
Performed on: 17/6/2021	Submitted on: 24/6/2021

Report's Outcomes

ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %

Evaluation Criterion	Grade	Points
Abstract answers of the questions: "What did you do? How did you do it? What ?"did you find	0.5	
Introduction and Theory Sufficient, clear and complete statement of objectives. In addition to Presents sufficiently the theoretical basis.	1.5	
Apparatus/ Procedure Apparatus sufficiently described to enable another experimenter to identify the equipment needed to conduct the experiment.Procedure .sufficiently described	2	
(Experimental Results and Discussion (In-Lab Worksheet Crisp explanation of experimental results. Comparison of theoretical predictions to experimental results, including discussion of accuracy .and error analysis in some cases	4	
Conclusions and Recommendations Conclusions summarize the major findings from the experimental results with adequate specificity. Recommendations appropriate in .light of conclusions. Correct grammar	1	
Appearance Title page is complete, page numbers applied, content is well organized, correct spelling, fonts are consistent, good visual appeal.	1	
Total	10	

4-bit Adder with Structural and Behavioral Implementation

Introduction:

We learned in digital 2, how to make a half adder that adding two bits only, then we use it to make a full adder by using two half adder to add two bits with carry, then we went to make a 4-bit ripple carry adder to adding four bits, we made by using four full adders. We learned it theoretically and applied that practically by using VHDL in EDA PlayGround, and now in our lab, we applied that practically by using software (Vivado) and showing that on hardware (ZedBoard).
Note: When we build 4-bit ripple carry adder we ignored input carry because in ZedBoard there is only 8 switch and the sum of input carry and A, B bits is 9 so we put input carry = 0;

Abstract:

We'll learn how to build a half adder then build a full adder by using two half adder then build a 4-bit ripple carry adder by using four full adders and then we applied simulation then connected our I/O on ZedBoard in the Constraint file. then applied Synthesis and implementation then generated bitstream and installed that on ZedBoard to show results.

Apparatus:

1-Vivado Design Suite HL WebPACK™ Edition.
2-ZedBoard.

Procedure:

Part1: 4-bit adder with structural description:

Using the hierarchy method the experimenters have implemented a half adder entity and use it to implement a full adder entity, so can use it to build the 4-bit adder.

Half adder component:

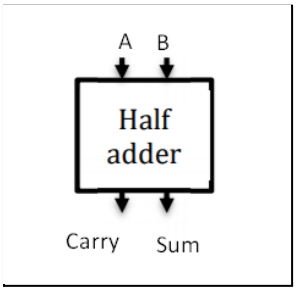


Fig1: half adder design

The truth table of half adder:

B	A	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth table 1: half adder truth table.

So, **sum = A xor B**, **carry = A and B**.

Half Adder implementation code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity halfadder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           s : out STD_LOGIC;
           c : out STD_LOGIC);
end halfadder;

architecture Behavioral of halfadder is

Begin
           s<= a xor b;
           c<= a and b;

end Behavioral;
```

Half Adder testbench code :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity halfadder_tb is
--  Port ( );                       -- no ports in the testbench entity
end halfadder_tb;

architecture Behavioral of halfadder_tb is
        component halfadder
                port(
                        a: in std_logic;
                        b: in std_logic;
                        s: out std_logic;
                        c: out std_logic
                );
        end component;
        signal a : std_logic := '0';
        signal b : std_logic := '0';
        signal s : std_logic := '0';
        signal c : std_logic := '0';
Begin
        uut: halfadder port map(a=> a,b=> b,s=> s,c=>c);    -- map signals with inputs and outputs

        stimulus: process

                begin
                        a<= '0';
                        b<= '0';
                        wait for 10 ns;

                        a<= '1';
                        b<= '0';
                        wait for 10 ns;

                        a<= '0';
                        b<= '1';
                        wait for 10 ns;

                        a<= '1';
                        b<= '1';
                        wait for 10 ns;

                        a<= '0';
```

```
                b<= '0';
                wait;
        end process;

end Behavioral;
```

The waveform of the half adder testbench:

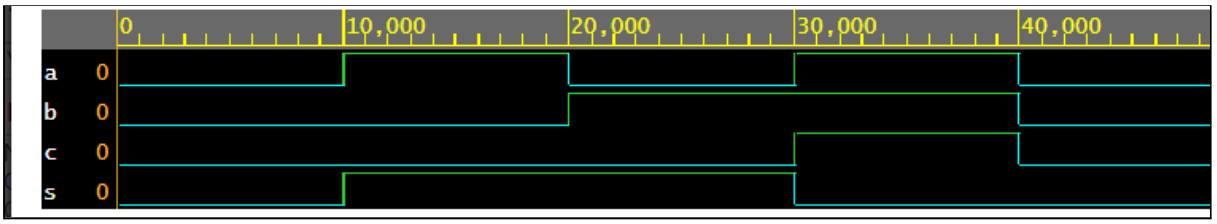


Fig2: half adder testbench waveform.

Full adder component:

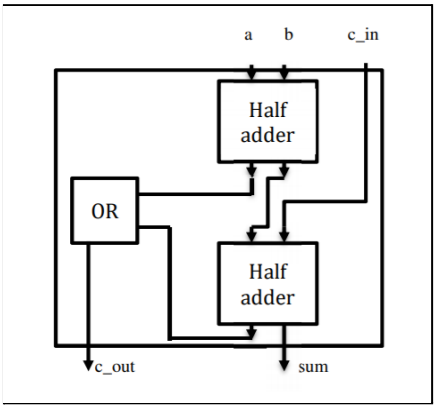


Fig3: full adder design.

The truth table of full adder:

C_in	B	A	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table 2: full adder truth table.

So as shown in Fig4, the components used to build a full adder is two half adder components and one OR gate. So for sure, the half adder design in the last part is used to build the full adder(the file should exist in the project).

Full Adder implementation code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fulladder is
    Port ( fa : in STD_LOGIC;
           fb : in STD_LOGIC;
           fcin : in STD_LOGIC;
           fs : out STD_LOGIC;
           fcout : out STD_LOGIC);
end fulladder;

architecture structural of fulladder is

           component halfadder  is
                    Port ( a,b : in STD_LOGIC;
                             s,c : out STD_LOGIC);
           end component;

           signal cH1,sH1,cH2: STD_LOGIC;

begin
           U1:halfadder PORT MAP(a=>fa,b=>fb,s=>sH1,c=>cH1);
           U2:halfadder PORT MAP(a=>sH1,b=>fcin,s=>fs,c=>cH2);

           fcout <= cH1 or cH2;

end structural;
```

Full Adder testbench code :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fulladder_tb is
--  Port ( );
end fulladder_tb;

architecture Behavioral of fulladder_tb is
```

```
        component fulladder
        port(
                fa: in std_logic;
                fb: in std_logic;
                fs: out std_logic;
                fcin: in std_logic;
                fcout: out std_logic
                );
        end component;
        signal fa : std_logic := '0';
        signal fb : std_logic := '0';
        signal fs : std_logic := '0';
        signal fcin : std_logic := '0';
        signal fcout : std_logic := '0';

Begin
        uut: fulladder port map(fa=> fa,fb=> fb,fs=> fs,fcout=>fcout,fcin=>fcin);

stimulus: process

        begin
                fa<= '0';
                fb<= '0';
                fcin<='0';
                wait for 10 ns;

                fa<= '1';
                fb<= '0';
                fcin<='0';
                wait for 10 ns;

                fa<= '0';
                fb<= '1';
                fcin<='0';
                wait for 10 ns;

                fa<= '1';
                fb<= '1';
                fcin<='0';
                wait for 10 ns;

                fa<= '0';
                fb<= '0';
                fcin<='1';
                wait for 10 ns;
```

```
            fa<= '1';
            fb<= '0';
            fcin<='1';
            wait for 10 ns;

            fa<= '0';
            fb<= '1';
            fcin<='1';
            wait for 10 ns;

            fa<= '1';
            fb<= '1';
            fcin<='1';
            wait for 10 ns;

            fa<= '0';
            fb<= '0';
            fcin<='0';
            wait ;
    end process;

end Behavioral;
```

The waveform of the full adder testbench:

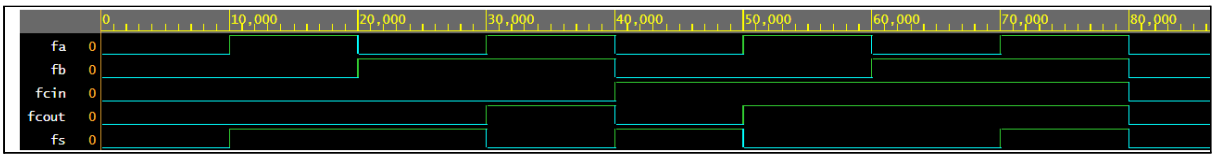
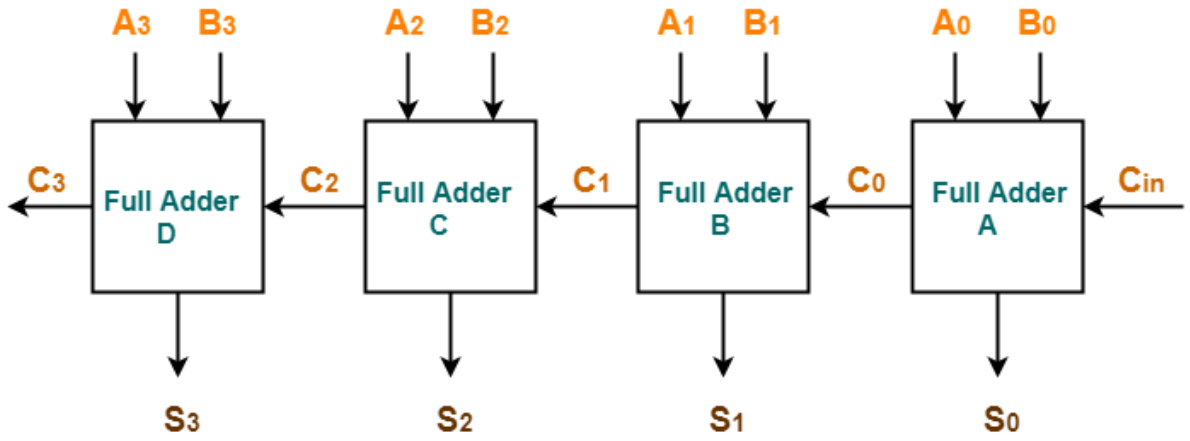


Fig4: full adder testbench waveform.

4-Bit adder component:



4-bit Ripple Carry Adder

The truth table of 4-Bit Ripple Carry adder:

	A				B				Sum				Carry
Cin	A3	A2	A1	A0	B3	B2	B1	B0	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	1	1	1	0	1	1	1	1	1	1	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	1	0	0	1	0	0	0	1
0	1	1	0	1	1	1	0	1	1	0	1	0	1
0	1	1	1	0	1	1	1	0	1	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

4-Bit Ripple Carry Adder implementation code:

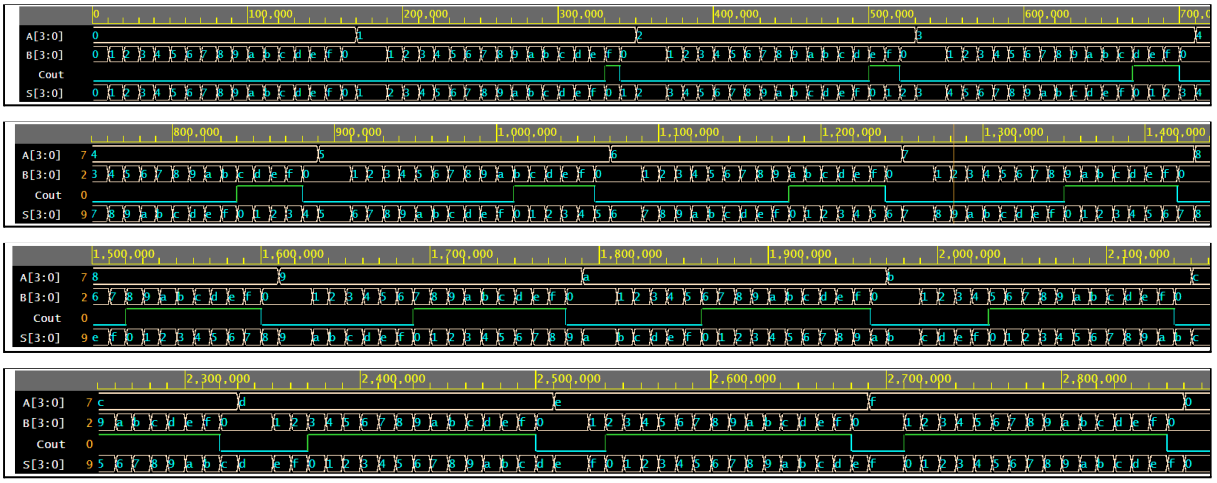
```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FOURbitadder is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Cout : out STD_LOGIC);
end FOURbitadder;

architecture Behavioral of FOURbitadder is
component fulladder  is

Port ( fa,fb,fcin : in STD_LOGIC;

       fs,fcout : out STD_LOGIC);

end component;
signal cF1,cF2,cF3 : std_logic;
begin
U1:fulladder PORT MAP(fa=>A(0),fb=>B(0),fs=>S(0),fcin=>'0',fcout=>cF1);

U2:fulladder PORT MAP(fa=>A(1),fb=>B(1),fs=>S(1),fcin=>cF1,fcout=>cF2);

U3:fulladder PORT MAP(fa=>A(2),fb=>B(2),fs=>S(2),fcin=>cF2,fcout=>cF3);

U4:fulladder PORT MAP(fa=>A(3),fb=>B(3),fs=>S(3),fcin=>cF3,fcout=>Cout);
end Behavioral;
--------------------------------------------------------------------------------
```

4-Bit Ripple Carry Adder testbench code :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fourbit_tb is
end fourbit_tb;
architecture Behavioral of fourbit_tb is
component FOURbitadder
```

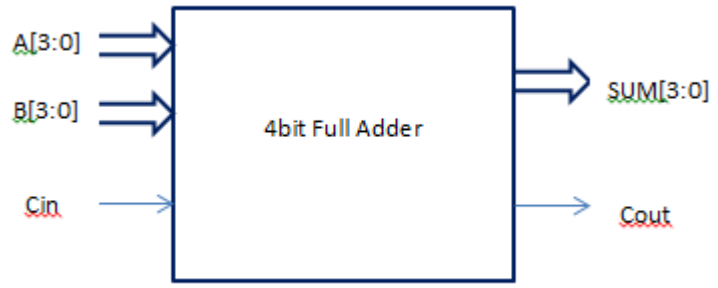
```
  Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
         B : in STD_LOGIC_VECTOR (3 downto 0);
         S : out STD_LOGIC_VECTOR (3 downto 0);
         Cout : out STD_LOGIC);
end component;

signal A: STD_LOGIC_VECTOR (3 downto 0):="0000";
signal B :STD_LOGIC_VECTOR (3 downto 0):="0000";
signal S : STD_LOGIC_VECTOR (3 downto 0):="0000";
signal Cout : std_logic := '0';

begin
utt: FOURbitadder port map(A=>A, B=>B, S=>S ,Cout=>Cout);

stimulus: process
begin
    wait for 10 ns;
        for i in 0 to 15 loop
                for j in 0 to 15 loop
                        B<= B + '1';
                        wait for 10 ns;
                end loop;
                A<= A + '1';
                wait for 20 ns;
        end loop;
        wait;
end process;

end Behavioral;
```

The waveform of the 4-Bit Ripple Carry Adder testbench:



Part2: 4-bit adder with behavioural description:

4-Bit adder with behavioural description component:



The truth table of 4-Bit adder with behavioural description:

	A				B				Sum				Carry
Cin	A3	A2	A1	A0	B3	B2	B1	B0	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	1	1	1	0	1	1	1	1	1	1	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	1	0	0	1	0	0	0	1
0	1	1	0	1	1	1	0	1	1	0	1	0	1
0	1	1	1	0	1	1	1	0	1	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

4-Bit adder with behavioural implementation code:

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
```

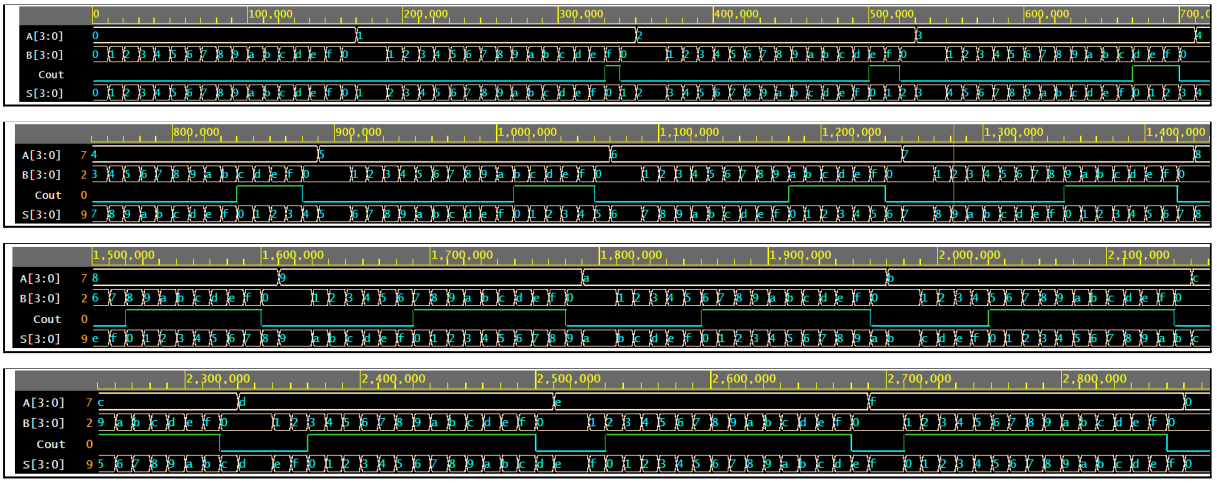
```
entity fourbitadder is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           S : out STD_LOGIC_VECTOR (3 downto 0);
           C : out STD_LOGIC);
end fourbitadder;

architecture Behavioral of fourbitadder is
Begin
           process(A,B)
                      variable X,Y,Z : std_logic_vector (4 downto 0):="00000";
                      -- define 5-bit variables to use the fifth bit in the result(Z) as a curry_out.
           Begin
                      X:=('0'&A);
                      Y:=('0'&B);
                      Z:=X+Y;
                      S<=Z(3 downto 0);
                      C<=Z(4);
           end process;
end Behavioral;
----------------------------------------------------------------------------------
```

4-Bit adder with behavioural testbench code :

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity fourbitadder_tb is
--  Port ( );
end fourbitadder_tb;

architecture Behavioral of fourbitadder_tb is
component fourbitadder
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          S : out STD_LOGIC_VECTOR (3 downto 0);
         C : out STD_LOGIC);
end component;

signal A: STD_LOGIC_VECTOR (3 downto 0):="0000";
signal B :STD_LOGIC_VECTOR (3 downto 0):="0000";
signal S : STD_LOGIC_VECTOR (3 downto 0):="0000";
```

```
signal C : std_logic := '0';

begin
utt: fourbitadder port map(A=>A, B=>B, S=>S, C=>C);
stimulus: process
        begin
                wait for 10 ns;
                for i in 0 to 15 loop
                        for j in 0 to 15 loop
                                B<= B + '1';
                                wait for 10 ns;
                        end loop;
                        A<= A + '1';
                        wait for 20 ns;
                end loop;
                wait;
        end process;
end Behavioral;
```

The waveform of the 4-Bit adder with behavioural Adder testbench:



Then by generating a bitstream to download it on the Zedboard and by connecting the inputs with switches and the outputs with LEDs. Note the first curry in did not connect to any of the switches and that is because there is no more switch to use it.

Conclusion:

By this experiment, the experimenters have learned how to build 4-bit adder using structural description and behavioural description. The structural part was using a component which was already built, which is the full adder that built using two half adders and one OR gate. The behavioural part was using a simple addition sign but using an additional library, in addition to some processes on the data in and out.