MÜHENDİSLİK FAKÜLTESİ
FACULTY OF ENGINEERING
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 23 02
F: +90 312 210 23 04
ee@metu.edu.tr
www.eee.metu.edu.tr

EXPERIMENT 1 - Parallel Adders, Subtractors, and Complementors

I Introduction

I.A Objectives

In this experiment, you will design and investigate parallel adders, subtractors and complementors. The design will take place in Quartus II 14.1 software and you will test your final implementation in FPGA. In this experiment, you need to download your designs to the FPGA and check the results by physical means, i.e., using LEDs. Also, you will implement the corresponding designs in Verilog hardware description language. Another objective of this experiment is exposing you to the hierarchical design method for logic circuits.

I.A.1 Background

Digital computers perform a number of arithmetic operations for information processing. These tasks are performed using various arithmetic logic circuits. Adders, subtractors and complementors are prevalent basic arithmetic circuits each of which has a short description below.

1.a Adders

Adders are divided into two groups: half adders and full adders. While half adders add two single bits and return sum and carry bits, full adders add three bits where one of them is carry bit from the preceding adder and return the same. The block diagram of a full adder is in **Figure 1.1** and the truth table of it is in **Table 1.1**. Two of the input variables, denoted by X_k and Y_k, represent two significant bits to be added. The third input, C_{k-1} represents the carry bit from the preceding adder of lower significant bits.

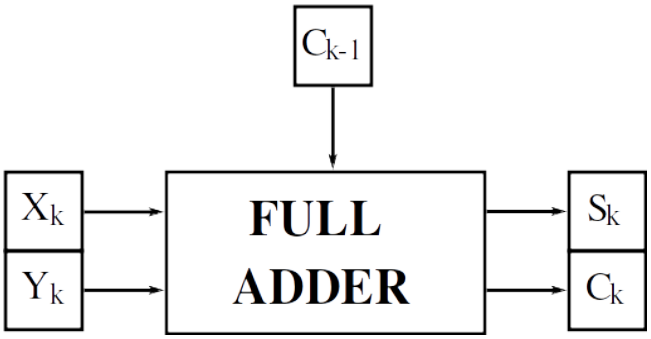


Figure 1.1 Block diagram of a full adder

Table 1.1 Truth table of a full adder ($X_k + Y_k + C_{k-1}$)

X_k	Y_k	C_{k-1}	S_k	C_k
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

1.b Subtractors

Subtractors are similar to the adders. There are full subtractors with three inputs one of which is the 'borrow' from the preceding subtractor. The two outputs are a difference bit and a borrow to the succeeding unit. Half subtractors do not have a borrow input. **Figure 1.2** shows the block diagram of a full subtractor. The inputs X_k, Y_K stands for the single bits to subtract and B_{k-1} is for borrow for the previous unit. The output D_k is for the difference result of the inputs and B_k is for the borrow bit for the next unit. **Table 1.2** gives truthtable for the full subtractor operation.

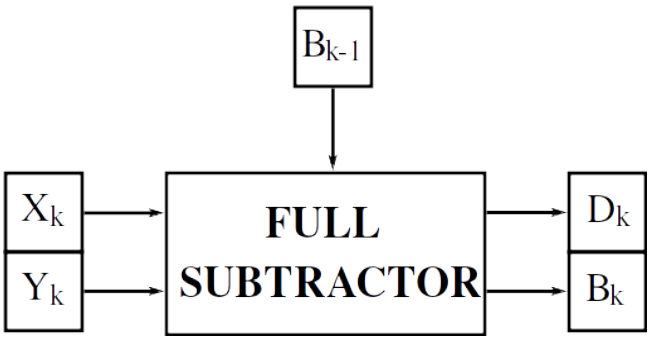


Figure 1.2 Block diagram of a full subtractor.

Table 1.2 Truth table of a full subtractor (X_k -Y_k - B_{k-1})

X_k	Y_k	B_{k-1}	D_k	B_k
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

1.c Complementors

Complementor units are of two types: 1's complement unit and 2's complement unit. The truthtable of 2-bit 1's complement and 2's complement units is in **Table 1.3**.

Table 1.3 Truth table of complementors

Input Bits	1's Compliment	2's Compliment
00	11	00
01	10	11
10	01	10
11	00	01

Note: 2's complement of a number is its 1's complement plus binary one.

The arithmetic unit of computers usually employs either 1's complement arithmetic or 2's complement arithmetic. Out of these two, the usage of 2's complement is more frequent. These units accept two operands (positive or negative) and perform an addition. If a subtraction is desired, first, the unit takes the complement of the subtrahend and, then, adds it to the minuend. Some examples are as follows:

i. Sign-magnitude arithmetic:

+7	+00111	+7	+00111
+ +5	+ +00101	- +5	- +00101
+12	+01100	+2	+00010

ii. 1's complement arithmetic:

+7	0111	
+ -5	+ 1010	(1's complement of 0101)
+2	1 ← 0001	
	+ →1	
	0010	

iii. 2's complement arithmetic:

+7	0111	
+ -5	+ 1011	(2's complement of 0101)
+2	1 ← 0001	(Ignore the carry bit)

The most significant bit of an n-bit word refers to the sign bit in the signed arithmetic. The remaining n-1 bits determines the magnitude of the n bits number. If the result of a 2's complement arithmetic operation exceeds the range of the available bit length (overflow for a large positive number or for a small negative number), then the unit returns an invalid result. A combinational circuit can detect this kind of erroneous results.

II Preliminary Work

1 Read sections 1.5 Complements(pp 10-14) and 4.5 Binary Adder–Subtractor(pp 130-139) from the textbook "Digital Design" by M. Mano and M. Ciletti (4th Ed., Prentice Hall). You can find related chapters in other editions also.

2 Design a half adder and a half subtractor using minimum number of gates using two-input NAND (7400) and two-input XOR (7486). Construct the truthtables for both of your designs. Draw and explain your designs in detail.

3 Design a full adder and a full subtractor using minimum number of two-input NAND (7400) and two-input XOR (7486) gates. Show how you can use half adders and half subtractors to build a full adder and full subtractor. Show and explain your designs in detail.

4 Design a 4-bit binary adder by using full adders. You Do not have to draw full adders in gate level. Instead, you should combine full adder blocks with the appropriate input and output connections. At the end, your design should take in 9 inputs representing two 4-bit numbers and a carry in, and 5 outputs representing one 4-bit number (which is the sum) and a carry out.

5 Design a 4-bit 2's complement unit with a control signal E such that

When E=0 output is the 2's complement of the input;

When E=1 output is the same as the input.

You can use two-input NAND (7400), 4-bit binary adder (7483) and two-input XOR (7486) gates.

6 Design an arithmetic unit that accepts 4-bit wide parallel numbers (X, Y) and an enable signal (E). Assume that input numbers are represented in 2's complement arithmetic. Enable signal E has a function such that

When E=1 unit performs addition (X+Y);
When E=0 unit performs subtraction (X-Y).

You can use two-input NAND (7400), 4-bit binary adder (7483) and two-input XOR (7486) gates.

7 Consider the circuit you designed in **Part 6**. Prepare a table showing the state of the outputs when input numbers and control signal are given as in **Table 1.4**.

Table 1.4 Input signals that should be applied to the circuit designed in part 6

E=1 X=0110 Y=0011	E=1 X=1010 Y=0111
E=1 X=0111 Y=0001	E=1 X=-5 Y=-8
E=0 X=0110 Y=0001	E=0 X=0011 Y=0100
E=0 X=0011 Y=0100	E=0 X=-1 Y=-4

III Experimental Work

III.A Designing 4-Bit Binary Adder

Throughout the experimental work, you will use Quartus II software and Altera FPGAs. As in **Part 4** and **Part 5** of the preliminary work using 7483 IC, namely the 4-bit binary adder, you will design a 4-bit binary adder and form a component out of it using the hierarchical design method. Then, you will add it to the symbol directories of your higher-level designs for later works. Note that your 4-bit binary adder will only utilize XOR and NAND gates.

III.A.1 Implementing the full-adder

1 Create an empty project named **Experiment1** in the folder ***Experiment1***. Create a new schematic file as ***FullAdder.bdf*** in the sub-folder as ***FullAdder*** in the folder ***Experiment1***. Do not forget to click **"Add file to current project"** option.

2 Based on the logic circuit design in your preliminary work **Part 3**, create the full adder by adding appropriate components, input/output pins and wiring.

3 In the files section find ***FullAdder.bdf*** and choose **"Set as Top-Level Entry"** from right click menu as shown in **Figure 1.3**. Then, to start compilation, follow **Processing > Start Compilation** and let the compilation end.

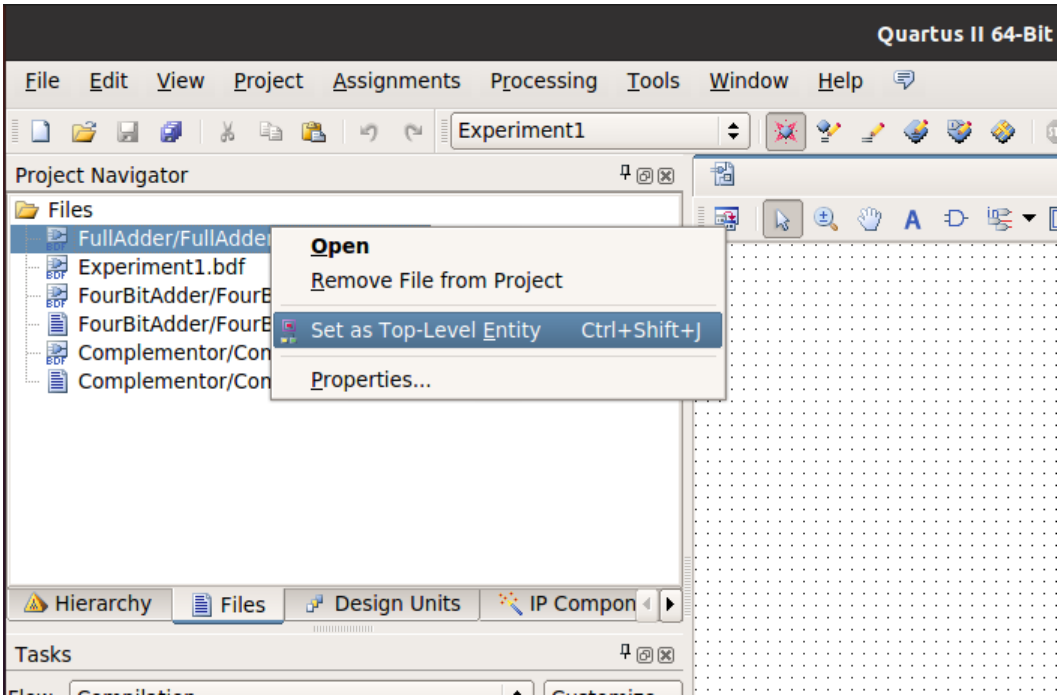


Figure 1.3 Setting as top file

III.A.2 Creating the symbol

1 After determining that the current project functions correctly, you need to create a symbol for this block to be able to use it in a higher-level of the design hierarchy.

2 Follow the drop-down menu **File > Create/Update > Create Symbol Files for Current File** as it's shown as in **Figure 1.4.**

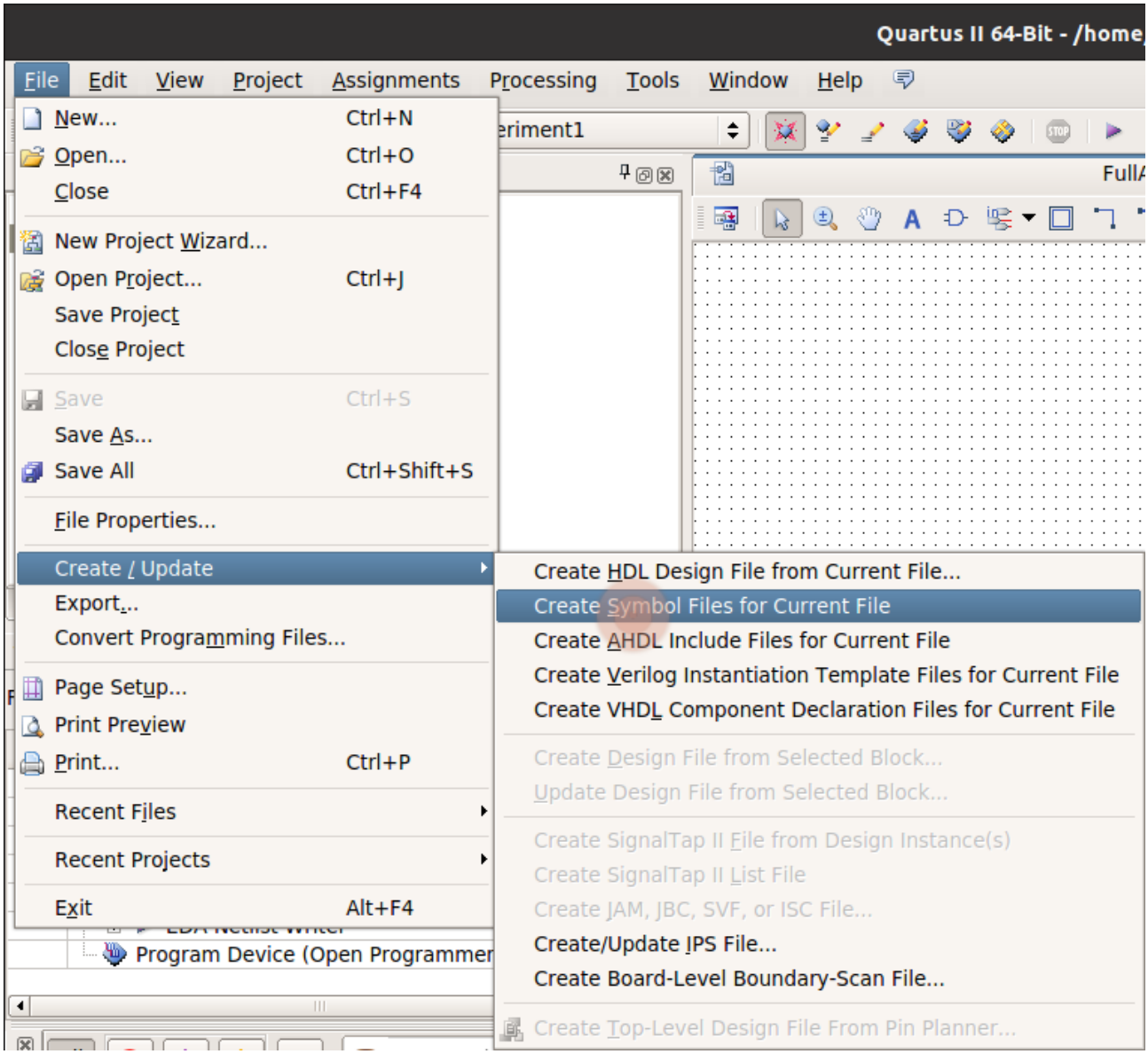


Figure 1.4 Symbol creation window

3 This symbol should have the same file name as the design file ***FullAdder*** but will have a **.bsf** file extension as it's shown on **Figure 1.45** It will be saved in the same project folder. Click **Save,** then click **OK**.

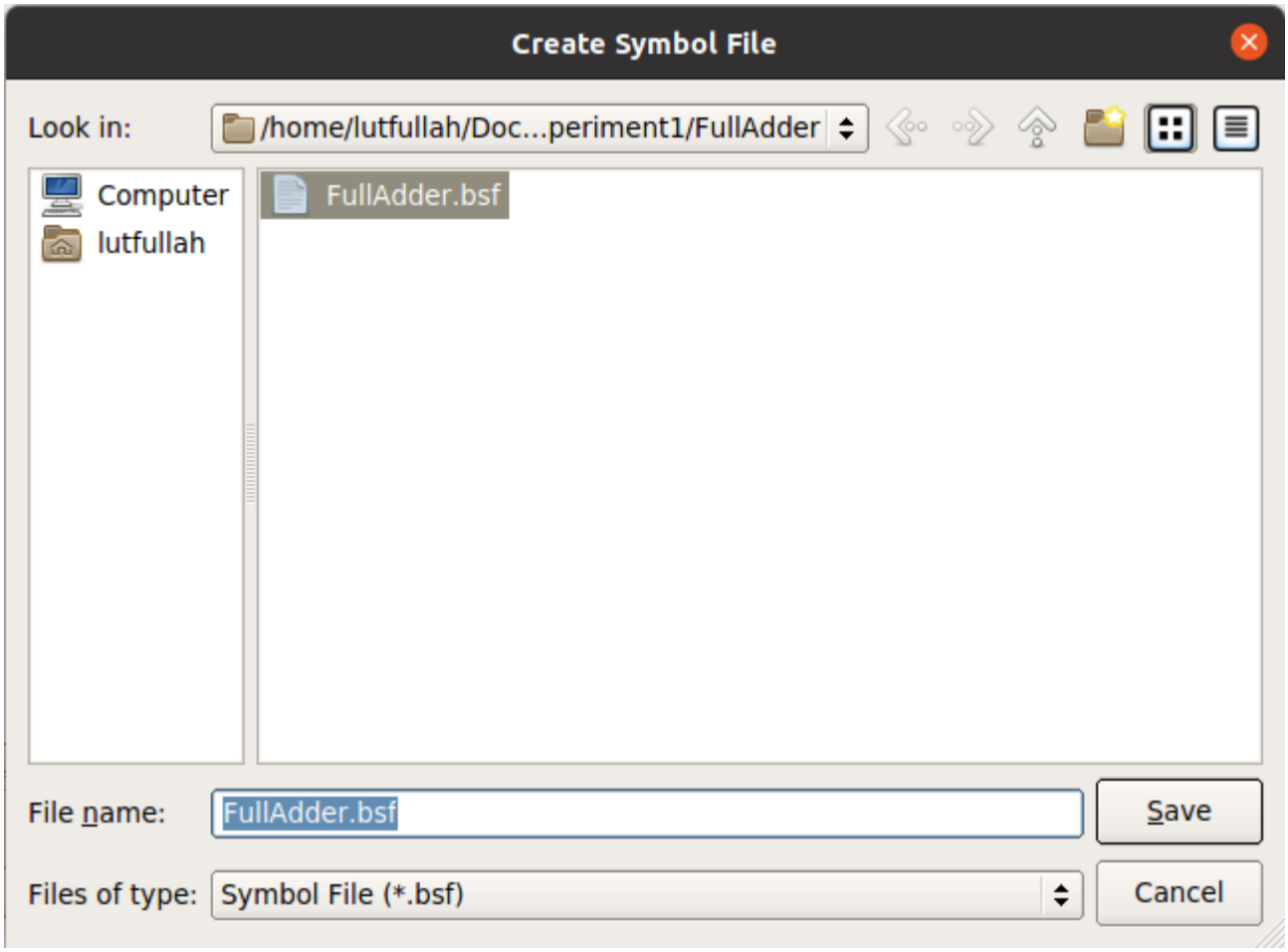


Figure 1.5 Saving newly created symbol

III.B Using Created Symbol in a Higher Level Block(4-bit Adder)

1 Create a new directory in the folder ***Experiment1*** with the name ***FourBitAdder***. Then create a new schematic file and save it as ***FourBitAdder.bdf*** in the the new folder. Do not forget to click **"*Add file to current project*"** option.

2 You will design 4-bit binary adder based on the logic circuit design in your Preliminary Work Part 4. To use the designed full adder, click on **Symbol Tool** and press the button with three dots(**Figure 1.6a**). Select ***FullAdder.bsf*** file from the file browser and press the **Open** button(**Figure 1.6b**). Now, you will see the **FullAdder** as a block with appropriate pins(**Figure 1.6c**). Press **OK** to insert it in the schematic.

3 Finish your design adding as many pins and full adders and other logic blocks. Save the schematic with the name ***FourBitAdder.bdf*** and use **"Set as Top-Level Entry"** for the file and compile the design.

4 Follow **File > Create/Update > Create Symbol Files for Current File** and save the symbol as ***FourBitAdder.bsf*** in the folder of ***FourBitAdder***.

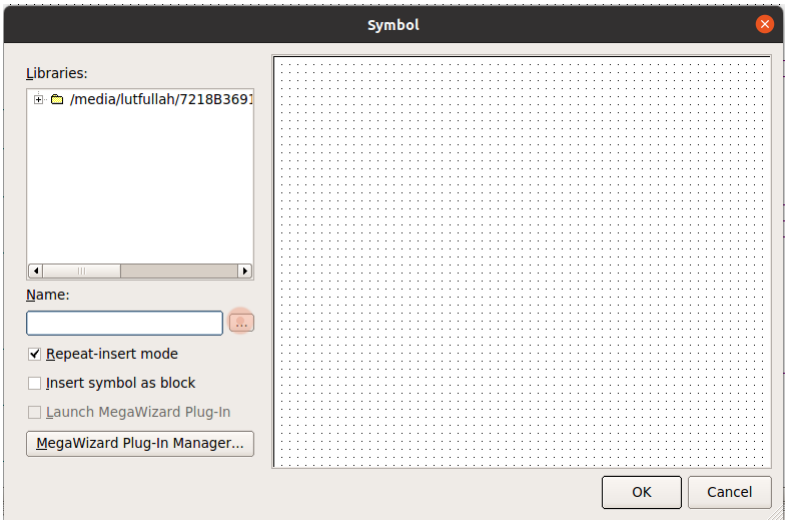


Figure 1.6a Using previously created symbol

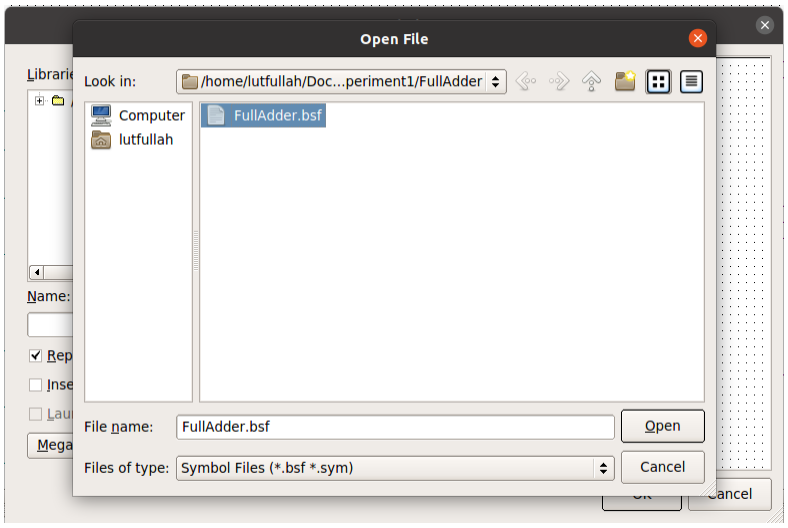


Figure 1.6b Using previously created symbol

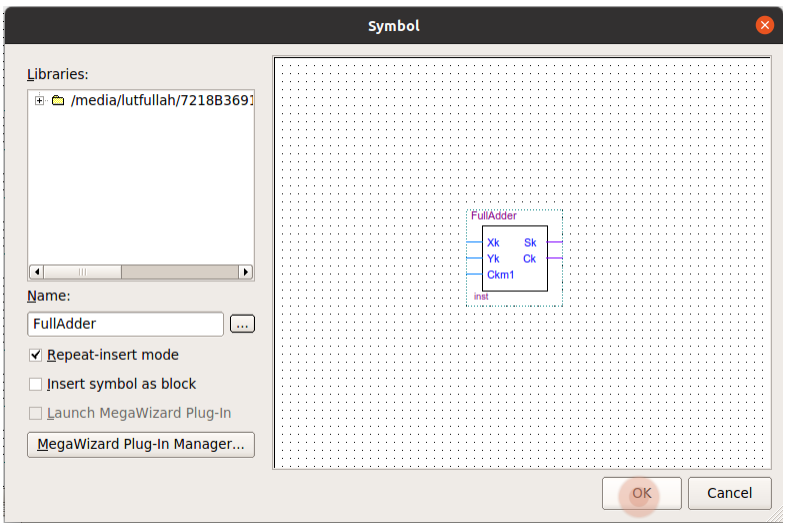


Figure 1.6c Using previously created symbol

III.C Designing the Complementor

1 Create a new directory in the folder ***Experiment1*** with the name **Complementor**. Then, create a new schematic file and save it as ***Complementor.bdf*** in the new folder. Do not forget to click **"*Add file to current project*"** option.

2 Based on the logic circuit design in your preliminary work **Part 5**, create the 4-bit 2's complementor by adding appropriate components, input/output pins and wiring. Assume that you have a 4-bit binary number $\mathbf{X_3X_2X_1X_0}$. This unit must result in $\mathbf{Z_3Z_2Z_1Z_0}$, which is the 2's complement of $\mathbf{X_3X_2X_1X_0}$ when the control bit **E** is **0**, and should give the output as input when **E** is **1**.

3 By using **FourBitAdder** symbol, and other logic gates, at your will, draw the schematic of your **Complementor** design. You may (or will) need a constant "**0**" (zero, low) or a constant "**1**" (one, high) to use as an input. You may use a "**GND"** block for **0** and a "**Vcc"** for **1** which can be found in "**other"** category under the "**primitives"** tab of the **Symbol Box**.

III.C.2 Creating a Bus for Complementor

1 You will need to use buses in the circuit schematic to represent input/output pins. A bus is basically a wire that represents more than one inputs or outputs. In **Quartus II**, a bus is named as, for example, **A[3..0]**, which means the bus is 4-bit, and composed of the signals **A3,A2,A1, and A0** where **A3** is the most significant bit.

2 To create a bus, you should put a regular pin from the pins library of the **Symbol Wizard** box. Add an input pin and name this input as "**X[3..0]**" as it's shown on **Figure 1.7**. You can easily do this by right-clicking the input object, selecting **Properties**, and changing the name.

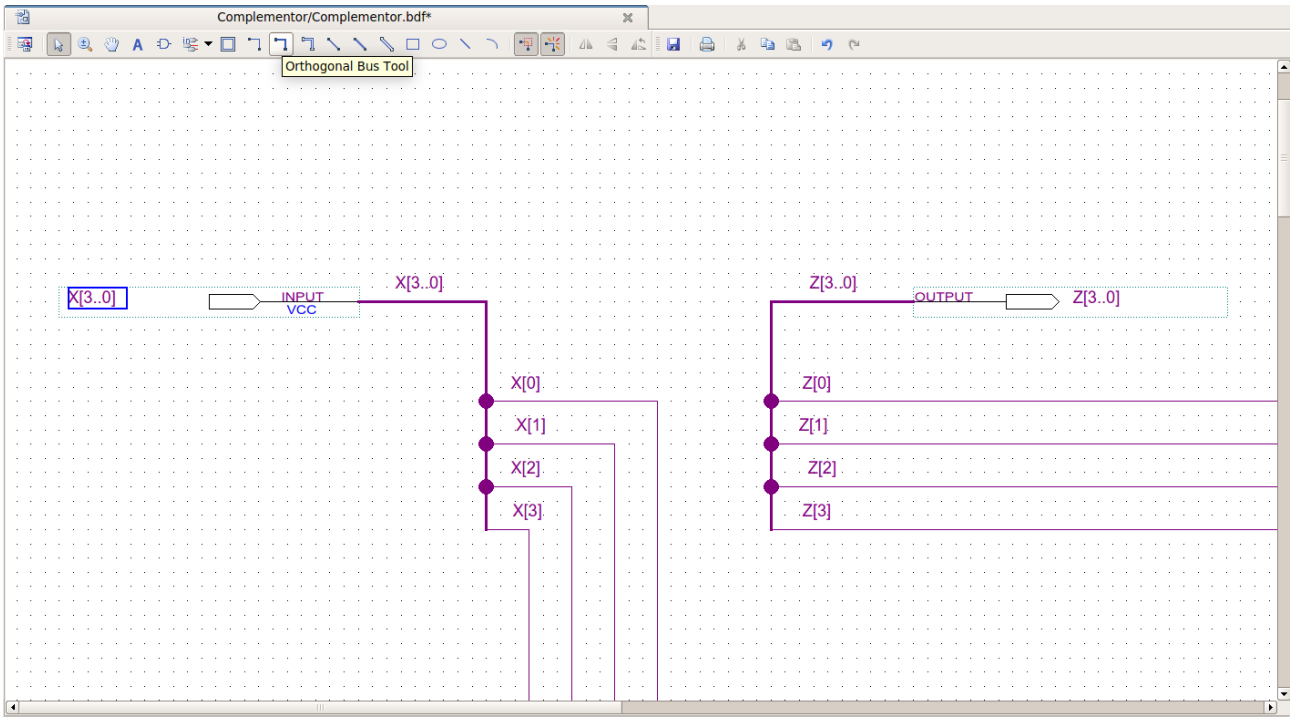


Figure 1.7 Bus creation for the input and output

3 Add a bus to the input shown as bold wire to the input pin **X[3..0]** in **Figure 1.7**. Do this by selecting the **Orthogonal Bus Tool** symbol (like the net symbol but with a thicker line) and drawing the bus line as in the same figure. Name this bus as **X[3..0]** too. To name a bus or a line, select the path with (**Hand Tool**) and start typing the name.

4 Now connect the input bits of your design to the bus, and name the paths to the each input line properly as **X[0]**, **X[1]**, **X[2]** and **X[3]** as it is shown in **Figure 1.7**. Note that these lines are from **Diagonal Node Tool** or **Orthogonal Node Tool** with thinner lines. Also, note that each of these lines will make each of input bits for your design.

5 Apply the same methodology to create the output bus with the name **Z[3..0]**.

6 Your **Complementor** design should lie in between these wires and input **E,** all of which should be connected according to your design.

7 Save the schematic and follow the same procedure as before to compile the ***Complementor.bdf***. Then, save it as a symbol file in the folder ***Complementor***.

III.C.3 Functional Simulation of the Complementor

1 Now you will verify the behavior of the top-level circuit design. Simulation of this project can be a little bit more complicated since there are a total of 5 inputs applied to the top-level circuit, namely there are 32 possible input combinations! Instead of exhaustively testing all possible combinations, you will test the design with the input combinations in **Table 1.5**. You will draw the corresponding test vectors in a **Vector Waveform File** (.vwf).

Table 1.5 Selected input test vectors for **Complementor** design

X[3]	X[2]	X[1]	X[0]	X	E
0	0	0	0	0	0
1	0	1	1	B	1
0	1	1	0	6	1
0	0	1	0	2	0
1	1	1	0	E	1
1	0	0	1	9	0
1	1	1	1	F	0
1	0	0	0	8	0
0	0	1	1	3	1
1	1	0	0	C	0

2 Open the Waveform Editor following **File > New** and selecting **University Program VWF**.

3 Save the file under the name ***Complementor.vwf***. Set the desired simulation to run from **0** to **300 ns** by selecting **Edit > End Time** and entering **300 ns**. By selecting **View > Fit in Window**, display the entire simulation range of **0** to **300 ns** in the window. Divide waveform into 10 pieces by selecting **Edit > Grid Size** and entering **30 ns**.

4 Next, we want to include the input and output nodes of the circuit to be simulated. Follow **Edit > Insert > Insert Node or Bus** to open the dialog box. Click **Node Finder**. Select **Pins:all** and click **List** button. To add bus input and output nodes easily, select the **Input Group** or **Output Group** type variables as shown in **Figure 1.8** instead of selecting every bit separately. After selecting nodes, Click **OK** in this popup window and the previous one as shown in **Figure 1.8**

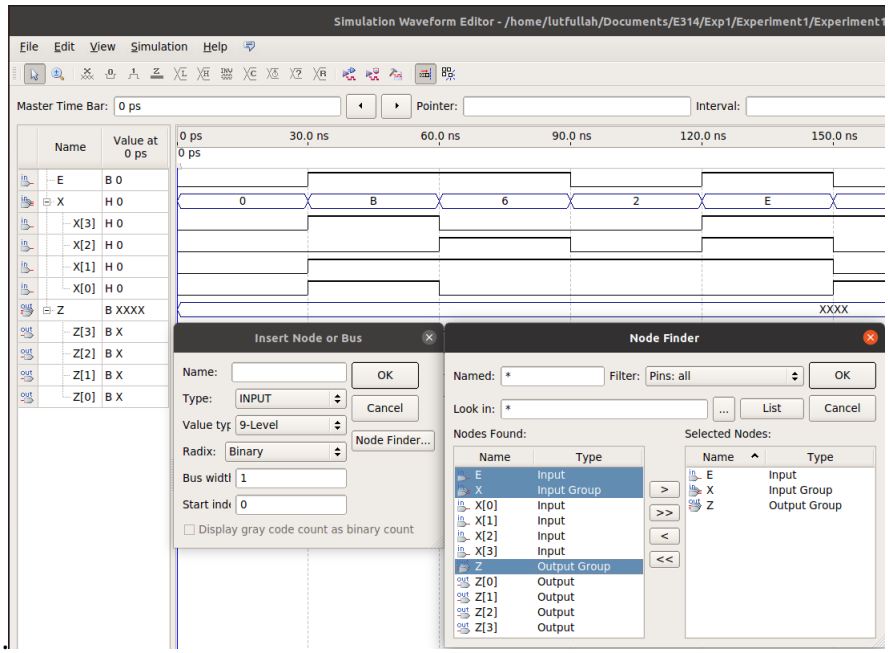


Figure 1.8 Adding buses to the functional simulation setup

5 If you extend your 4-bit input groups by double clicking on them, you see the sub-bits of the group (**Figure 1.9**).

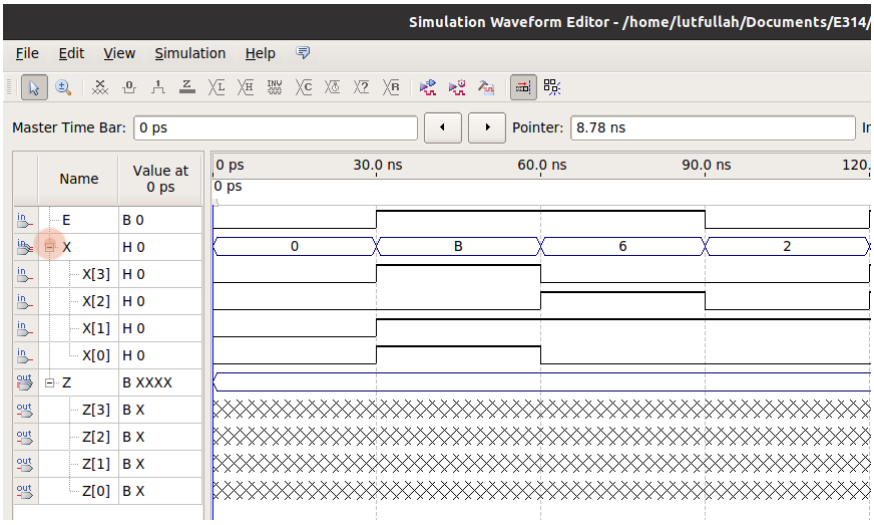
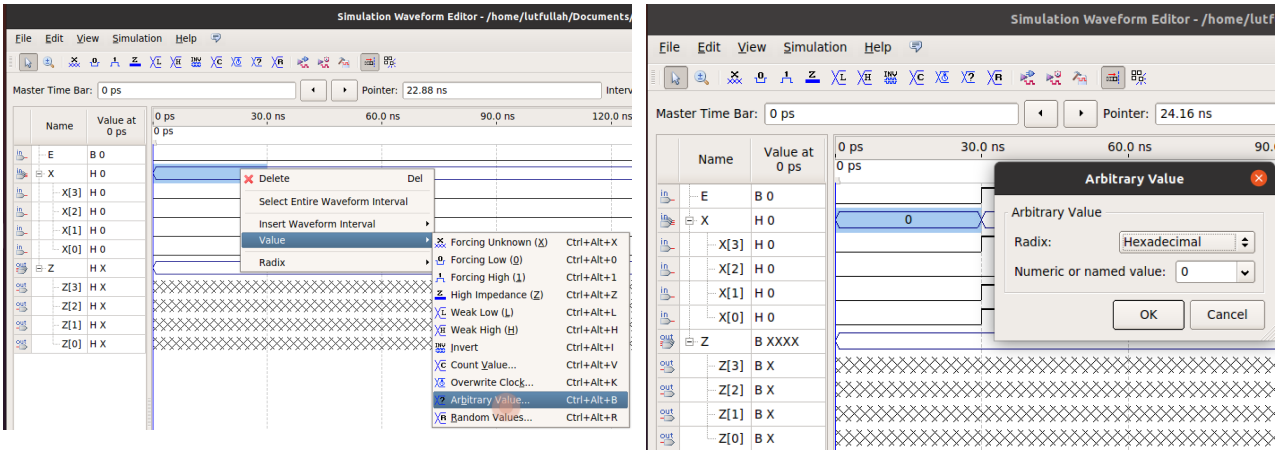


Figure 1.9 One possible way to visualize the inputs/outputs

6 Arrange the waveform by selecting every **30 ns** period

- Right click and select **Value > Arbitrary Value> Radix** (**Figure 1.10a**)
- Choose **Hexadecimal** for 4-bit groups or **Binary** for others and enter the values according to the **Table 1.5**(**Figure 1.10b**)

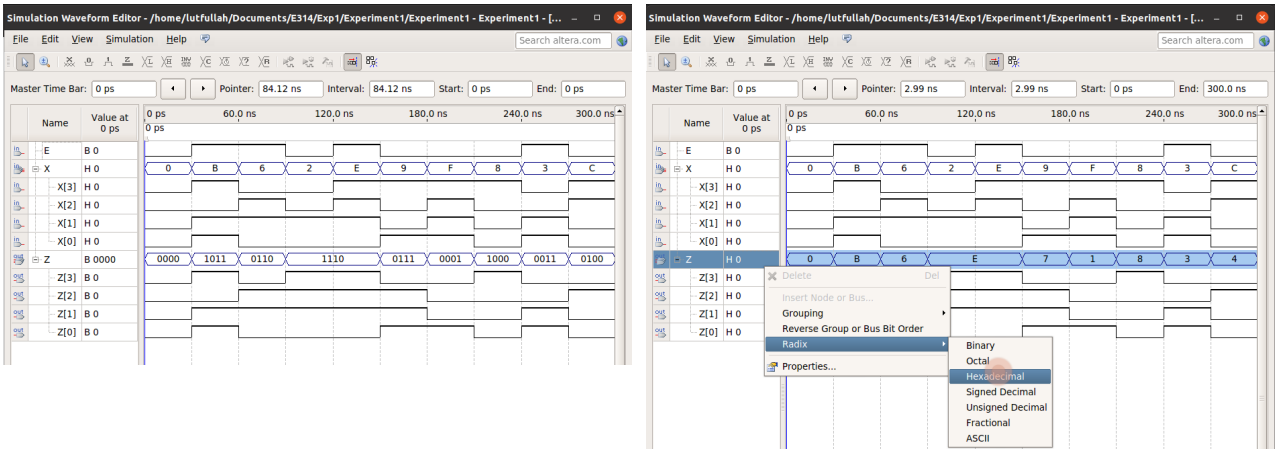


a b

Figure 1.10 Setting Waveforms

7 Save the waveform and **Run Functional Simulation**.

8 You will see the simulation results of your design with the given inputs. Check the output (**Z[3..0]**) for different input combinations to be sure the design works properly. Your functional simulation results will hopefully be like in **Figure 1.11a**.



a b

Figure 1.11 One way to see simulation results

To change the representations of the input and output, you can right click on the values under the name tab situated on the left and select **Radix > Binary** or **Radix > Hexadecimal** or anyone of them as you like(**Figure 1.11b)**.

III.D Designing the Adder/Subtractor

1 Based on your design in preliminary work **Part 6**, implement your adder/subtractor design on a new file ***Experiment1.bdf*** in folder ***Experiement1***. Do not forget to select "***Add file to current project"*** option.

Note: While implementing your adder/subtractor design, you should use busses to represent input and output vectors. After all, your design should include 3 input pins (2 of them are for input busses, and the other is for control input **E**), and 1 output pin for the output bus. For the sake of convenience, use **X**, **Y**, and **E** as input names, and **Z** for output name.

Table 1.6 Selected input test vectors for adder/subtractor design

X[3]	X[2]	X[1]	X[0]	X	Y[3]	Y[2]	Y[1]	Y[0]	Y	E
0	1	1	0	6	0	0	1	1	3	1
0	1	1	1	7	0	0	0	1	1	1
1	1	0	1	D	0	1	0	1	5	0
0	0	1	1	3	0	1	0	0	4	0
1	1	1	0	E	1	1	1	1	F	1
1	0	0	1	9	1	0	0	1	9	1
1	1	1	1	F	1	1	0	0	C	0
1	0	0	0	8	0	0	0	0	0	0

2 Compile your design following the steps in **III.A.1.3** for this file.

3 Simulate your design functionally by using input combinations given in **Table 1.6.** Do not forget to save and compile as you did in previous your designs beforehand.

III.E Final Work

1 Implement 4-bit binary adder using Verilog-HDL you designed in preliminary work

IV IC List

7400 IC Four NAND2, i.e., four two input NAND gates

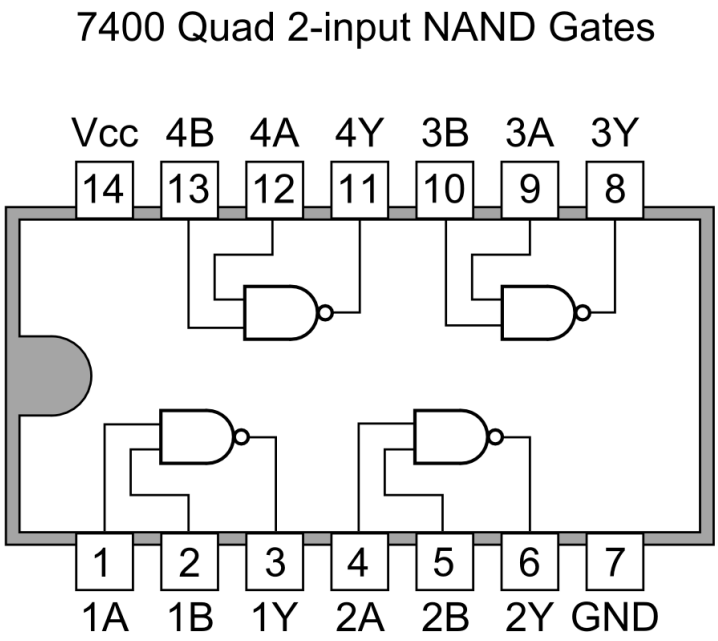


Figure 1.4 7400 IC

7483 IC 4-bit binary adder

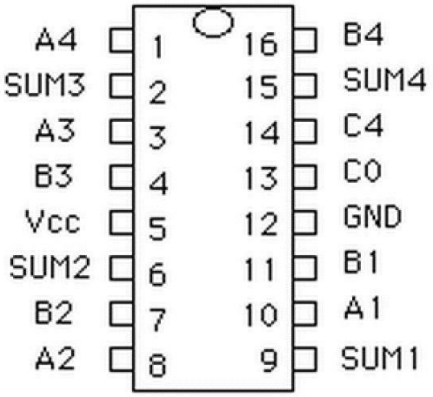


Figure 1.5 7483 IC

7486 IC Four XOR2, i.e., four two-input XOR gates

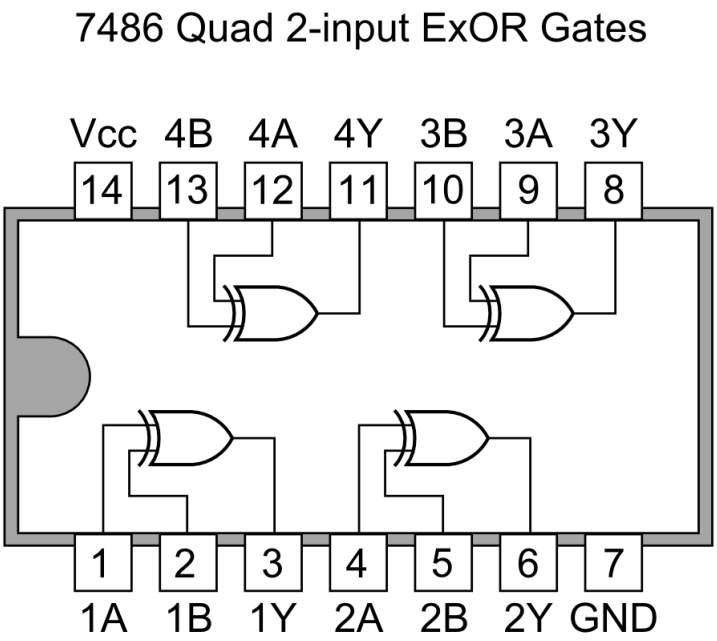


Figure 1.6 7486 IC

V References

[1] G.L. Moss, “Quartus Tutorial 3-Hierarchical Designs, A step-by-step tutorial using Quartus II v9.x.” May 2010.

[2] https://commons.wikimedia.org/wiki/File:7400_Quad_2-input_NAND_Gates.PNG

[3] https://commons.wikimedia.org/wiki/File:7486_Quad_2-input_ExOR_Gates.PNG