EEE- 3104

Digital Electronics

AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT
OF
ELECTRICAL AND ELECTRONIC ENGINEERING

EEE- 3104

Digital Electronics-1 Lab

Edition 2020

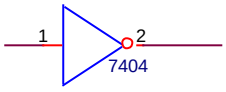
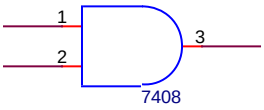
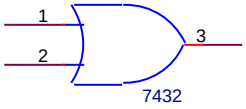
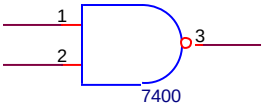
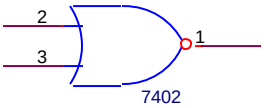
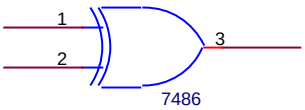
Table of Contents

Experiment: 1
Experiment name: Introduction to different digital ICs.

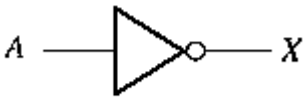
Introduction:

In this experiment you will be introduced to different digital ICs that will be used in this digital lab to perform different functions and also the function of each IC. You are asked to memorize the followings associated with each IC.

1. IC number
2. IC name
3. Total number of pins
4. V_{cc} pin number
5. Ground pin number

IC number	IC name	Schematic view
7404	NOT/INVERTER	
7408	AND	
7432	OR	
7400	NAND	
7402	NOR	
7486	XOR	

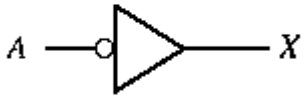
The INVERTER/NOT Gate



A	X
0	1
1	0

$X = \overline{A}$

Boolean expression



Distinctive shape symbols

Truth table

0 = LOW
1 = HIGH

The output of an inverter is always the complement (opposite) of the input.

The AND Gate

Distinctive shape symbol

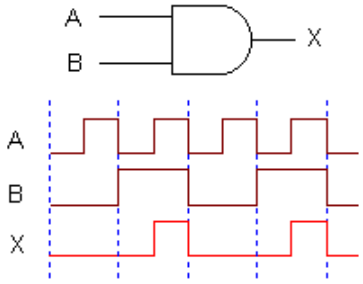
X = AB

Boolean expression

B	A	X
0	0	0
0	1	0
1	0	0
1	1	1

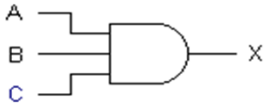
Truth table

0 = LOW
1 = HIGH



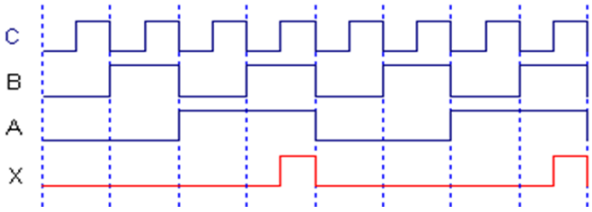
Pulsed Waveforms

The output of an AND gate is HIGH only when all inputs are HIGH.



X = ABC

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



3 Input AND Gate

The OR Gate

Distinctive shape symbol

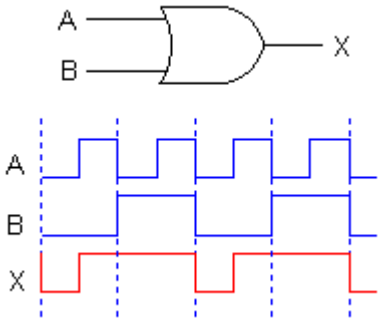
$X = A + B$

Boolean expression

B	A	X
0	0	0
0	1	1
1	0	1
1	1	1

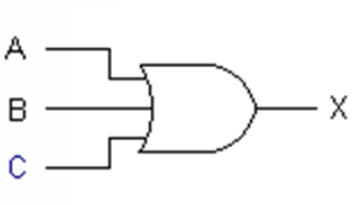
Truth table

0 = LOW
1 = HIGH



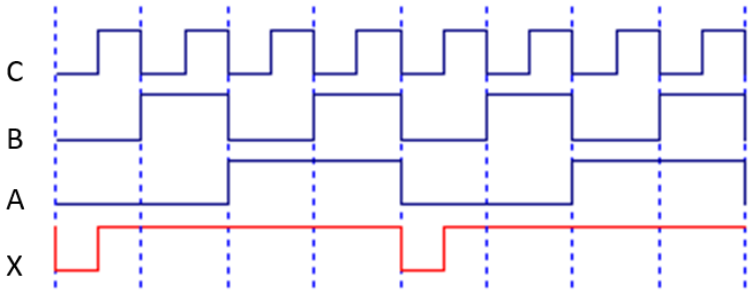
Pulsed Waveforms

The output of an OR gate is HIGH whenever one or more inputs are HIGH.



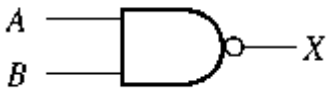
$X = A+B+C$

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

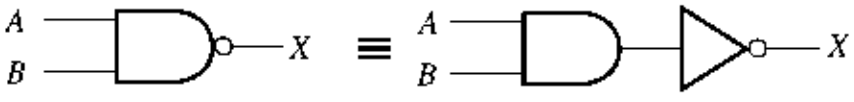


3 Input OR Gate

The NAND Gate



Distinctive shape symbol



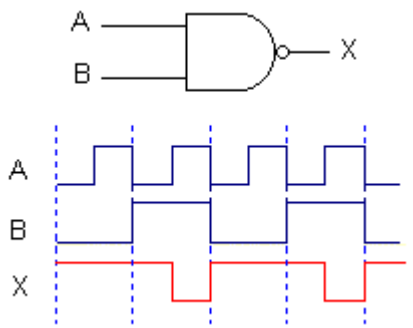
B	A	X
0	0	1
0	1	1
1	0	1
1	1	0

Truth table

0 = LOW
1 = HIGH

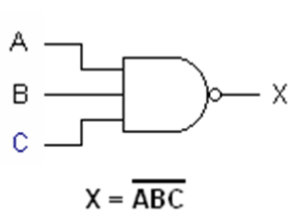
$X = \overline{AB}$

Boolean expression

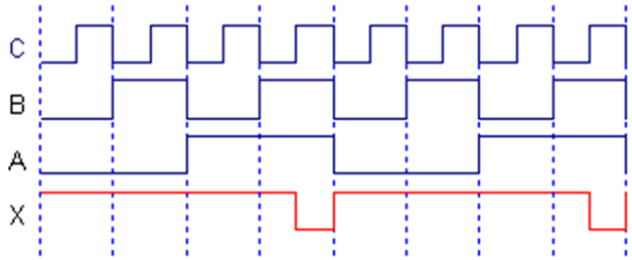


Pulsed Waveforms

The output of a NAND gate is HIGH whenever one or more inputs are LOW.

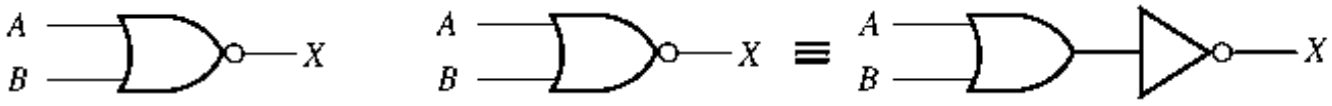


A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



3 Input NAND Gate

The NOR Gate



Distinctive shape symbol

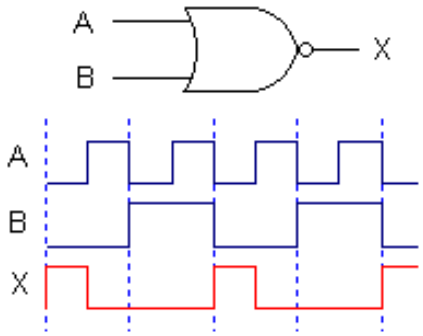
B	A	X
0	0	1
0	1	0
1	0	0
1	1	0

Truth table

0 = LOW
1 = HIGH

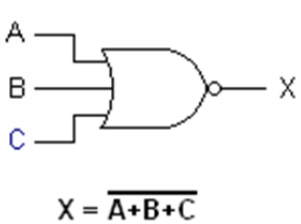
$X = \overline{A + B}$

Boolean expression



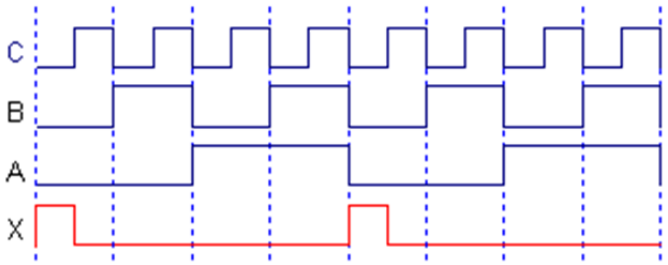
Pulsed Waveforms

The output of a NOR gate is LOW whenever one or more inputs are HIGH.



$X = \overline{A+B+C}$

A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



3 Input NOR Gate

Exclusive-OR Gate

Distinctive shape symbol

B	A	X
0	0	0
0	1	1
1	0	1
1	1	0

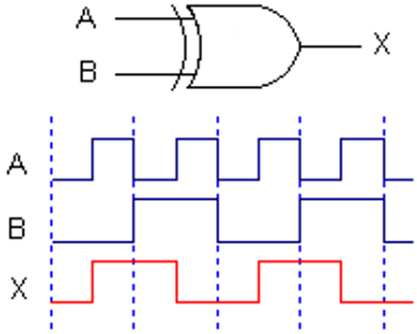
Truth table

0 = LOW
1 = HIGH

$X = A \oplus B$

Boolean expression

The output of an XOR gate is HIGH whenever the two inputs are different.



Pulsed Waveforms

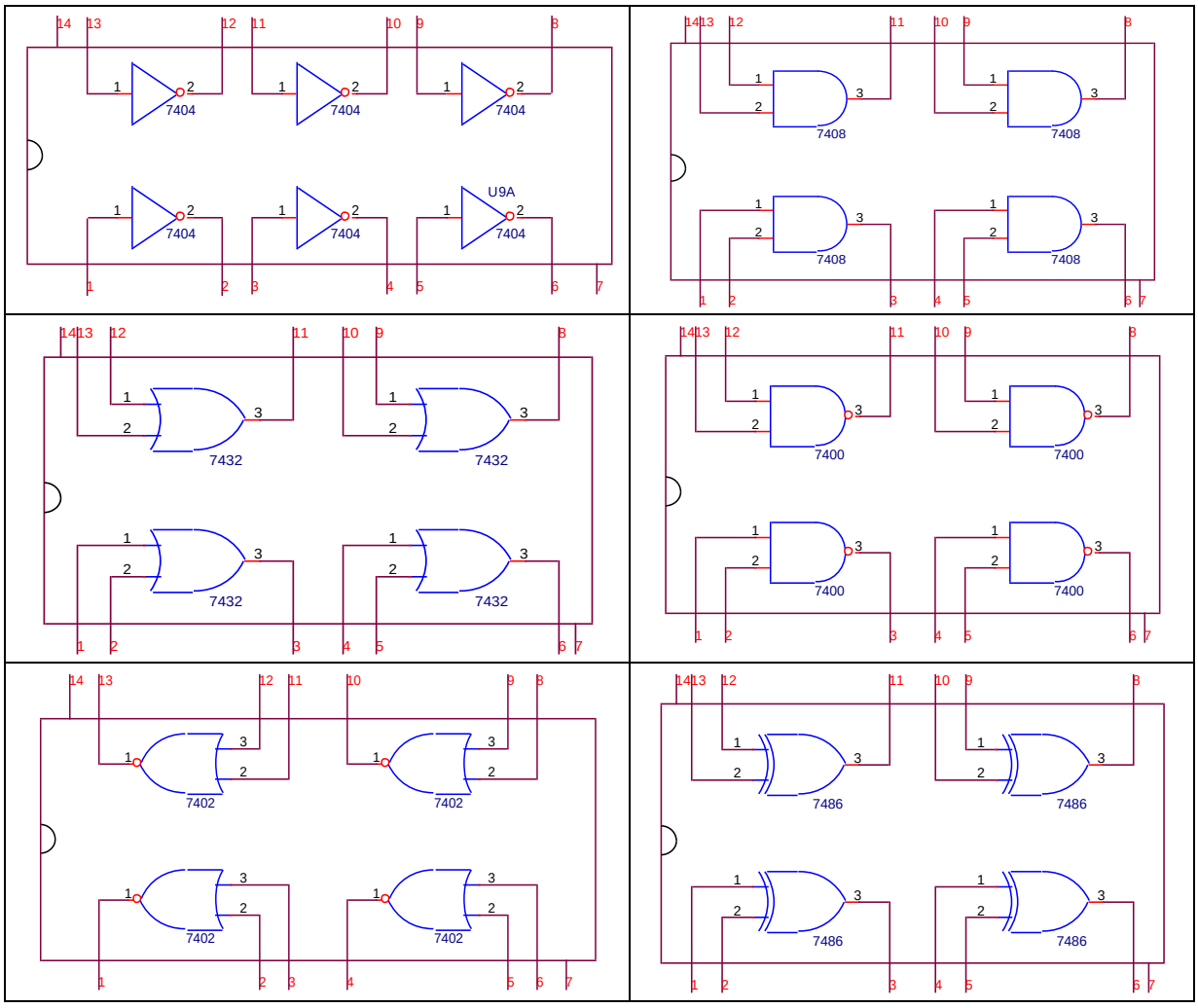
Equipment:

1. Trainer Board
2. IC 7400,7402,7404,7408,7432,7486
3. Microprocessor Data handbook

Procedure:

1. Take any of the following ICs. From microprocessor data handbook find the name of the IC, total number of pins that it has, V_{cc} pin and ground pin.

IC Number	IC name	Total number of pin	V_{cc} pin no.	Ground pin no.
7400	NAND	14	14	7
7402	NOR	14	14	7
7404	NOT	14	14	7
7408	AND	14	14	7
7432	OR	14	14	7
7486	XOR	14	14	7



2. Note the number of gates each IC has from the handbook.
3. Now fill up the following table:

Input A	**Input B**	**7400 NOT** $Y=\overline{A}$	**7432 OR** $Y=A+B$	**7402 NOR** $Y=\overline{A+B}$	**7486 XOR** $Y=A\oplus B$	**7408 AND** $Y=AB$	**7400 NAND** $Y=\overline{AB}$
0	0						
0	1						
1	0						
1	1						

4. Now verify the observed output with the desired output for different combination of inputs.
5. Repeat step 1 to 4 for different ICs.

Report:

1. How can you make a three input AND/OR/XOR gate with a two input AND/OR/XOR gate?
2. Is it possible to make a three input NAND/NOR gate with a two input NAND/NOR gate? Justify your answer.

Experiment: 2
Experiment name: Introduction to Combinational logic and K map minimization.

Introduction:
Logic design basically means the construction of appropriate function, presented in Boolean algebraic form, then edification of the logic diagram, and finally choosing of available ICs and implementing the IC connection so that the logic intended is achieved. The efficiency in simplifying the algebra leads to less complicated logic diagram, which in the end leads to easier IC selection and easier circuit implementation.

Caution:
1. Remember to properly identify the pin numbers so that no accidents occur.
2. Properly bias the ICs appropriate voltages to appropriate pins.

Equipment:
1. Trainer Board
2. IC 7400,7402,7404,7408,7432,7486
3. Microprocessor Data handbook

Job 1:

Implement of function $f = AB + BC' + CA$

$$= ABC + ABC' + ABC' + A'BC' + ABC + AB'C$$
$$= ABC + ABC' + A'BC' + AB'C$$
$$= m_7 + m_6 + m_2 + m_5$$
$$= \sum m(2,5,6,7)$$

Truth Table

Row no.	Input			Output
	A	B	C	*f*
0	0	0	0	0
1	0	0	1	0
2	0	1	0	**1**
3	0	1	1	0
4	1	0	0	0
5	1	0	1	**1**
6	1	1	0	**1**
7	1	1	1	**1**

K-Map

A \ BC	00	01	11	10
0	0	0	0	**1**
1	0	**1**	**1**	**1**

AC **BC'**

SOP: $F = AC + BC'$

POS: $F' = B'C' + A'C$

$\Rightarrow F = (B'C' + A'C)'$

$\Rightarrow F = (B + C)(A + C')$

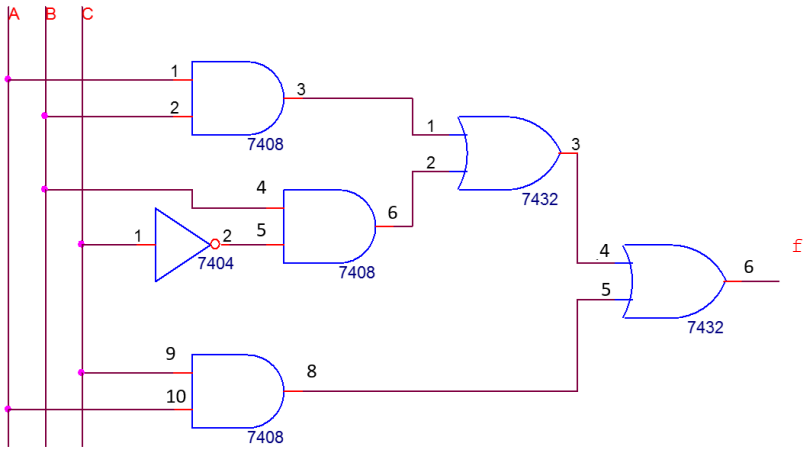


Figure 2.1: Logic Diagram of Job 1

Procedure:

1. Draw logic diagram to implement the function.
2. Select ICs from the equipment list.
3. Note the output logic for all combination of inputs.
4. Repeat step-1, 2 and 3 for SOP and POS function.

Job 2:

Implement of function $f = (AB + B)(C + A)(AC + B)$

Now,

$$
\begin{aligned}
f &= (AB + B)(C + A)(AC + B) \\
&= B(A+1)(A + B)(A + C)(A + B)(B + C) \\
&= B(A + B)(A + C)(B + C) \\
&= (B + AA')(A + B)(A + C)(B + C) \\
&= (A + B)(A' + B)(A + B)(A + C)(B + C) \\
&= (A + B)(A' + B)(A + C)(B + C) \\
&= (A + B + CC')(A' + B + CC')(A + C + BB')(B + C + AA') \\
&= (A + B + C)(A + B + C')(A' + B + C)(A' + B + C')(A + B + C)(A + B' + C)(A + B + C)(A' + B + C) \\
&= (A + B + C)(A + B + C')(A' + B + C)(A' + B + C')(A + B' + C) \\
&= M_0 M_1 M_4 M_2 M_5 \\
&= \prod M\ (0,1,2,4,5)
\end{aligned}
$$

Distributive Law

$x + yz = (x + y)(x + z)$

Truth Table

Row no.	Input			Output
	A	B	C	*f*
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

K-MAP

A \ BC	00	01	11	10
0	0	0	**1**	0
1	0	0	**1**	**1**

BC **AB**

SOP: $F = AB + BC$

POS: $F' = B' + A'C'$

$\Rightarrow F = B(A + C)$

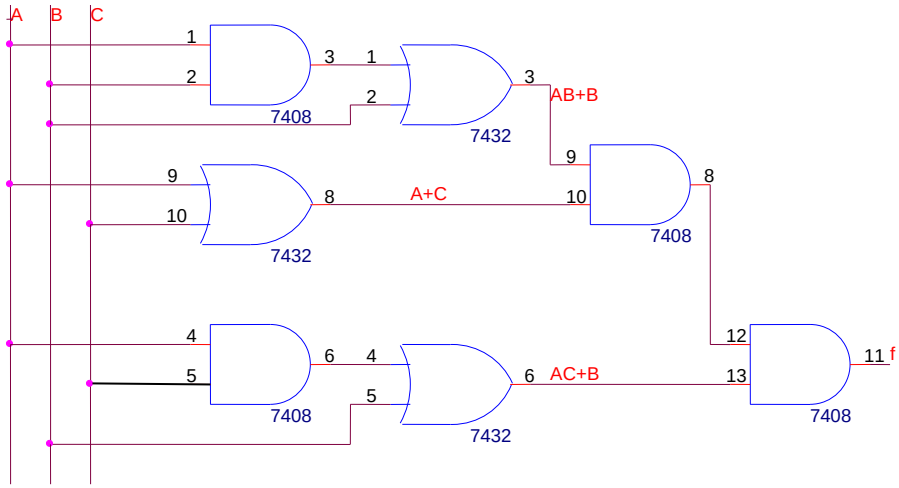


Figure 2.2: Logic Diagram of Job 2

Procedure:

1. Simplify the function in POS form and in SOP form by using Boolean algebra.
2. Draw logic diagram to implement the function.
3. Select ICs from the equipment list.
4. Note the output logic for all combination of inputs.

Experiment: 3
Experiment name: Construction of adders, sub tractors, using basic logic gates.

Introduction:
Adders and sub tractors are the basic operational units of simple digital arithmetic operations. In this experiment, the students will construct the basic adder and sub tractor circuit with common logic gates and test their operability. Then in the last job, they will cascade adder ICs to get higher bit adders.

Binary Adder
Among the basic functions encountered are the various arithmetic operations. The most basic arithmetic operation, is the addition of two binary digits. This simple addition consists of four possible elementary operations, namely, 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10. The first three operations produce a sum whose length is one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a *carry.* When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher-order pair of significant bits. A combinational circuit that performs the addition of two bits is called a *half-adder.* One that performs the addition of three bits (two significant bits and a previous carry) is *full-adder.*

Half Adder
From the basic understanding of a half-adder, we find that the circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. It is necessary to specify two output variables because the result may consist of two binary digits. We arbitrarily assign symbols *x* and *y* to the two inputs and S (for sum) and C (for carry) to the outputs.
Now that we have established the number and names of the input and output variables, we are ready to formulate a truth table to identify exactly the function of the half-adder. This truth table is

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The carry output is 0 unless both inputs are 1. The S output represents the least significant bit of the sum.
The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum of products expressions are

$$S = x'y + xy' = x \oplus y$$

$$C = xy$$

The logic diagram for this implementation is shown below

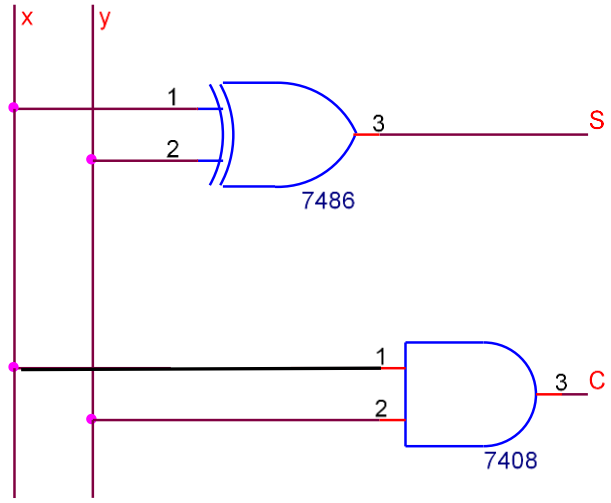


Fig 3.1. Half-adder

Full Adder

A full-adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the input variables, denoted by *x* and *y*, represent the two significant bits to be added. The third input, z, represents the carry from the previous lower significant position. Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary 2 or 3 needs two digits. The two outputs are designated by the symbols S for sum and C for carry. The binary variable S gives the value of the least significant bit of the sum. The binary variable Cr gives the output carry. The truth table of the full-adder is

x	y	z	C	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

The eight rows under the input variables designate all possible combinations of 1's and 0's that these variables may have. The 1's and 0's for the output variables are determined from the arithmetic sum of the input bits. When all input bits are 0's, the output is 0. The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1. Physically, the binary signals of the input wires are considered binary digits added arithmetically to form a two-digit sum at the output wires. On the other hand, the same binary values are considered variables of Boolean functions when expressed in the truth table or when the circuit is implemented with logic gates. It is important to realize that two different interpretations are given to the values of the bits encountered in this circuit. The input-output logical relationship of the full-adder circuit may be expressed in two

Boolean functions, one for each output variable. This implementation uses the following Boolean expressions:

$$S = x'y'z + x'yz' + xy'z' + xyz = z'(x'y + xy') + z(x'y' + xy) = z'(x \oplus y) + z(x \oplus y)' = x \oplus y \oplus z$$
$$C = x'yz + xy'z + xyz' + xyz = x(y'z + yz') + yz(x + x') = x(y \oplus z) + yz$$

The logic diagram for the full-adder implemented in sum of products is shown in Fig. 2.2

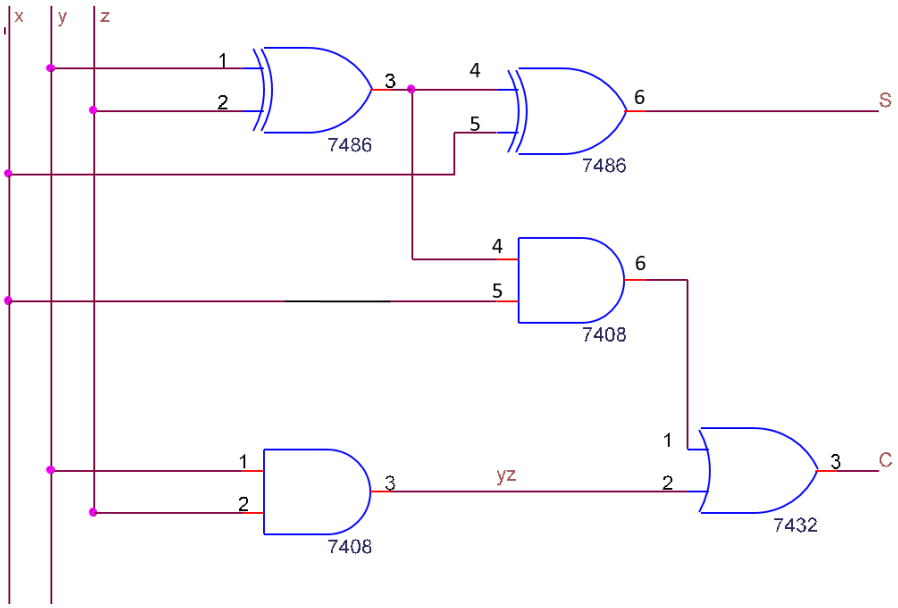


Fig 3.2. Full-adder

Half Subtractor

A half-subtractor is a combinational circuit that subtracts two bits and produces their difference. It also has an output to specify if a 1 has been borrowed. Designate the minuend bit by *X* and the subtrahend bit by *y*. To perform *x* - *y*, we have to check the relative magnitudes of *x* and *y*. If *x* '3 *y*, we have three possibilities: 0 - 0 = 0,
1 - 0 = 1, and 1 - 1 = O. The result is called the *difference bit.* If *x* < *y*, we have 0 - 1, and it is necessary to borrow a 1 from the next higher stage. The 1 borrowed from the next higher stage adds 2 to the minuend bit, just as in the decimal system a borrow adds 10 to a minuend digit. With the minuend equal to 2, the difference becomes
2 - 1 = 1. The half-subtractor needs two outputs. One output generates the difference and will be designated by the symbol *D.* The second output, designated *B* for borrow, generates the binary signal that informs the next stage that a 1 has been borrowed.
The truth table for the input-output relationships of a half-subtractor can now be derived as follows:

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

The Boolean functions for the two outputs of the half-subtractor are derived directly from the truth table:

$$D = x'y + xy' = x \oplus y$$

$$B = x'y$$

It is interesting to note that the logic for D is exactly the same as the logic for output S in the half-adder.

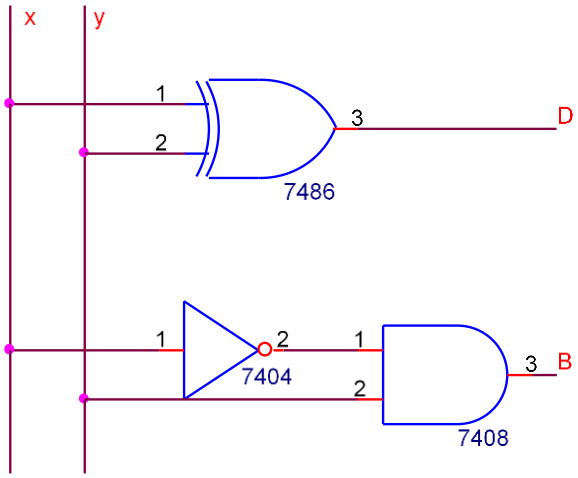


Fig 3.3. Half-subtractor

Full Subtractor

A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage. This circuit has three inputs and two outputs. The three inputs, *x*, *y*, and z, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, *D* and *B*, represent the difference and output borrow, respectively. The truth table for the circuit is

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$D = x'y'z + x'yz' + xy'z' + xyz = x'(y'z + yz') + x(y'z' + yz) = x'(y \oplus z) + x(y \oplus z)' = x \oplus y \oplus z$$

$$B = x'y'z + x'yz' + x'yz + xyz = x'(y'z + yz') + yz(x' + x) = x'(y \oplus z) + yz$$

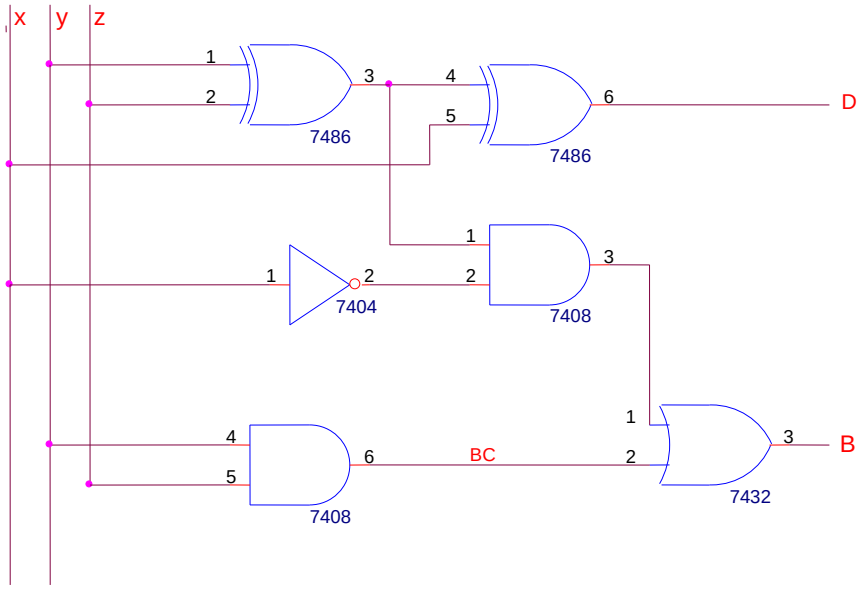


Fig 3.4. Full-subtractor

Caution:

1. Remember to properly identify the pin numbers so that no accidents occur.
2. Properly bias the ICs with appropriate pins.

Equipment:

1. Trainer Board
2. IC 7400,7402,7404,7408,7432,7486
3. Microprocessor Data handbook

Procedure:

1. Fill up the truth table for a half adder
2. Verify the Boolean function for a half adder.
3. Construct the logic diagram from the Boolean functions.
4. Select the ICs from the equipment list.
5. Implement the output logic.
6. Repeat the whole procedure for half a subtractor.
7. Fill up the truth table for a full adder.
8. Verify the Boolean function for a full adder.
9. Construct the logic diagram from the Boolean functions.
10. Select the ICs from the equipment list.
11. Implement the output logic.
12. Repeat the whole procedure for a full sub tractor.

Report

1. Design a full adder using two half adder block and basic gates.

Software: Quartus
Experiment name: Introduction to FPGA

Equipment:

Altera DE 2 FPGA Development Board

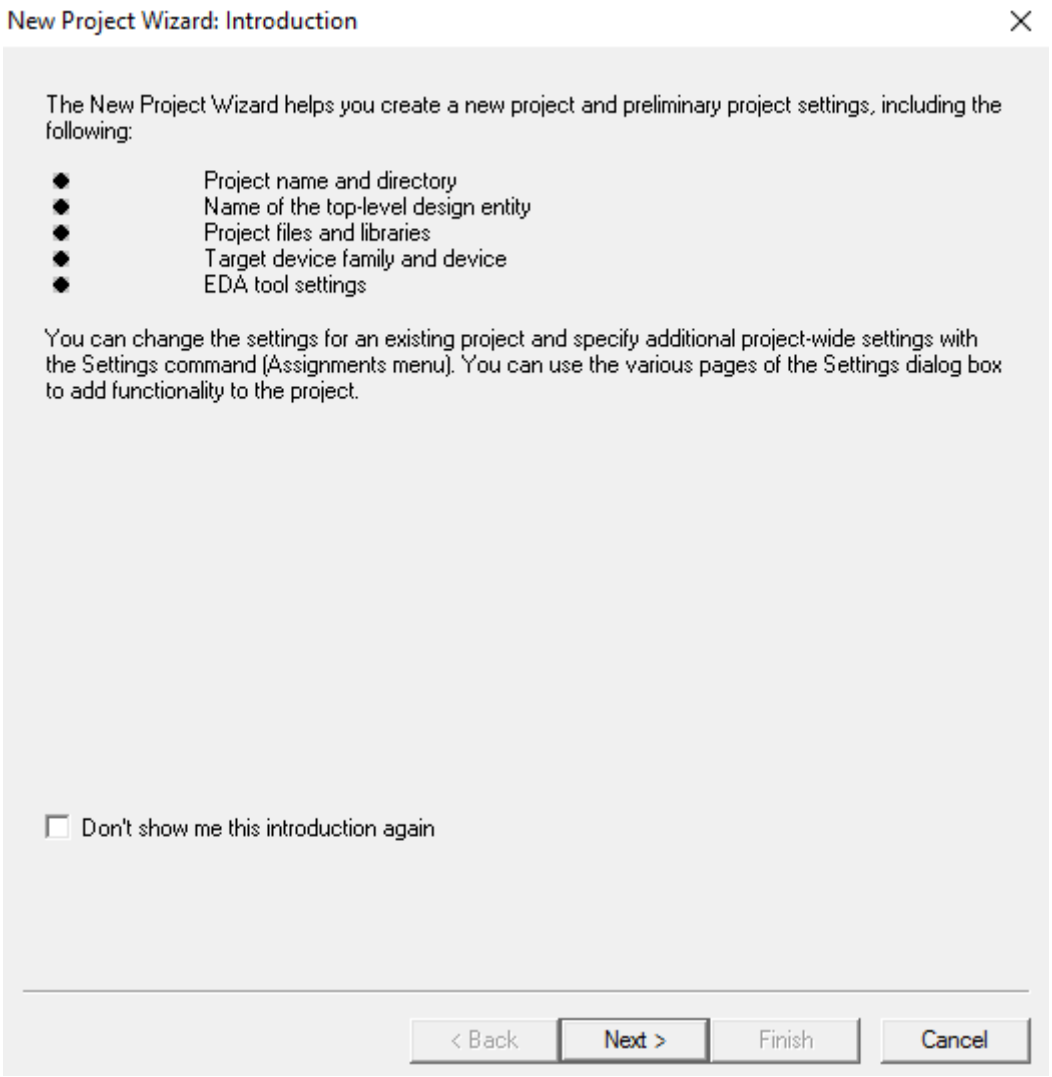
Procedure:

For installation procedure, go to Annexure I.

Creating a New Project in Quartus II (Steps 1-6):

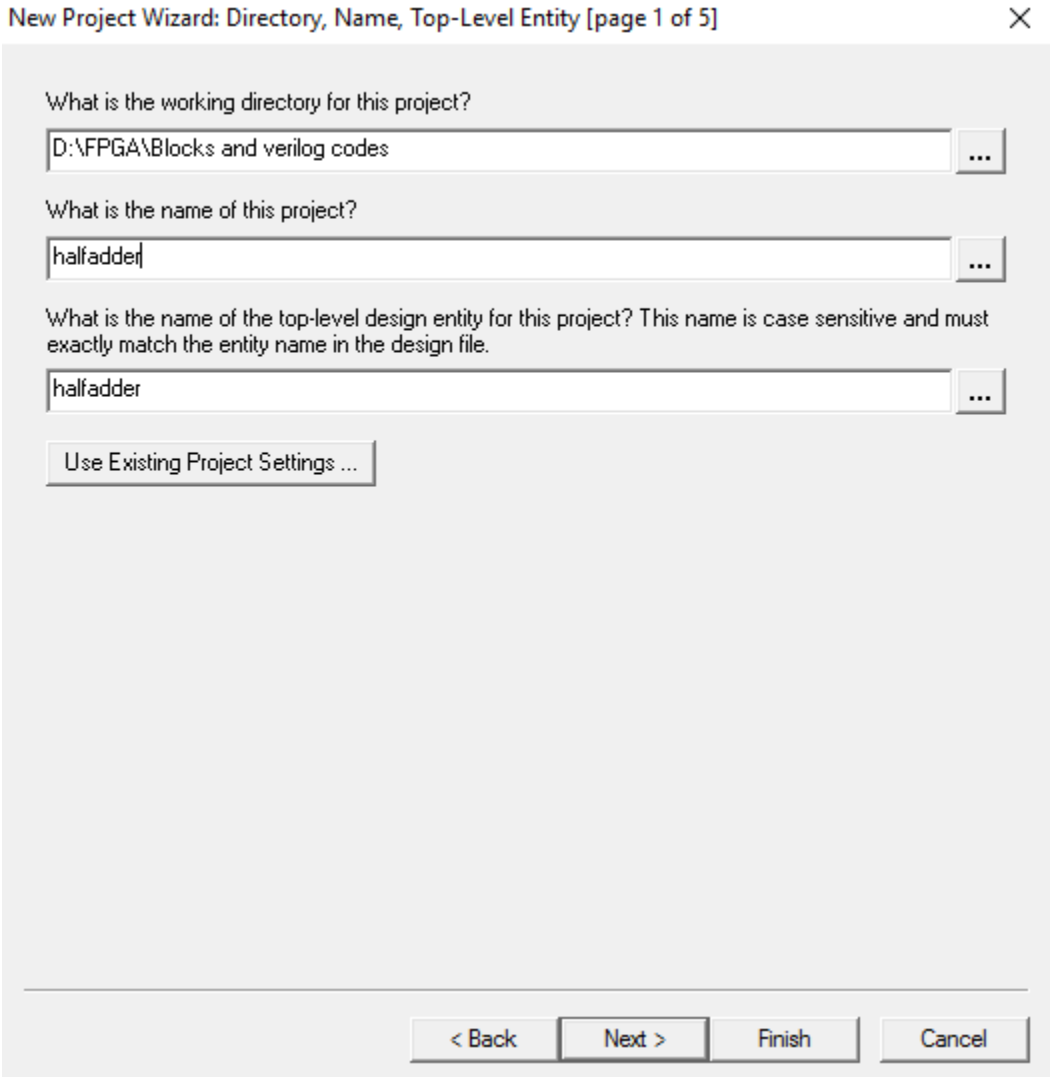
1. Open **Quartus II 9.0 sp1 web edition**.

2. Go to ***File→New Project Wizard.*** The following window will open up:



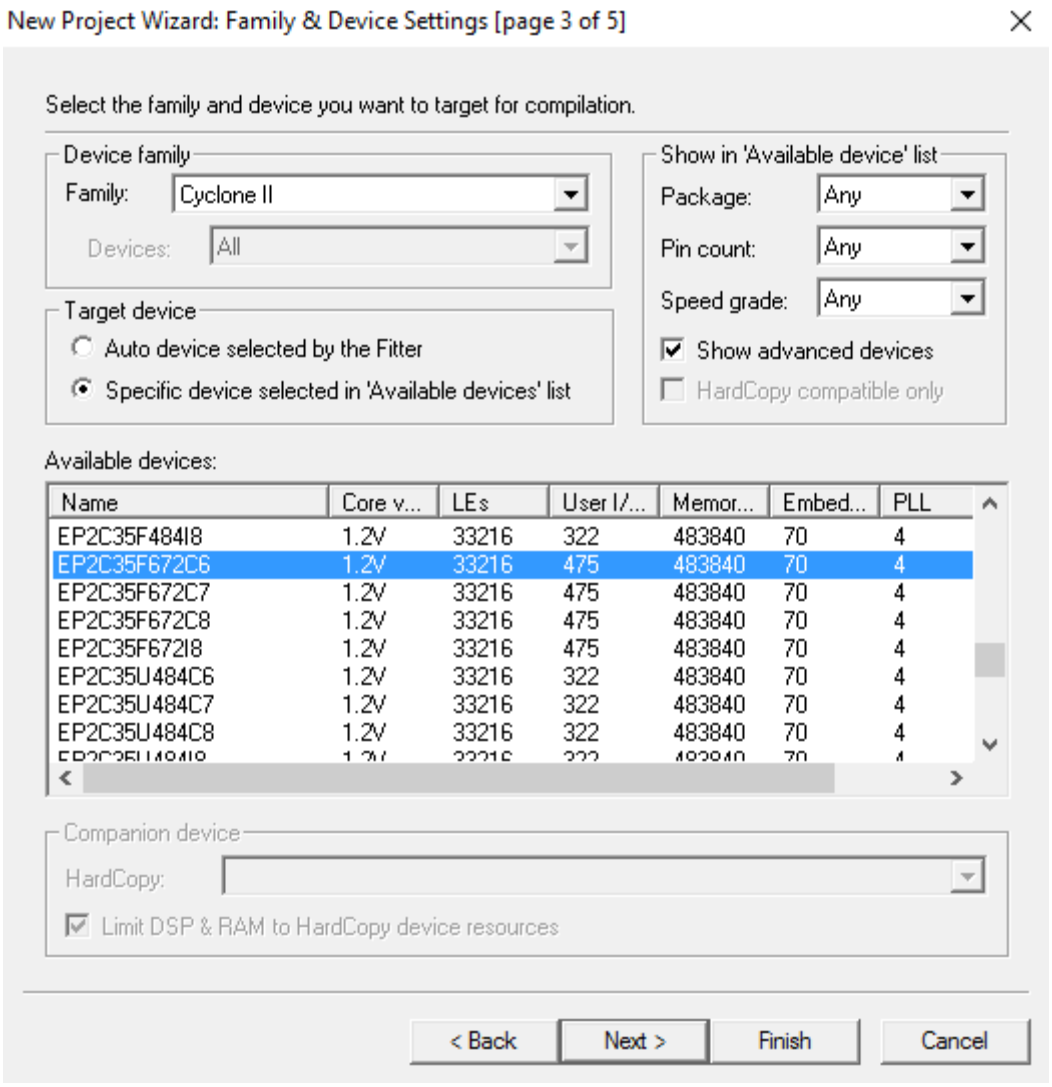
3. Click **Next**.

4. In the next window that appears, change the default working directory to your working directory (e.g. E:\130205001) and give a name to this project (**space in the name is not allowed**). **Your top-level block diagram file, vector waveform files, etc. should have the same name as the name of the project.**



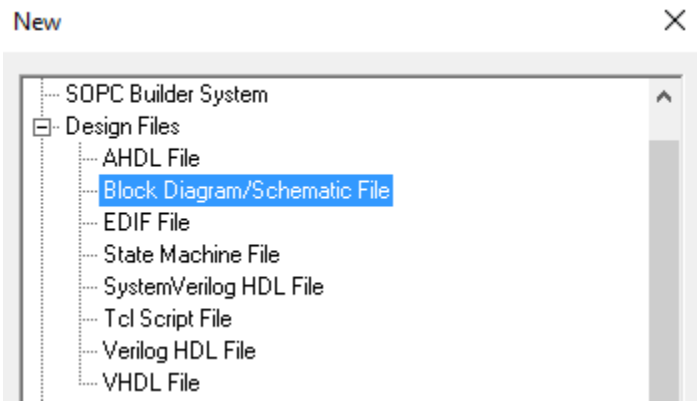
5. In the next window, you may include files to your project, which we will demonstrate later, for now, click **Next**.

6. In the following window, select **Cyclone II** under **Device family** and **EP2C35F672C6**under **Available devices**. Click **Finish**. This completes the steps for creating a project file.

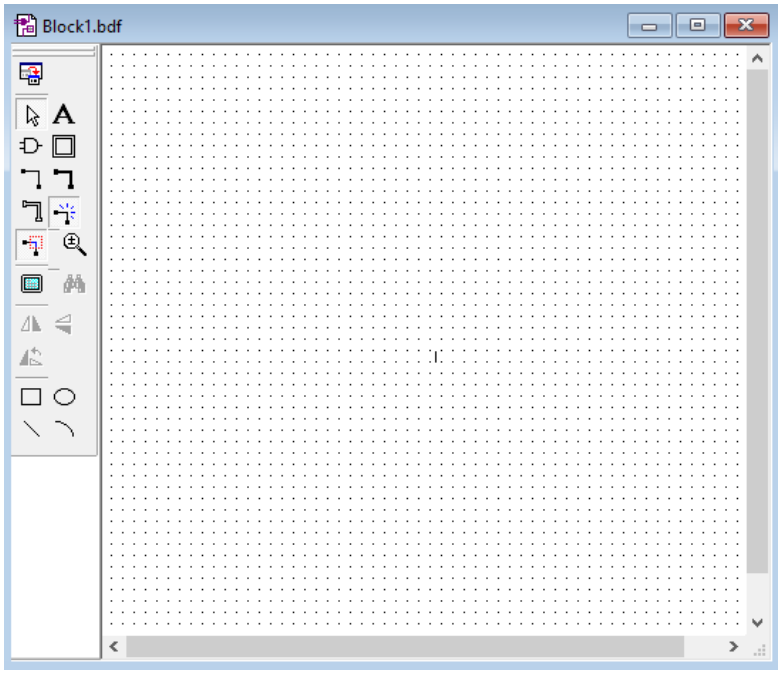


Creating a Block Diagram/Schematic file in Quartus II:

7. Go to ***File→New***. Select **Block Diagram/Schematic File** and click **OK**.

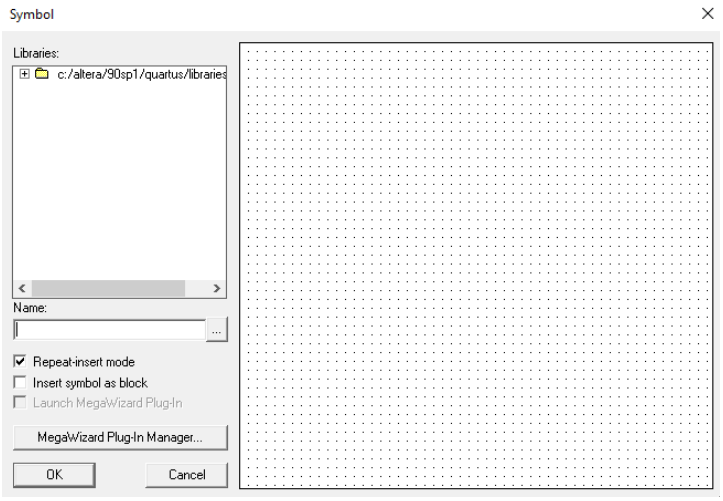


A blank block diagram window will appear.

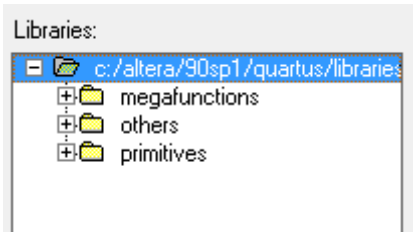


8. To implement half-adder, we will need an AND gate and an XOR gate. From the left menu bar, click on icon for **Symbol Tool**, or alternatively double clock on the blank schematic window.

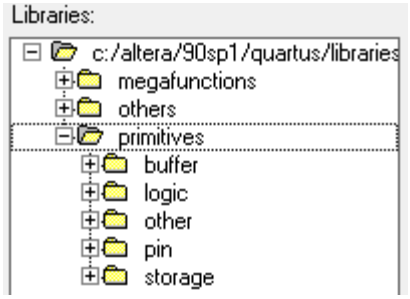
9. The following window will appear. Under **Libraries**, click on the plus icon beside **c:/altera/…**



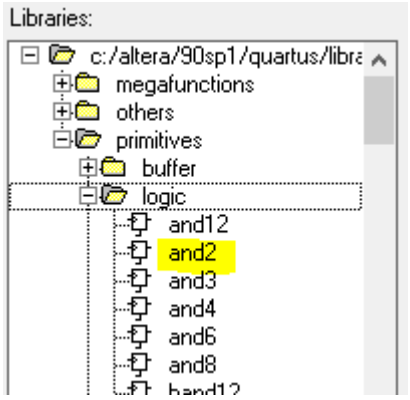
10. After expanding the plus icon, you will see the following library directories:



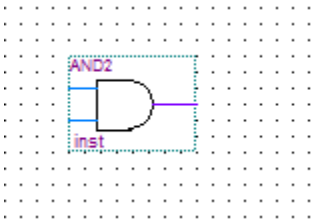
11. Click on the plus sign beside **primitives**.



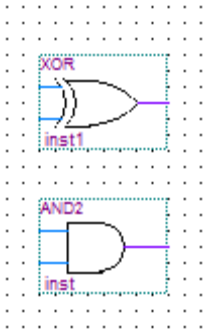
12. Gate functions (AND, OR, XOR etc.) are under **logic** directory, input and output pins are under **pin** directory, and flip-flops are under **storage** directory. Go to **logic** directory and select **and2** from the list for a 2-input AND gate and click **OK**.



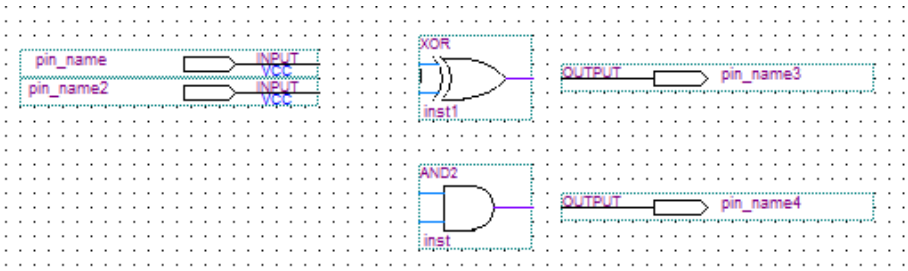
13. Then go to block diagram window and place the symbol on it.



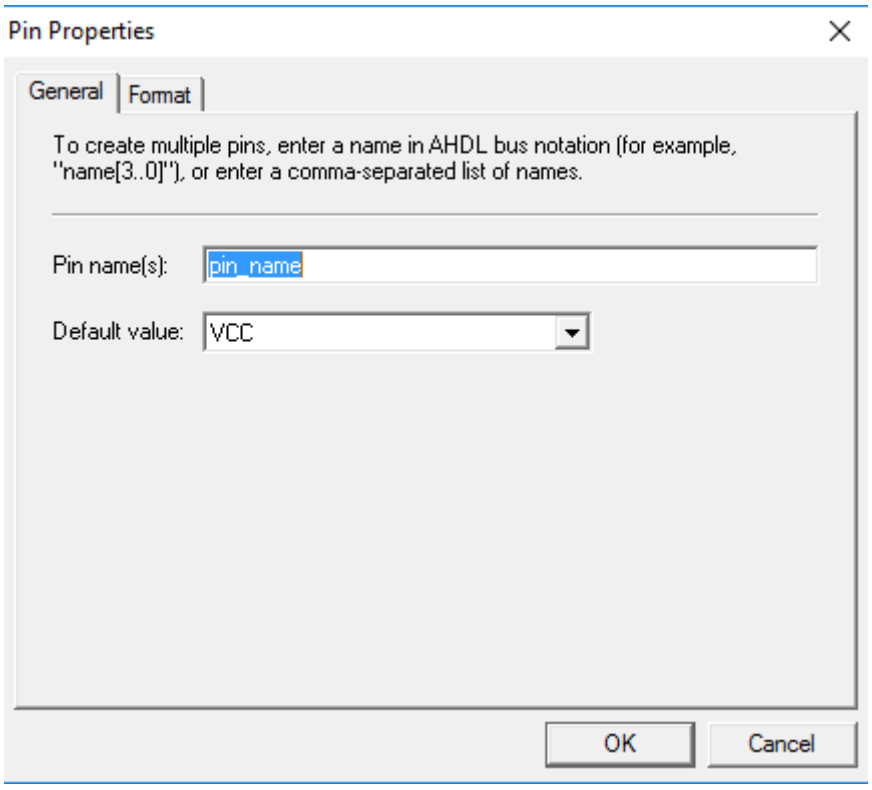
14. Do the same for the XOR gate.



15. Now, go to **pin** directory and select **input/output** for input/output pins and place them on the schematic.

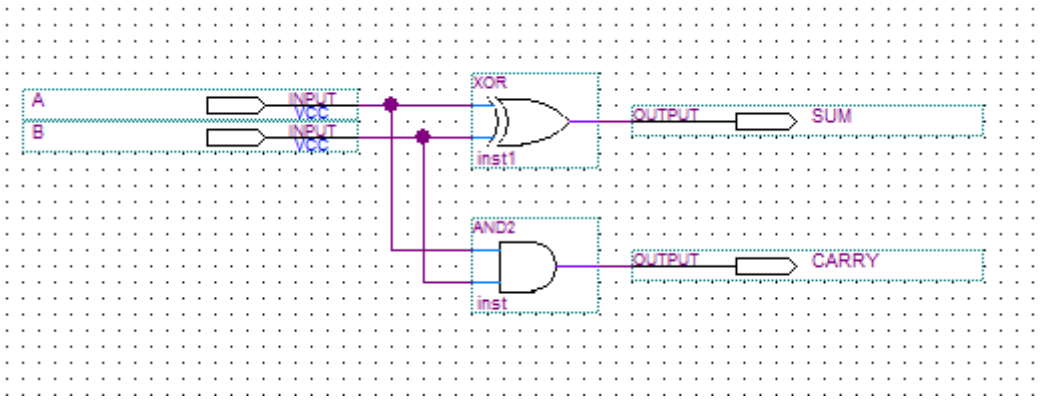


16. You can click on the pin names and rename them.



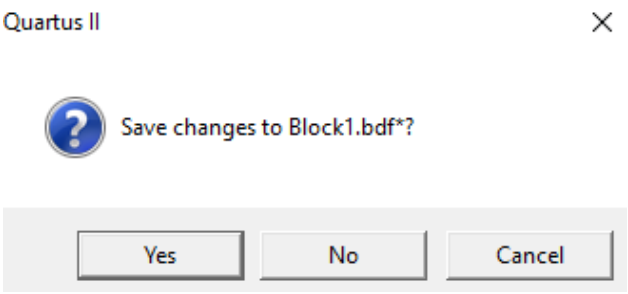
17. Now, in the block diagram window, move cursor to input/output pins on gates and you will see wiring icon showing up, or you can select **Orthogonal Node Tool** from the left menu bar.

18. Now, wire the gates and pins to construct a half-adder circuit. When completed, it should look something like the following:

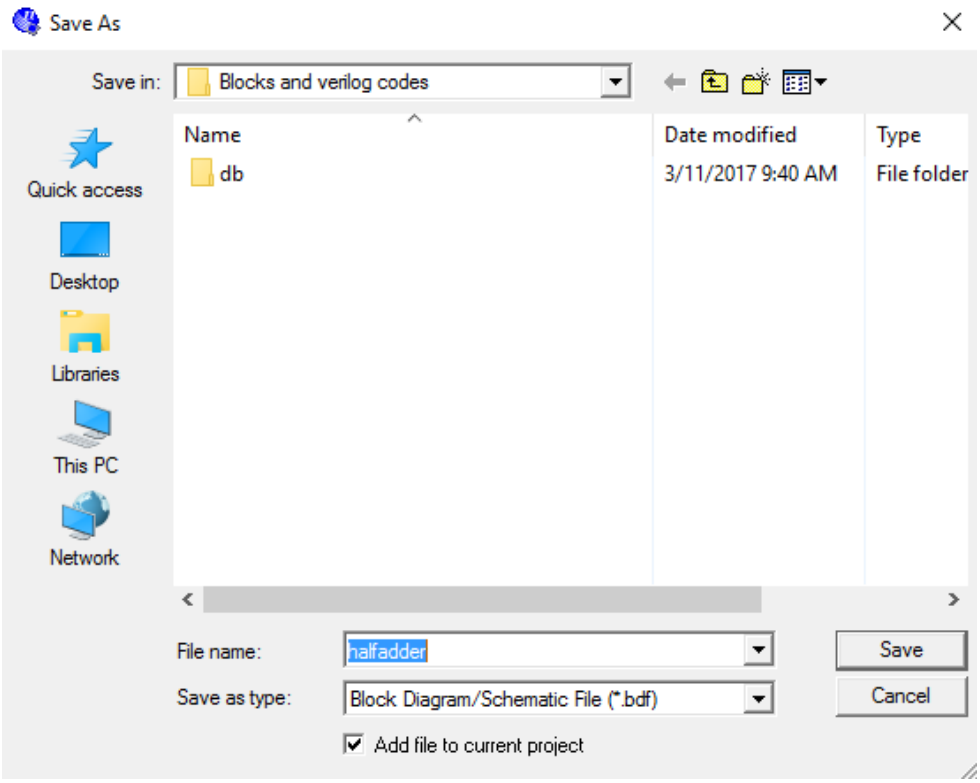


19. Click on **start compilation** button on the top menu bar.

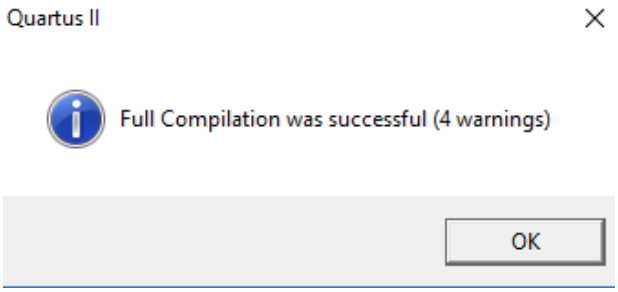
20. Click **Yes,** if you are prompted to save the block diagram.



21. File name of **top-level design entity** must be the same as that of the project name (e.g. half-adder in this case). Click **Save**.



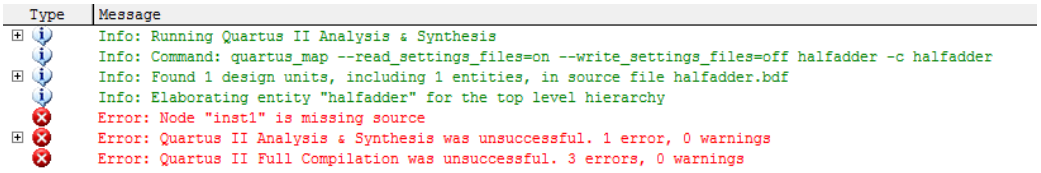
22. If compilation is successful, you will get a message like the following. Click **OK**.



23. Ignore warnings for now. **Compilation report-flow summary** will present you with the details:

Flow Status	Successful - Sat Mar 11 10:06:11 2017
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	halfadder
Top-level Entity Name	halfadder
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	2 / 33,216 (< 1 %)
Total combinational functions	2 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	4 / 475 (< 1 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

If any error occurs, you will find them at the bottom window.

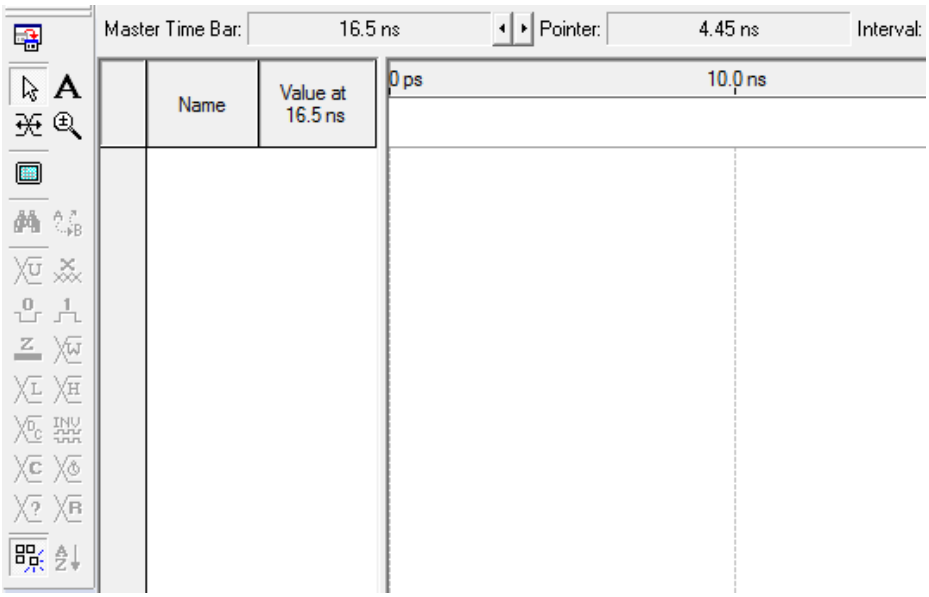


In case an error occurs, find the error on the block diagram and rerun compilation.

Running Simulation of Schematic file in Quartus II:

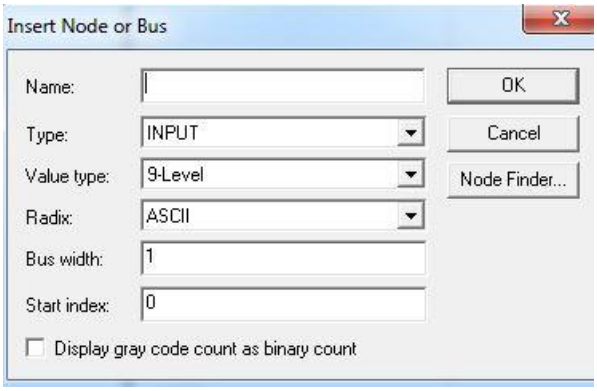
24. Go to ***File→New*** to create a vector waveform file which is required for simulating inputs and outputs. Select **Vector Waveform File** and click **OK**.

25. The following window will open up.



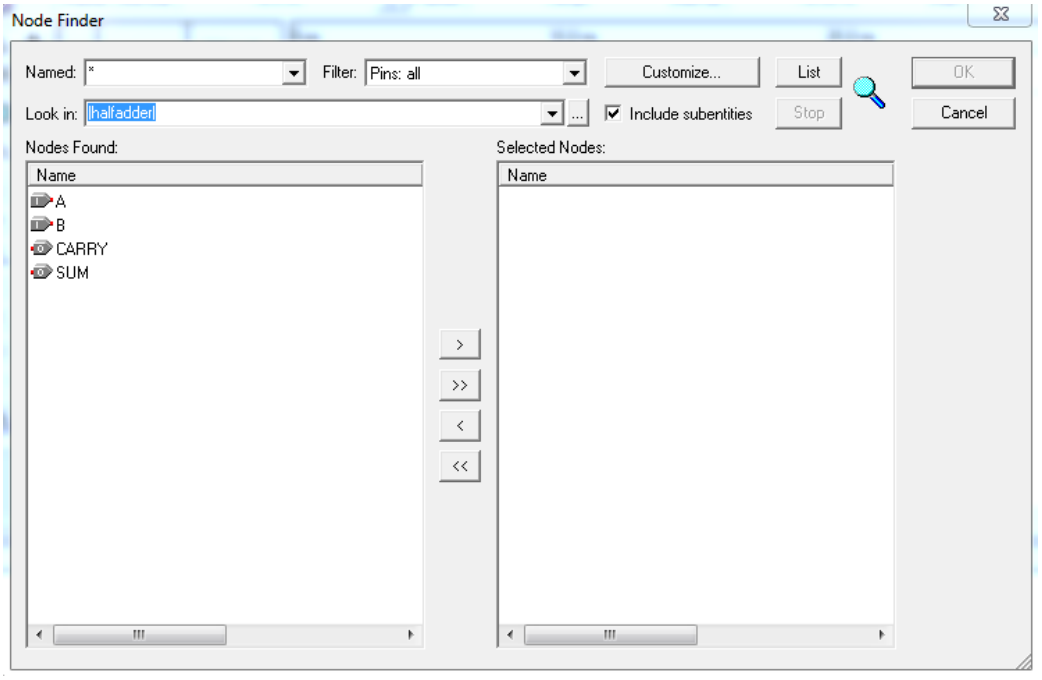
26. Double click on the white space under 'Name|Value at 16.5 ns'. Or Right click on that space and select ***Insert→Insert Node or Bus***.

27. In the **Insert Node or Bus** window, click **Node Finder**.

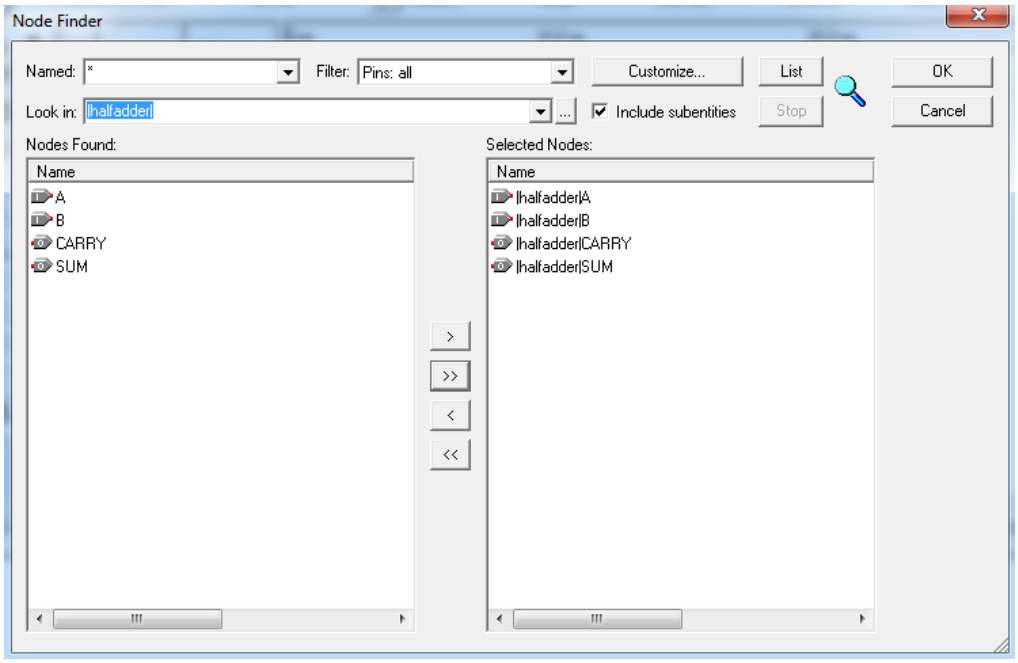


28. In the **Node Finder** window, Click **List**. Make sure **Pins:all** is selected under **Filter**.

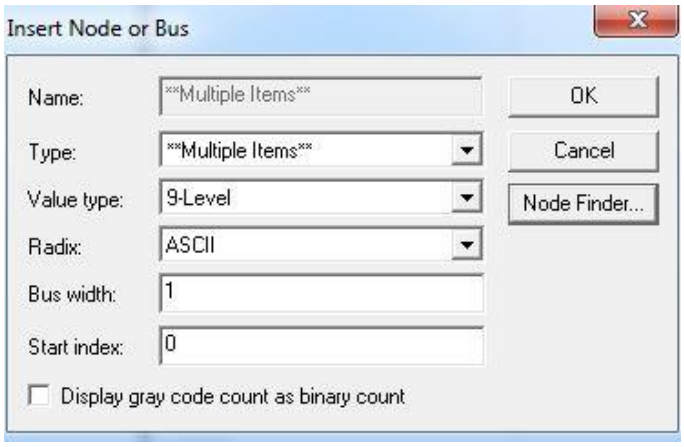
29. Now the window will look like the following. Click on the *'>>'* button.



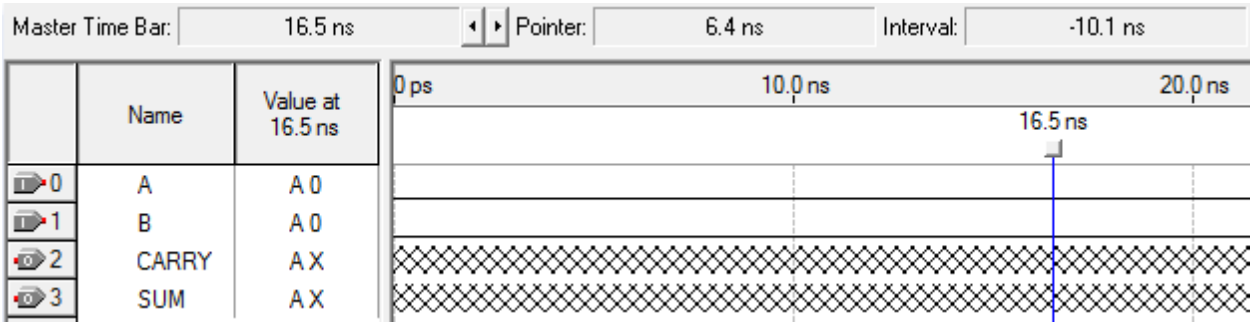
30. Now, it should appear like the following. Click **OK**.



31. In the following window, click **OK**.

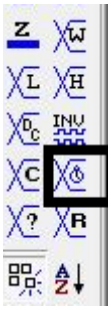


32. The vector waveform file will now look like the following:

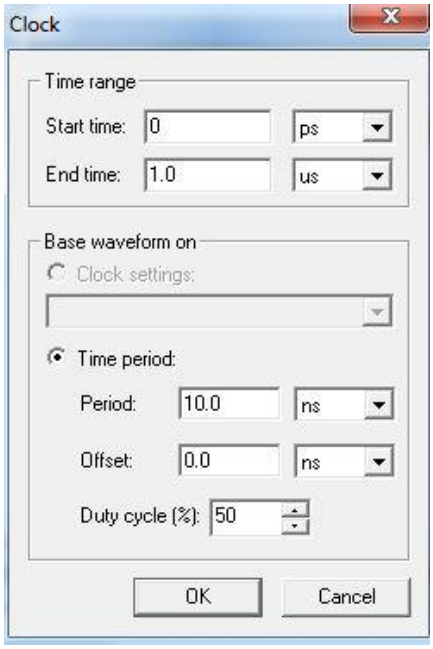


Now, you can clearly see the inputs and outputs.

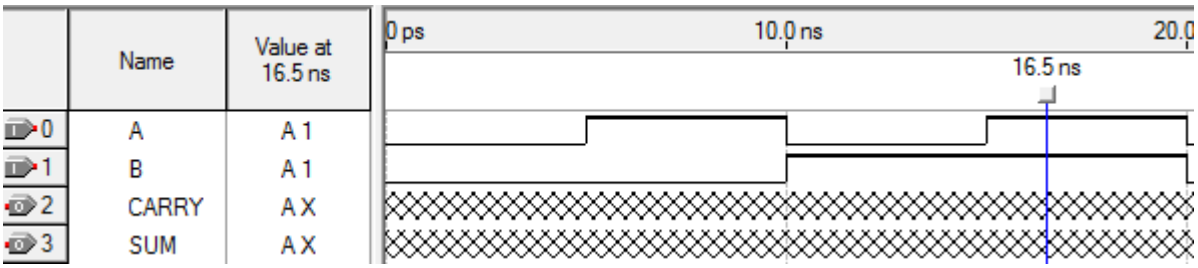
33. Select an input and from the left palette, click on the **Overwrite Clock** icon.



34. In the **Clock** window, set parameters of the clock. Only change the **Period** and keep everything else the same as before. Double clock period as you move from one input (**LSB**) to another as this will enable you to simulate the circuit for all possible input signal conditions.

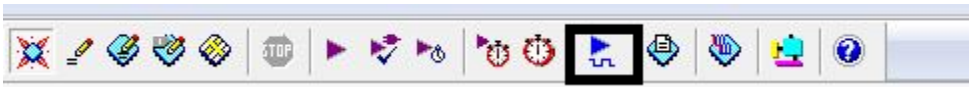


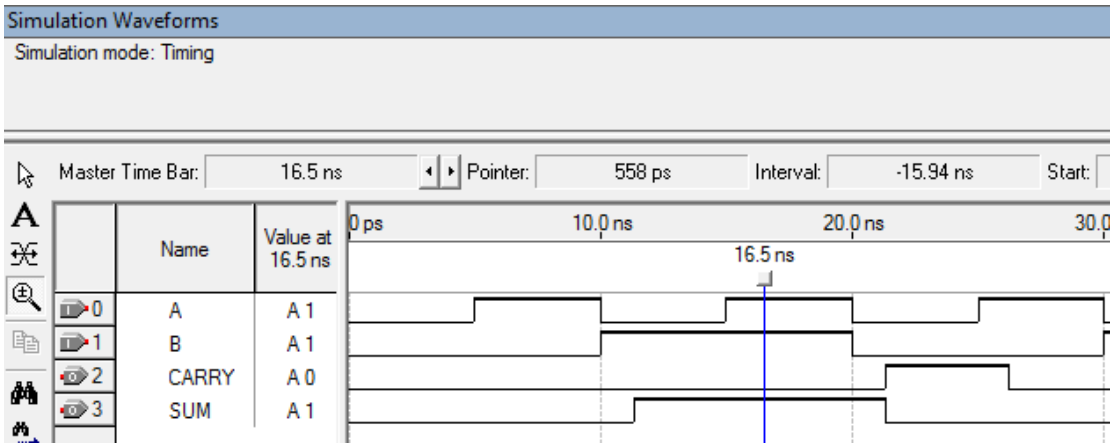
35. Now, after setting all the input clocks, **Vector Waveform File** will look like the following:



36. Save the **Vector Waveform File**. It must have the same name as the **Block Diagram/ Schematic file**.

37. Click on the blue play button and you should observe the simulated waveforms.

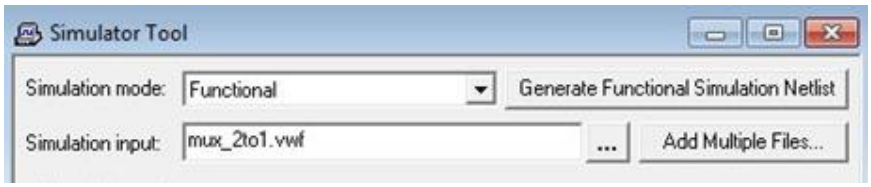




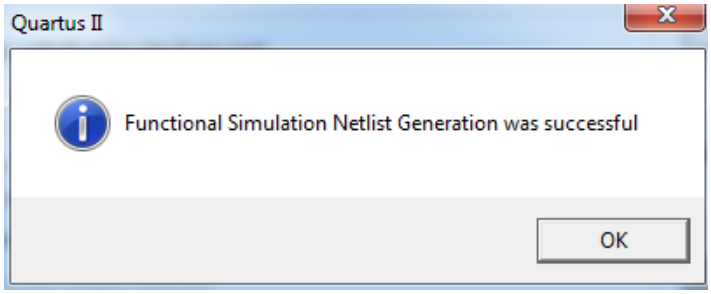
Note there is a delay between input and outputs which simulates the real effect of gate delays. If you are interested in only functional analysis rather than timing analysis, go through some more steps.

38. Go to **Processing→Simulator Tool**.

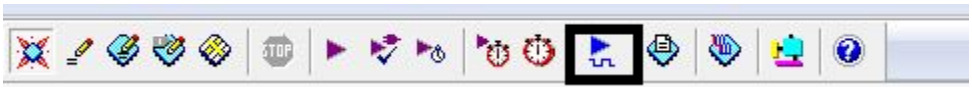
In the following window, Select **Functional** as **Simulation Mode** and click on **Generate Functional Simulation Netlist.**



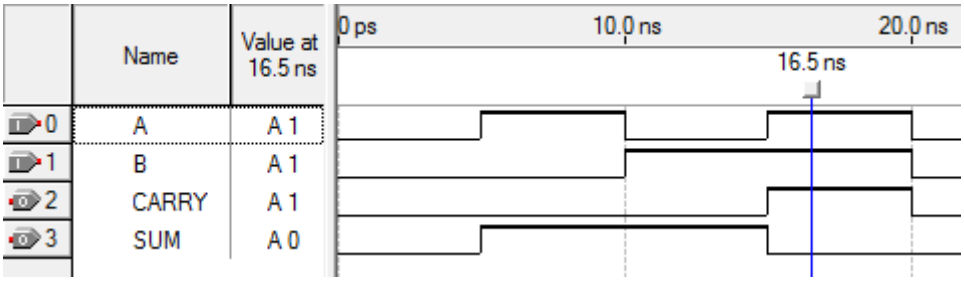
39. After generating functional simulation netlist, the following window will appear. Click **OK**.



40. Click on the blue play button and you should observe the following message.

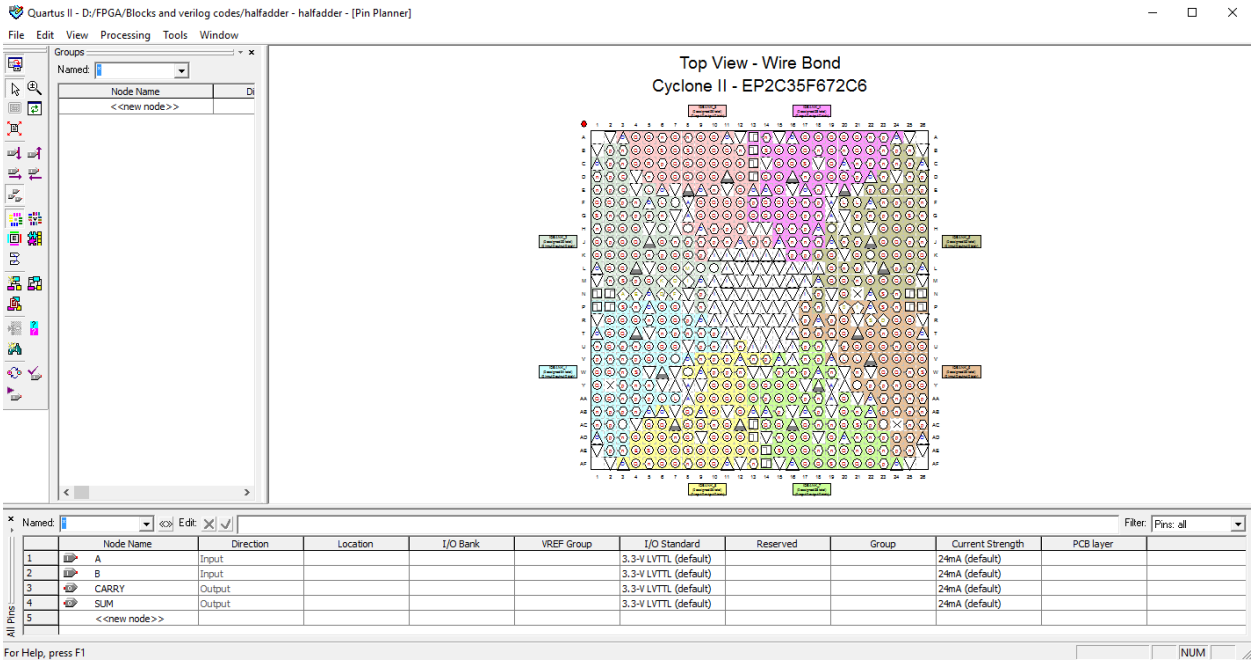


41. Go to **Simulation Report** window and observe the simulation waveforms, and note that there is no time delay between inputs and outputs and the behavior of the circuit is indeed that of a half-adder.



Programming FPGA in JTAG mode:

42. To load the program on to the FPGA board, go to ***Assignments→Pins*.**



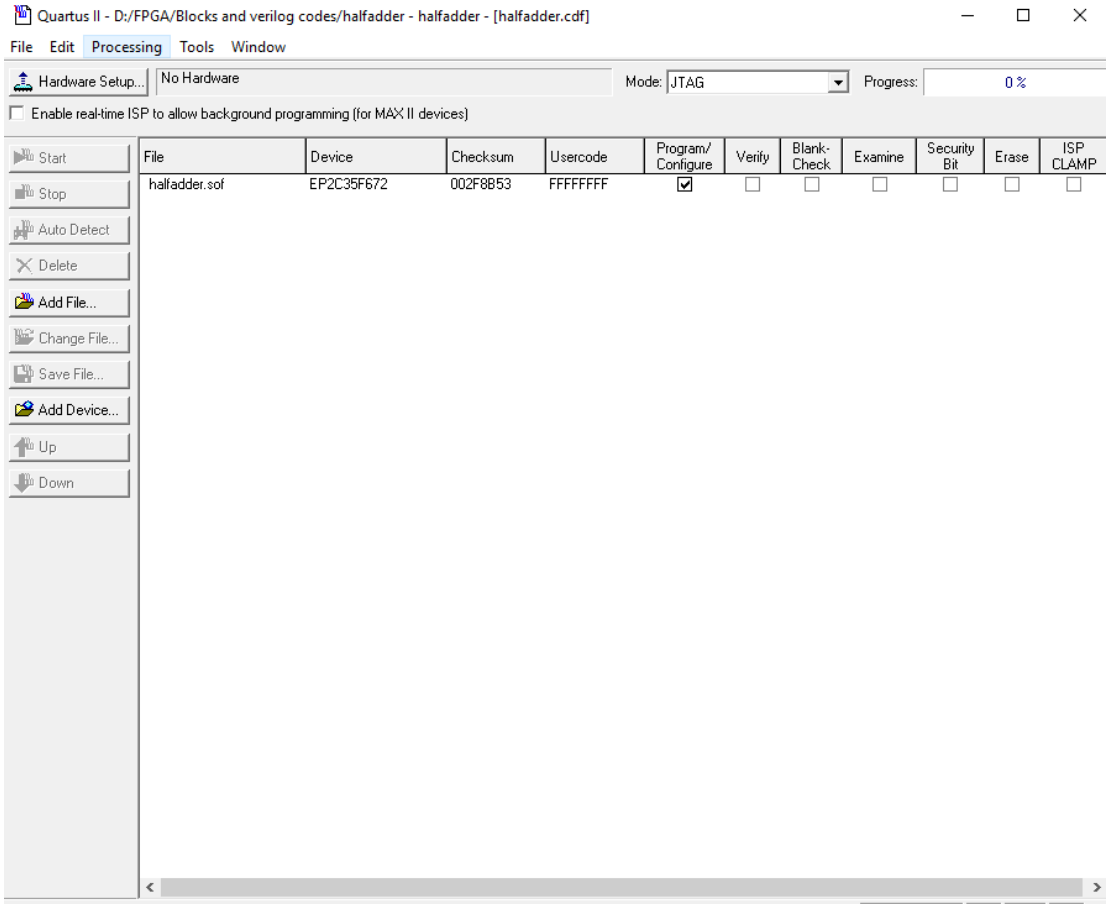
We will be using toggle switches (SW1 for A and SW0 for B) for inputs and LEDs (LED1 for CARRY and LED0 for SUM) for outputs. Note that, unless you compile the code once before assigning pins, the inputs and outputs will not appear in this window.

43. Assign the pins as follows. Click on **Location** and type in the pin name.

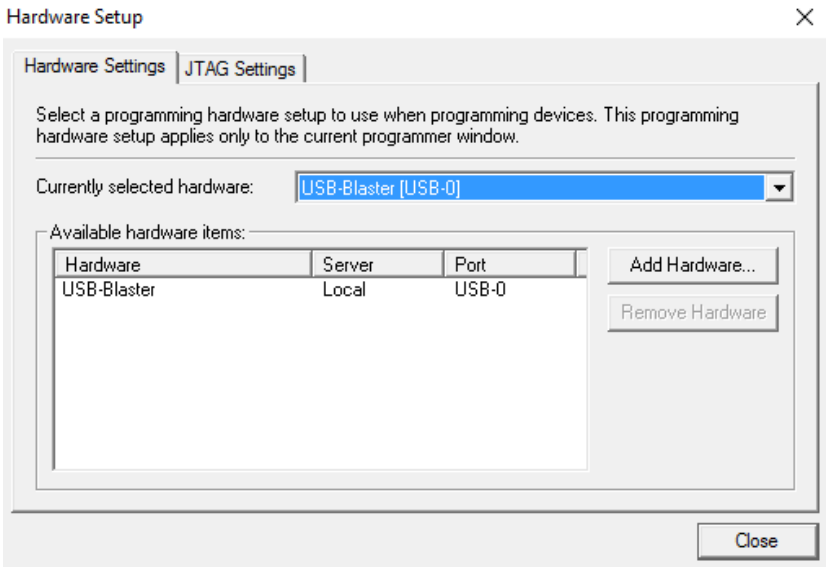
	Node Name	Direction	Location	I/O Bank
1	A	Input	PIN_N25	5
2	B	Input	PIN_N26	5
3	CARRY	Output	PIN_AF23	7
4	SUM	Output	PIN_AE23	7

44. Close the window. Note that, now you will have to compile the code again. So go **to start compilation**.

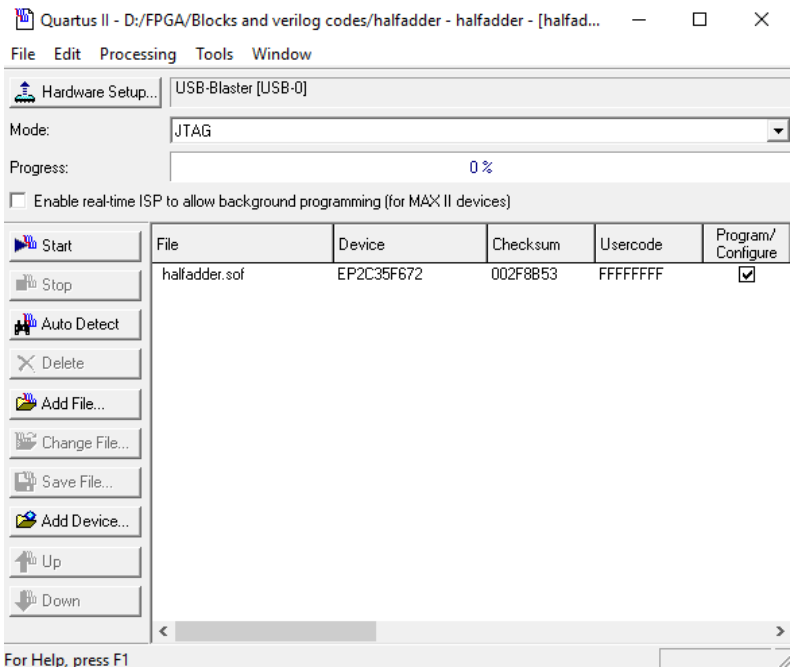
45. After successful compilation, go to ***Tools→Programmer***.



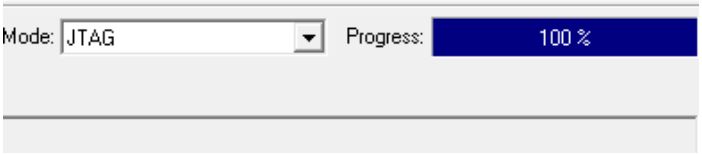
46. Select **Hardware Setup**. In the **Hardware Setup** window, select **USB-Blaster(USB-0)** and click **Close**.



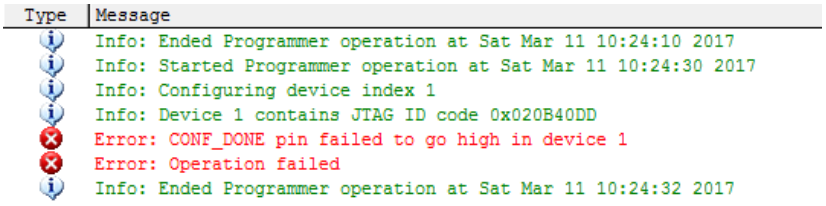
47. Now put a tick mark under **Program/Configure**, if it is not already checked. Make sure that **RUN/PROG** switch is in **RUN** position for **JTAG mode programming** and click **Start**.



48. You should see the progress bar moving and going to 100% if loading is successful.



Also note that, if an error occurs, it can be found in the message window:



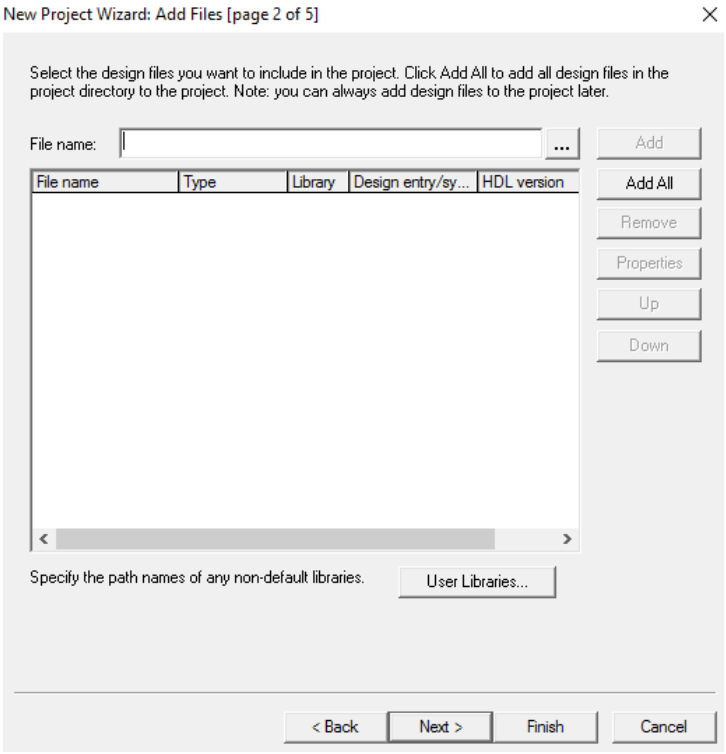
49. After loading .sof file successfully, check the functionality of the circuit the FPGA board.

Creating a Symbol from a Block Diagram/Schematic File and Using it for Hierarchical Design:

1. To create symbol of half-adder, go to ***File→Create/Update→Create Symbol Files for Current File***. Symbol will be created and saved in the project directory for half-adder.

2. Create a new project for full-adder using half-adder in a different directory.

3. Click on browse icon in **Add Files** window.

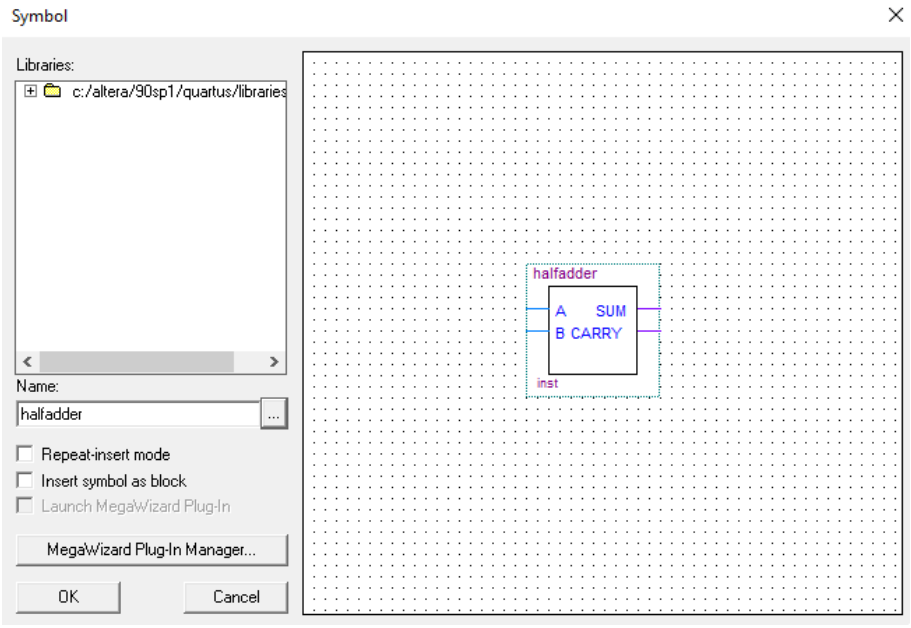


4. Select the **.bdf** file of half-adder. Click **Add**. That file will be added to the project.

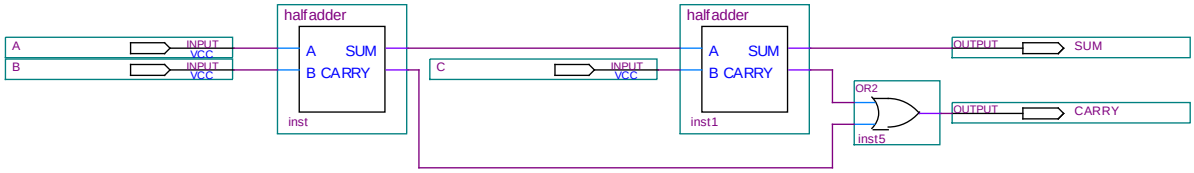
5. The rest of the steps of creating a project are like that for half-adder. Follow those steps.

6. After creating the project, go to ***File→New*** and select **Block Diagram/Schematic** file.

7. Open **Symbol Tool**. Click on Browse icon. Show the path for symbol file (**.bsf**) of half-adder.



8. After placing the symbols, wire them. The final block diagram should look as follows:



9. Now, save the schematic and follow all the steps you have completed for half-adder. You may assign pins for full-adder as follows:

	To	Location
1	A	PIN_N25
2	B	PIN_N26
3	C	PIN_P25
4	CARRY	PIN_AF23
5	SUM	PIN_AE23

Experiment: 4

Experiment name: *Design a Combinational circuit that will act as an Adder if control bit is '0' and as a sub tractor if control bit is '1'.*

Introduction:

Addition of two 4-bit binary numbers can be easily done using a 4-bit binary adder IC (7483/74283).
Taking the 2's complement of the subtrahend and then adding that with the minuend can do subtraction of two 4-bit binary numbers.

Subtraction with Complements

The direct method of subtraction, we borrow a 1 from a higher significant position when the minuend digit is smaller than the subtrahend digit. This seems to be easiest when people perform subtraction with paper and pencil. When subtraction is implemented with digital hardware, this method is found to be less efficient than the method that uses complements. For more details on complements of a number see Annexure III.

The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:
1. Add the minuend M to the r's complement of the subtrahend N. This performs

$$M + (r^n - N) = M - N + r^n.$$

2. If $M \geq N$, the sum will produce an end carry, r^n, which is discarded; what is left is the result $(M-N)$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N-M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Examples

Subtraction between two binary numbers
Let two binary numbers X=1100 and Y=0110, find out (a) X-Y and (b) Y-X
(a)

X	=	1100
2's complement of Y	=	+ 1010
Sum	=	10110
Discard end Carry 2^4	=	- 10000
Answer	=	0110

(b)

Y	=	0110
2's complement of X	=	+ 0100
Sum	=	1010

There is No end Carry.

So, Y-X = - (2's complement of 1010) = - 0110

Special Use of XOR Gate

If there are 2 inputs A and B and output X in an XOR gate then the truth table will be

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

The output of an XOR gate is HIGH whenever the two inputs are different.

XOR Gate as Inverter

If one of the input, A is always 1(High) then the truth table will be like

A	B	X
1	0	1
1	1	0

So from here we can see that the output X will be inverted version of B.

So, $X = B'$; if A=1.

XOR Gate as Buffer

If one of the inputs, A is always 0 (LOW) then the truth table will be like

A	B	X
0	0	0
0	1	1

So from here we can see that the output X will be same as B.

So, $X = B$; if A=0.

Caution:

1. Remember to properly identify the pin numbers so that no accidents occur.
2. Properly bias the ICs with appropriate voltages to appropriate pins.

Equipment:

1. Trainer Board
2. IC 74283,7408,7432,7486.
3. Microprocessor Data handbook

Procedure:

1. Draw the logic diagram to implement the task.
2. Select the required ICs.

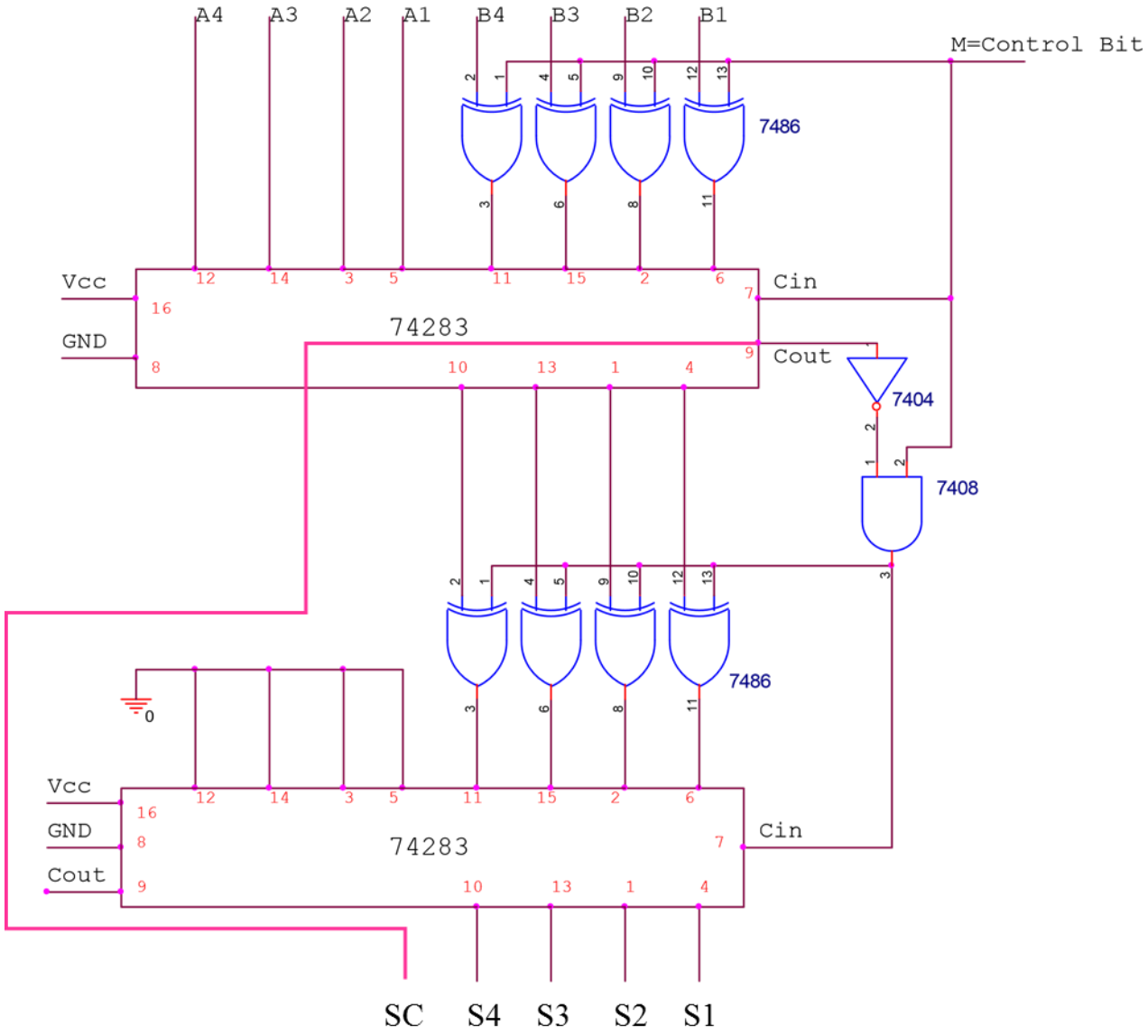


Figure 4.1: Logic diagram of the Combinational Circuit

3. Record the outputs for different (a representative sample of) inputs in the following table:

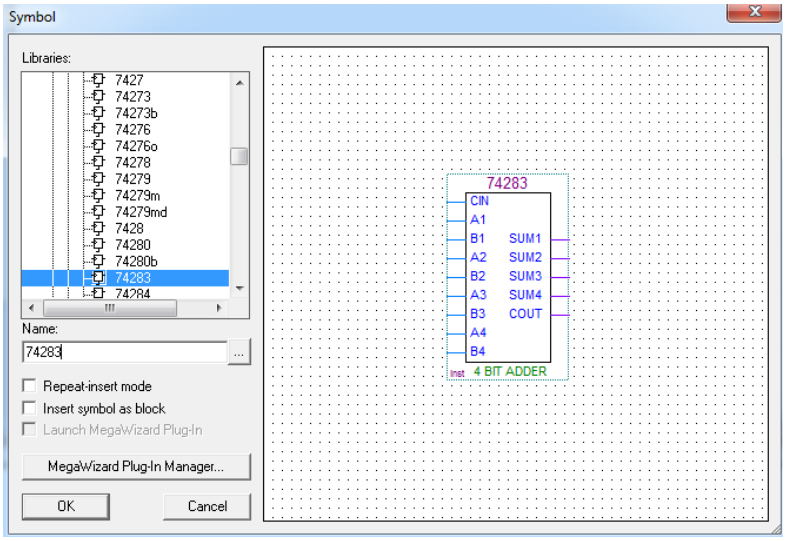
Input			**Output (S)**	
A	***B***	***M***	***SC***	***S4 S3 S2 S1***
0001	0010	0 (+)		
0001	0010	1 (–)		
1001	0011	0 (+)		
1011	0111	1 (–)		
1011	0111	0 (+)		
1100	1001	1 (–)		
1111	1111	0 (+)		
1111	1111	1 (–)		

Procedure for FPGA

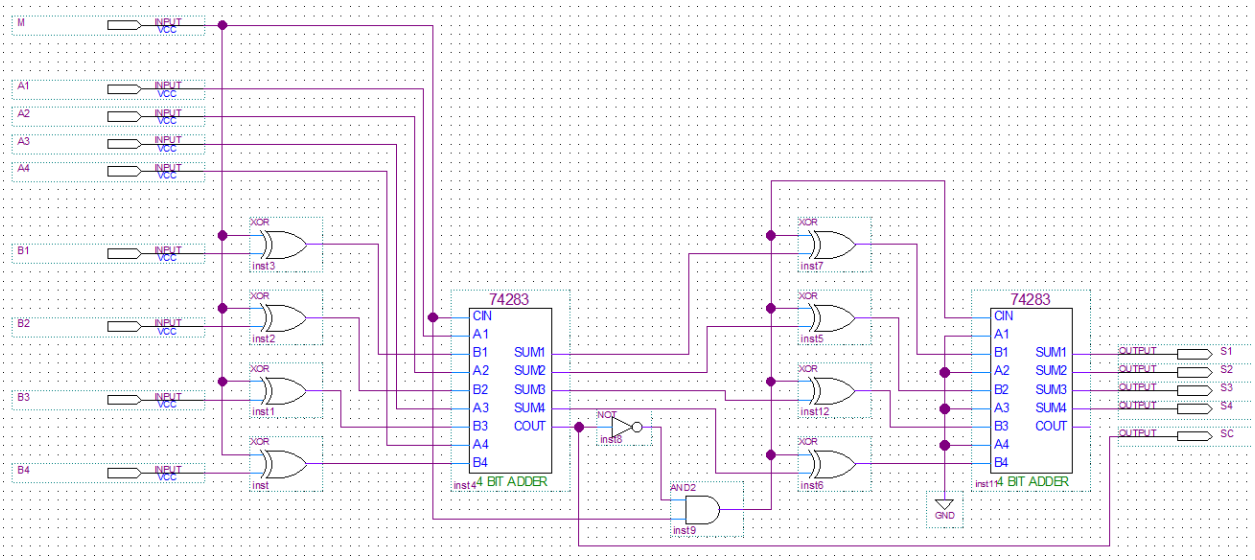
1. Follows steps mentioned in '**Creating a new project in Quartus II**' from Experiment 3 to create a new project.

2. Follow steps mentioned in '**Creating a Block Diagram/Schematic file in Quartus II**' from Experiment 3 to create a new schematic.

3. Under library directories, type 74283 under name and IC 74283 will appear. Place it on schematic.



4. Place all other components in the same manner, and connect those using wires and add input/output pins. Components required are: ***gnd***, ***input***, ***output***, ***7402***, ***7486***, and ***74283***.



5. Now compile the schematic file and run a simulation using vector waveform file. After successful simulation, assign pins and then program FPGA in JTAG mode following steps mentioned in Experiment 3. Assign pins as follows:

INPUT			**OUTPUT**		
Signal	**Switch No.**	**Pin No.**	**Signal**	**LED No.**	**Pin No.**
M	SW8	PIN_B13	S1	LEDR0	PIN_AE23
A1	SW4	PIN_AF14	S2	LEDR1	PIN_AF23
A2	SW5	PIN_AD13	S3	LEDR2	PIN_AB21
A3	SW6	PIN_AC13	S4	LEDR3	PIN_AC22
A4	SW7	PIN_C13	SC	LEDR4	PIN_AD22
B1	SW0	PIN_N25			
B2	SW1	PIN_N26			
B3	SW2	PIN_P25			
B4	SW3	PIN_AE14			

6. Verify that the observed output at FPGA is the same as the one observed using discrete ICs.

Experiment: *5*

Experiment name: *Design a BCD adder that will add two BCD numbers and the sum will be also in BCD.*

Introduction:

Before discussing BCD Adder circuitry first, we can review the basic concepts of BCD no system and BCD addition technique.

BCD

Binary coded decimal (BCD) is a weighted code that is commonly used in many computers and calculators to represent decimal numbers. This code takes each decimal digit and represents it by a four-bit code ranging from 0000 to 1001.

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	00010000
11	1011	00010001
12	1100	00010010
13	1101	00010011
14	1110	00010100
15	1111	00010101

The table illustrates the difference between straight binary and BCD. BCD represents each decimal digit with a 4-bit code. Notice that the codes 1010 through 1111 are not used in BCD.

BCD Addition

The addition of decimal numbers that are in BCD form can be best understood by considering the two cases that can occur when two decimal digits are added.

Sum Equals 9 or Less

Consider adding 45 and 33 using BCD to represent each digit:

```
  45        0100 0101   ← BCD for 45
+ 33      + 0011 0011   ← BCD for 33
  78        0111 1000   ← BCD for 78
```

In the examples above, none of the sums of the pairs of decimal digits exceeded 9; therefore, *no decimal carries were produced.* For these cases, the BCD addition process is straightforward and is actually the same as binary addition.

Sum Greater than 9

Consider the addition of 6 and 7 in BCD:

```
 6      0110   ← BCD for 6
+7    + 0111   ← BCD for 7
+13    1101   ← invalid code group for BCD
```

The sum 1101 does not exist in the BCD code; it is one of the six forbidden or invalid four-bit code groups. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs, the sum must be corrected by the addition of six (0110) to consider the skipping of the six invalid code groups:

```
        0110   ← BCD for 6
      + 0111   ← BCD for 7
        1101   ← invalid sum
        0110   ← add 6 for correction
      ------
 0001   0011   ← BCD for 13
   1      3
```

As shown above, 0110 is added to the invalid sum and produces the correct BCD result. Note that with the addition of 0110, a carry is produced in the second decimal position. This addition must be performed whenever the sum of the two decimal digits is greater than 9.

Consider the addition of 59 and 38 in BCD:

```
                1
59     0101   1001   ← BCD for 59
+38  + 0011   1000   ← BCD for 38
97     1001   0001   ← perform addition
              0110   ← add 6 to correct
       ----------
       1001   0111     BCD for 97
        9      7
```

Here, the addition of the least significant digits (LSDs) produces a sum of 17 = 10001**.** This generates a carry into the next digit position to be added to the codes for 5 and 3. Since 17 7 9, a correction factor of 6 must be added to the LSD sum. Addition of this correction does not generate a carry; the carry was already generated in the original addition.

To summarize the BCD addition procedure:

1. Using ordinary binary addition, add the BCD code groups for each digit position.
2. For those positions where the sum is 9 or less, no correction is needed. The sum is in proper BCD form.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum to get the proper BCD result. This case always produces a carry into the next digit position, either from the original addition (step 1) or from the correction addition.

BCD ADDER

Consider the arithmetic addition of two decimal digits in BCD, together with a possible carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater

than 9 + 9 + 1 = 19, the 1 in the sum being an input carry. Suppose we apply two BCD digits to a 4-bit binary adder. The adder will form the sum in *binary* and produce a result that may range from 0 to 19. These binary numbers are listed in Table and are labeled by symbols *K*, Z_4, Z_3, Z_2, and Z_1. *K* is the carry, and the subscripts under the letter Z represent the weights 4, 3, 2, and 1 that can be assigned to the four bits in the BCD code. The first column in the table lists the binary sums as they appear in the outputs of a 4-bit *binary* adder. The output sum of two *decimal digits* must be represented in BCD and should appear in the form listed in the second column of the table. The problem is to find a simple rule by which the binary number, in the first column can be converted to the correct BCD-digit representation of the number in the second column. In examining the contents of the table, it is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed. When the binary sum is greater than 1001, we obtain an invalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required. The logic circuit that detects the necessary correction can be derived from the table entries. It is obvious that a correction is needed when the binary sum has an output carry $K = 1$. The other six combinations from 1010 to 1111 that need a correction have a 1 in position Z_4. To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_4 we specify further that, either Z_3 or Z_2 must have a 1 along with Z_4. The condition for a correction and an output carry can be expressed by the Boolean function

$$C = K + Z_4Z_3 + Z_4Z_2$$

When C = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.

A *BCD adder* is a circuit that adds two BCD digits in parallel and produces a sum digit also in BCD. A BCD adder must include the correction logic in its internal construction. To add 0110 to the binary sum, we use a second 4-bit binary adder, as shown in Figure. The two decimal digits, together with the input carry, are first added in the top 4-bit binary adder to produce the binary sum. When the output carry is equal to zero, nothing is added to the binary sum. When it is equal to one, binary 0110 is added to the binary sum through the bottom 4-bit binary adder. The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal. The BCD adder can be constructed with three IC packages.

Caution:

1. Remember to properly identify the pin numbers so that no accidents occur.
2. Properly bias the ICs with appropriate voltages to appropriate pins.

Procedure:

1. Draw the logic diagram to implement the task.

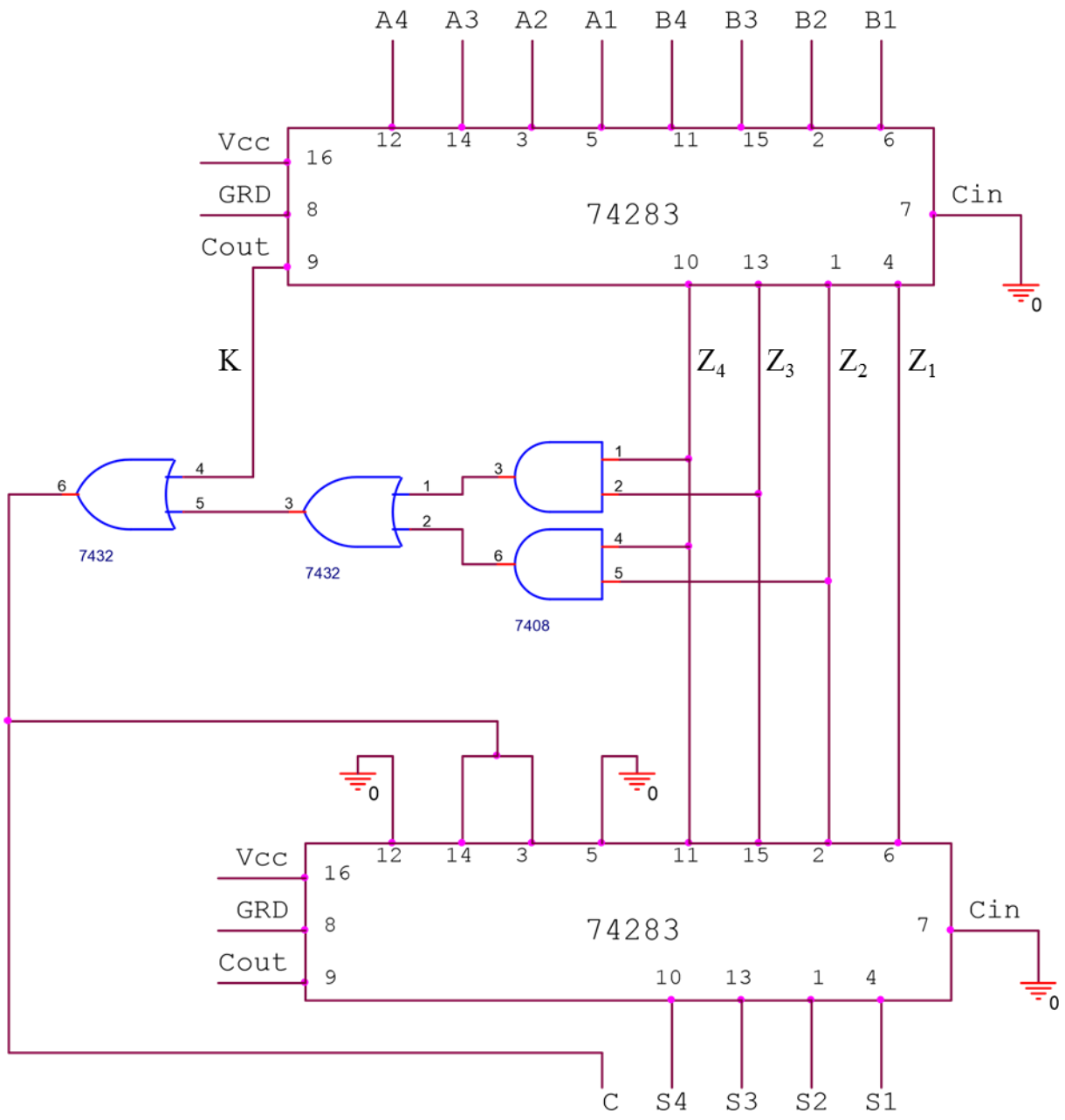


Figure 5.1: BCD Adder

2. Select the required ICs.
3. Verify the following truth table for 20output values (0-20).

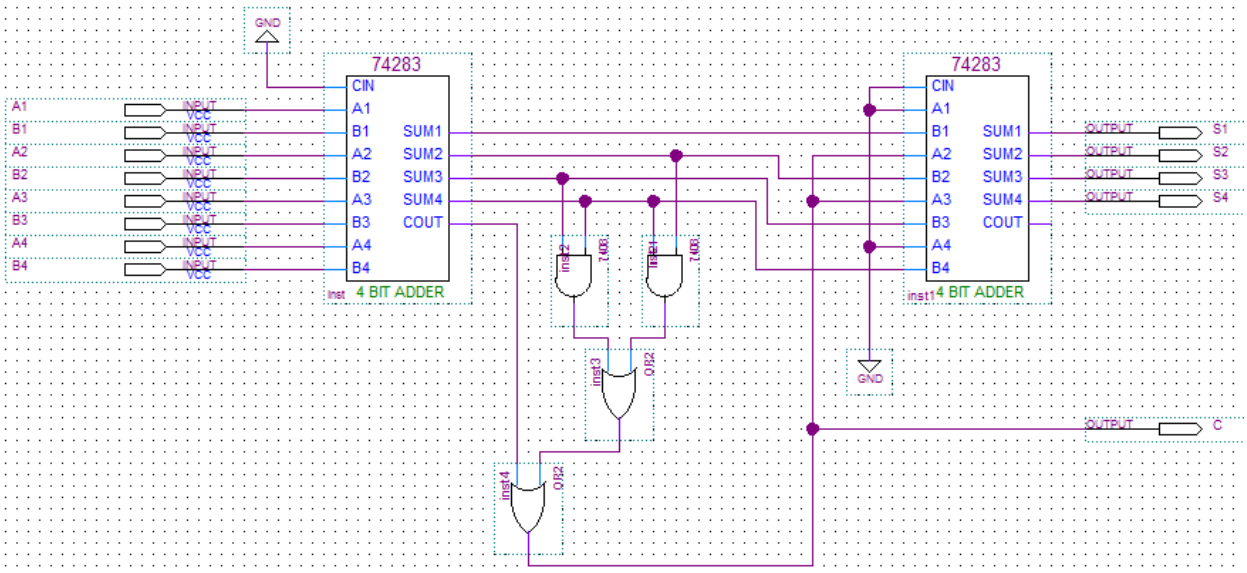
	Binary Sum						BCD Sum				
Decimal	K	Z_4	Z_3	Z_2	Z_1		C	S_4	S_3	S_2	S_1
0	0	0	0	0	0		0	0	0	0	0
1	0	0	0	0	1		0	0	0	0	1
2	0	0	0	1	0		0	0	0	1	0
3	0	0	0	1	1		0	0	0	1	1
4	0	0	1	0	0		0	0	1	0	0
5	0	0	1	0	1		0	0	1	0	1
6	0	0	1	1	0		0	0	1	1	0
7	0	0	1	1	1		0	0	1	1	1
8	0	1	0	0	0		0	1	0	0	0
9	0	1	0	0	1		0	1	0	0	1
10	0	1	0	1	0		**1**	**0**	**0**	**0**	**0**
11	0	1	0	1	1		**1**	**0**	**0**	**0**	**1**
12	0	1	1	0	0		**1**	**0**	**0**	**1**	**0**
13	0	1	1	0	1		**1**	**0**	**0**	**1**	**1**
14	0	1	1	1	0		**1**	**0**	**1**	**0**	**0**
15	0	1	1	1	1		**1**	**0**	**1**	**0**	**1**
16	1	0	0	0	0		**1**	**0**	**1**	**1**	**0**
17	1	0	0	0	1		**1**	**0**	**1**	**1**	**1**
18	1	0	0	1	0		**1**	**1**	**0**	**0**	**0**
19	1	0	0	1	1		**1**	**1**	**0**	**0**	**1**

Procedure for FPGA

1. Create a new project and create a new block diagram/schematic file.

2. Compile and simulate the schematic. If everything is ok, assign pins as follows:

INPUT			OUTPUT		
Signal	**Switch No.**	**Pin No.**	**Signal**	**LED No.**	**Pin No.**
A1	SW4	PIN_AF14	C	LEDR4	PIN_AD22
A2	SW5	PIN_AD13	S1	LEDR0	PIN_AE23
A3	SW6	PIN_AC13	S2	LEDR1	PIN_AF23
A4	SW7	PIN_C13	S3	LEDR2	PIN_AB21
B1	SW0	PIN_N25	S4	LEDR3	PIN_AC22
B2	SW1	PIN_N26			
B3	SW2	PIN_P25			
B4	SW3	PIN_AE14			



3. Test the functionality of the designed circuit using switches and LEDs on the FPGA board for the following table –

A (Decimal)	A (Binary)				***B (Decimal)***	B (Binary)				***BCD SUM (Decimal)***	BCD SUM (Binary)				
	A4	A3	A2	A1		B4	B3	B2	B1		C	S4	S3	S2	S1
5	0	1	0	1	***0***	0	0	0	0	***05***					
7	0	1	1	1	***4***	0	1	0	0	***11***					
2	0	0	1	0	***6***	0	1	1	0	***08***					
9	1	0	0	1	***8***	1	0	0	0	***17***					
1	0	0	0	1	***3***	0	0	1	1	***04***					

Experiment: 6
Experiment name: *Introduction to Multiplexers.*

Introduction
Multiplexers are the most important attributions of digital circuitry in communication hardware. These digital switches enable us to achieve the communication network we have today. In this experiment the students will have to construct MUX (as they call multiplexers) with simple logic gates and they will implement general logic using 8:1 MUX as the basic constructional unit.

Multiplexer
A modern home stereo system may have a switch that selects music from one of four sources: a cassette tape, a compact disc (CD), a radio tuner, or an auxiliary input such as audio from a VCR or DVD. The switch selects one of the electronic signals from one of these four sources and sends it to the power amplifier and speakers. In simple terms, this is what a **multiplexer (MUX)** does: it selects one of several input signals and passes it on to the output.

A digital multiplexer or data selector is a logic circuit that accepts several digital data inputs and selects one of them at any given time to pass on to the output. The routing of the desired data input to the output is controlled by SELECT inputs (often referred to as ADDRESS inputs). Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.
A 4 to l line multiplexer is shown in Figure. Each of the four input lines, I_0 to I_3 is applied to one input of an AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate. The function table, Figure lists the input-to-output path for each possible bit combination of the selection lines. To demonstrate the circuit operation, consider the case when $S_1S_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of I_2 thus providing a path from the selected input to the output.

Caution:
1. Remember to properly identify the pin numbers so that no accidents occur.
2. Properly bias the ICs with appropriate voltages to appropriate pins.

Equipment:
1. Trainer Board
2. IC 74151, 7432, 7408, 7404
3. Microprocessor Data handbook.

Job 1:
Implementation of a four to one way Multiplexer, (4:1 MUX) with basic gates.

Procedure:

1. Write the truth table for four to one way MUX.

S_1	S_0	Y

2. Write the Boolean function for the output logic.
3. Draw the logic diagram to implement the Boolean function.

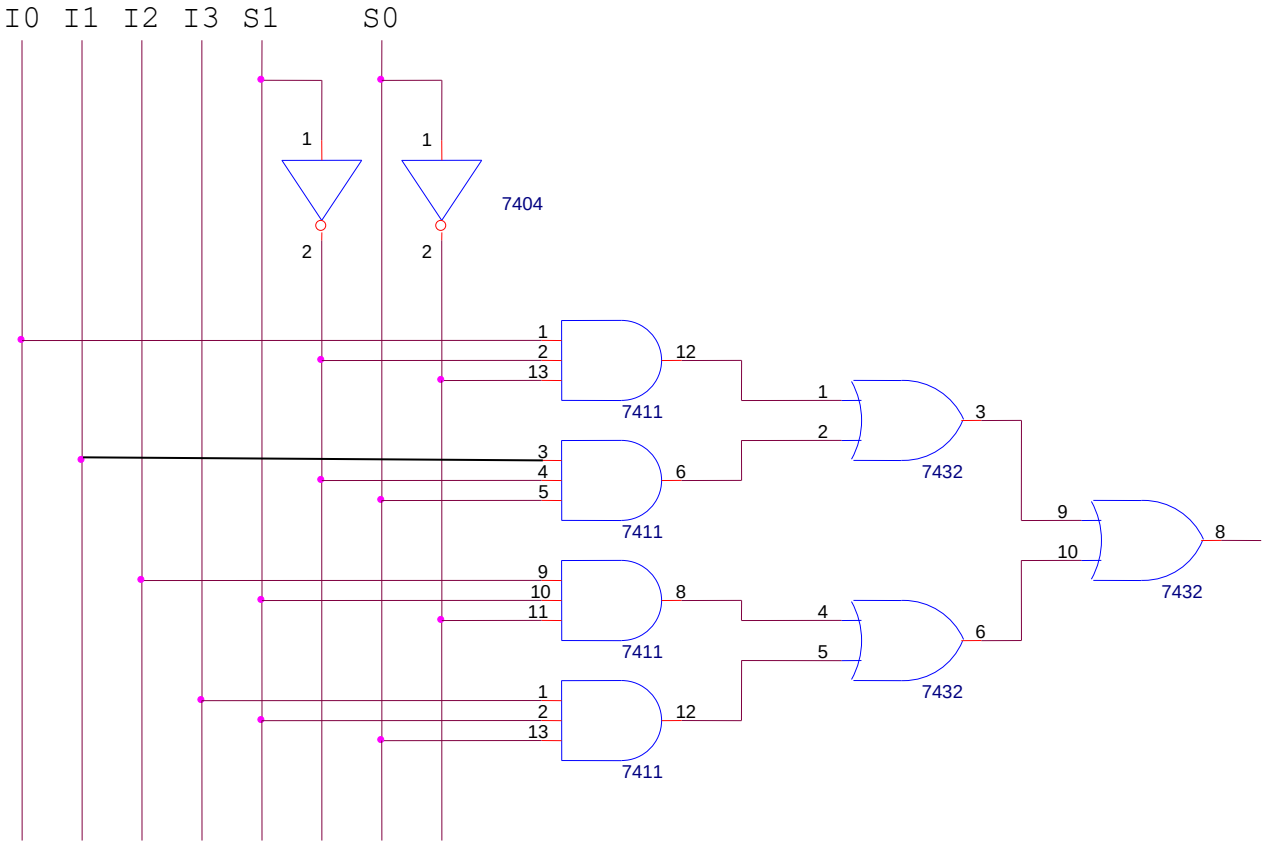


Figure 6.1: 4 to 1 Multiplexer

4. Select ICs from the equipment list.
5. Observe and note the output logic for all combination of inputs.

Job 2:

Implement the following function using an 8:1 MUX.

$$F(A,B,C,D)=\sum(0,1,3,5,8,9,14,15)$$

If we have a Boolean function of $n + 1$ variables, we take n of these variables and connect them to the selection lines of a multiplexer. The remaining single variable of the function is used for the inputs of the multiplexer. If A is this single variable, the inputs of the multiplexer are chosen

to be either A or A' or 1 or 0. By judicious use of these four values for the inputs and by connecting the other variables to the selection lines, one can implement any Boolean function with a multiplexer. In this way, it is possible to generate any function of $n + 1$ variables with a 2^n to1 multiplexer.

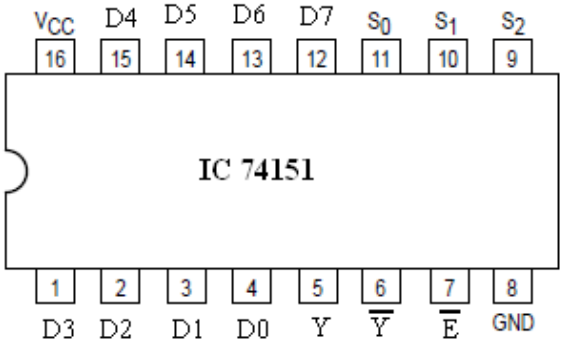


Fig 6.1. Pin diagram of IC 74151

Procedure:

1. Write the truth table for the above function.

A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Let, B, C, D of the 4 variables (A, B, C, D) are connected to the selection lines of a multiplexer and remaining single variable A of the function is used for the inputs of the multiplexer

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	1	1		1		1		
A	1	1					1	1
	1	**1**	**0**	A'	**0**	A'	A	A

2. Draw the logic diagram to implement the Boolean function.
3. Select ICs from the equipment list.
4. Observe and note the output logic for all combination of inputs.

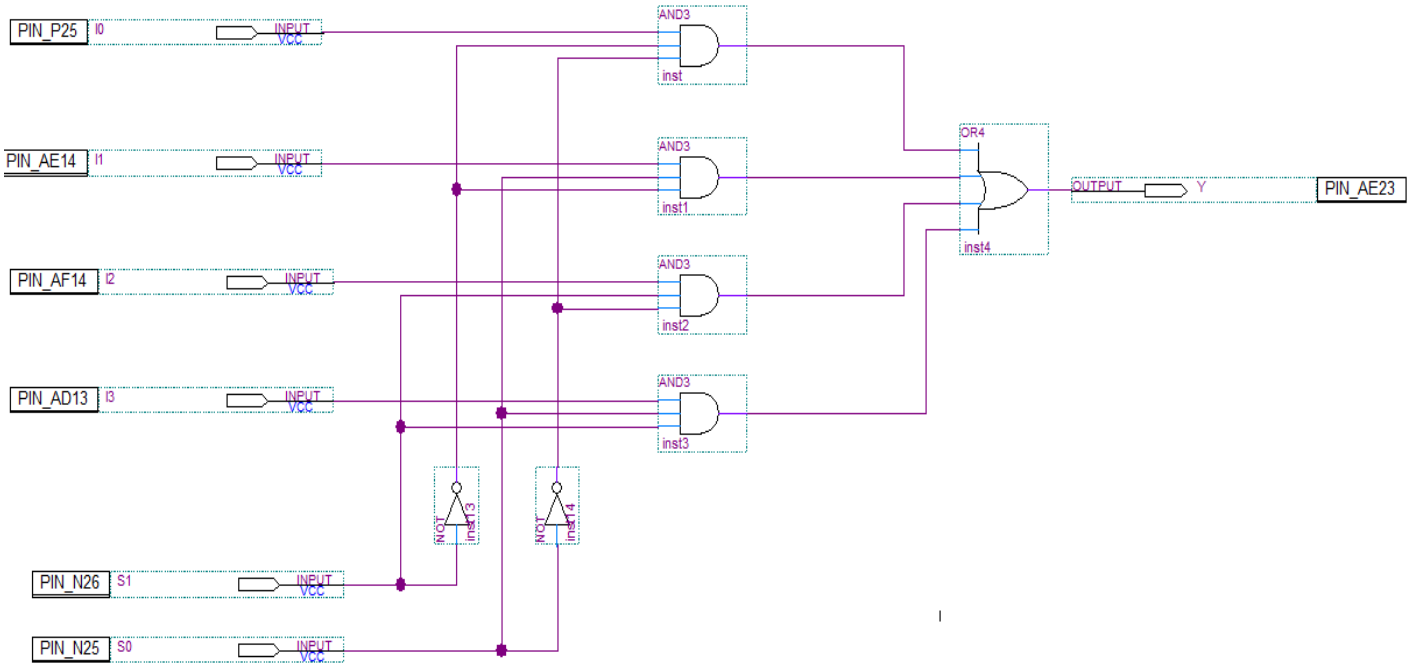
Procedure for FPGA:

Design 4:1 MUX

1. Create a new project and create a new block diagram/schematic file.
2. Compile and simulate the schematic. If everything is ok, assign pins as follows:

INPUT			OUTPUT		
Signal	**Switch No.**	**Pin No.**	**Signal**	**LED No.**	**Pin No.**
I0	SW2	PIN_P25	Y	LEDR0	PIN_AE23
I1	SW3	PIN_AE14			
I2	SW4	PIN_AF14			
I3	SW5	PIN_AD13			
S0	SW0	PIN_N25			
S1	SW1	PIN_N26			

3. After assigning pins, the final schematic should look like the following one:

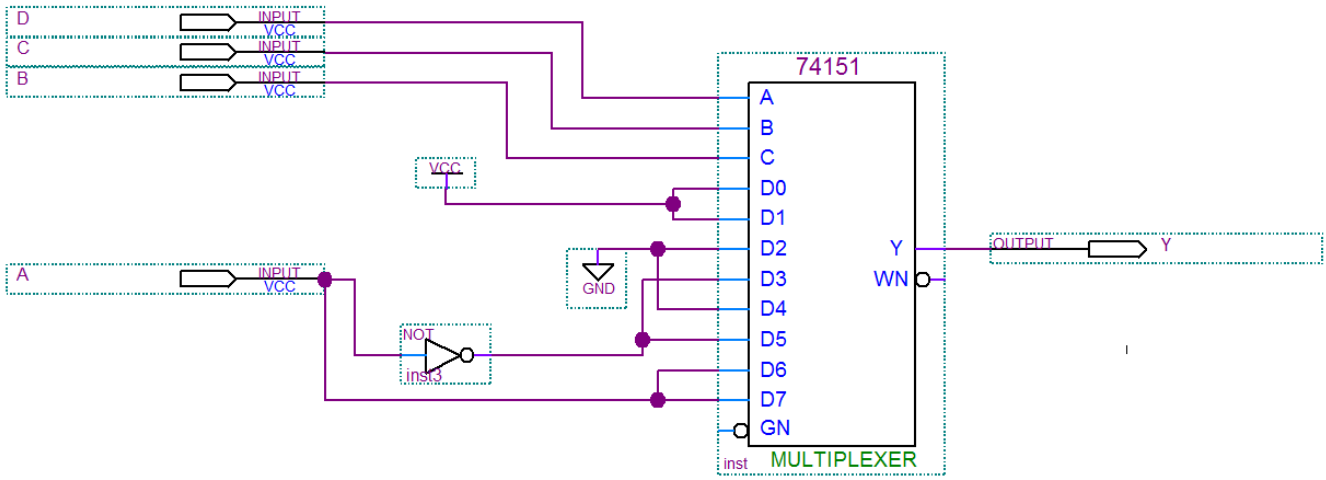


4. Test the functionality of the designed circuit using switches and LEDs on the FPGA board.

Implement the following function using an 8:1 MUX

$$F(A,B,C,D)=\sum(0,1,3,5,8,9,14,15)$$

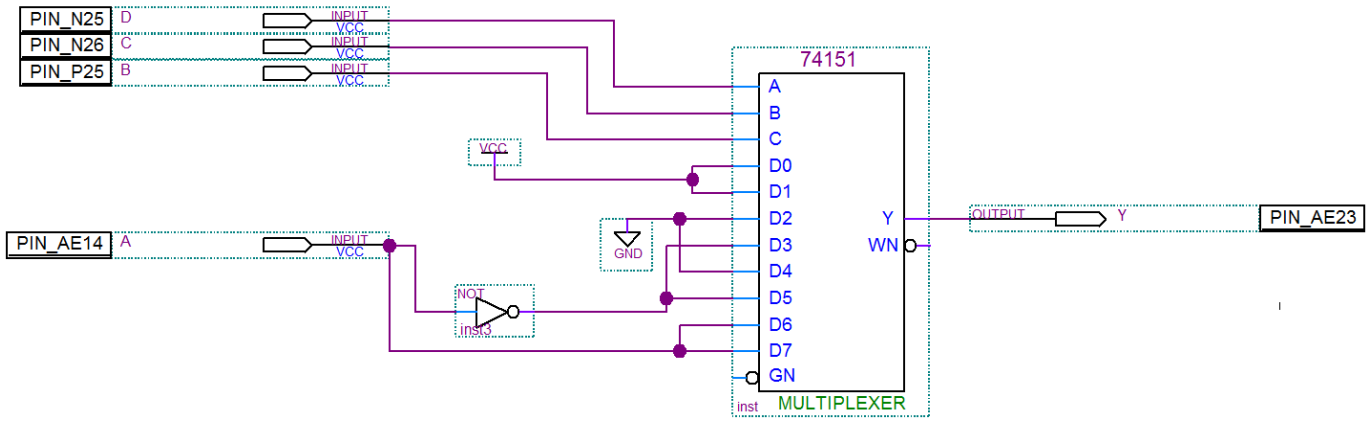
1. Create a new project and create a new block diagram/schematic file. After completing the schematic, it should look like the following:



2. Compile and simulate the schematic. If everything is ok, assign pins as follows:

INPUT			OUTPUT		
Signal	**Switch No.**	**Pin No.**	**Signal**	**LED No.**	**Pin No.**
A	SW3	PIN_AE14	Y	LEDR0	PIN_AE23
B	SW2	PIN_P25			
C	SW1	PIN_N26			
D	SW0	PIN_N25			

3. After assigning pins, the final schematic should look like the following one:



4. Test the functionality of the designed circuit using switches and LEDs on the FPGA board.

Report:

1. Implement a Full Adder using an 8:1 MUX.
2. Repeat 1 using two 4:1 MUX and basic gates.
3. How can you implement a 4:1 MUX using only three 2:1 MUX?

Experiment: 7
Experiment name: *Implementation of Demultiplexers and Priority Encoders.*

Introduction:
Demultiplexer
A Demultiplexer does the opposite function of multiplexers. A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit values of n selection lines. The output channel can be selected depending on the combination of selection bits.

Priority Encoder
A priority encoder is an encoder circuit that includes the priority function. **Priority Encoder** includes the necessary logic to ensure that when two or more inputs are activated, the output code will correspond to the highest-numbered input. The truth table for a 4 to 2 priority encoder is given in Table. The X's are don't-care conditions that designate the fact that the binary value may be equal either to 0 or 1. Input D_3 has the highest priority; so regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3). D_2 has the next priority level. The output is 10 if $D_2 = 1$ provided that $D_3 = 0$, regardless of the values of the other two lower-priority inputs. The output for D_1 is generated only if higher-priority inputs are 0, and so on down the priority level. A valid-output indicator, designated by V, is set to 1 only when one or more of the inputs are equal to 1. If all inputs are 0, V is equal to 0, and the other two outputs of the circuit are not used.

Truth table of a Priority Encoder

Inputs					Outputs		
D_3	D_2	D_1	D_0		x	y	V
0	0	0	0		X	X	0
0	0	0	1		0	0	1
0	0	1	X		0	1	1
0	1	X	X		1	0	1
1	X	X	X		1	1	1

Caution:
1. Remember to properly identify the pin numbers so that no accidents occur.
2. Properly bias the ICs with appropriate voltages to appropriate pins.

Equipment:
1. Trainer Board
2. IC 7432, 7408, 7404
3. Microprocessor Data handbook.

Job 1:

Implementation of a one to four way Demultiplexer, (1:4 DEMUX) with basic gates.

Procedure:

1. Write the truth table for one to four way DEMUX.

S_1	S_0	I_0	I_1	I_2	I_3

2. Write the Boolean function for the output logic.
3. Draw the logic diagram to implement the Boolean function.

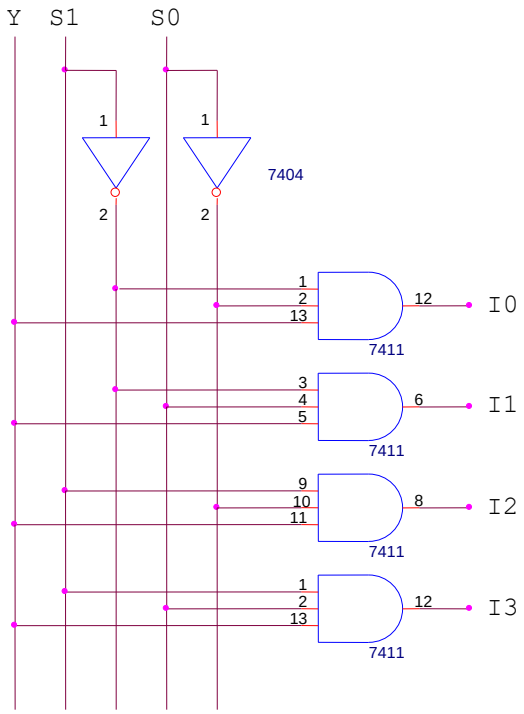


Figure 7.1: Logic diagram of the Demultiplexer

4. Select ICs from the equipment list.
5. Observe and note the output logic for all combination of inputs.

Job 2:

Implement a 4×2 *priority encoder with basic gates.*

Procedure:

1. Write the truth table for 4×2 priority encoder.

D_3	D_2	D_1	D_0	X	Y	V

2. Write the Boolean function for the output logic.
3. Simplify the Boolean function using K-map.
4. Draw the logic diagram to implement the simplified Boolean function.

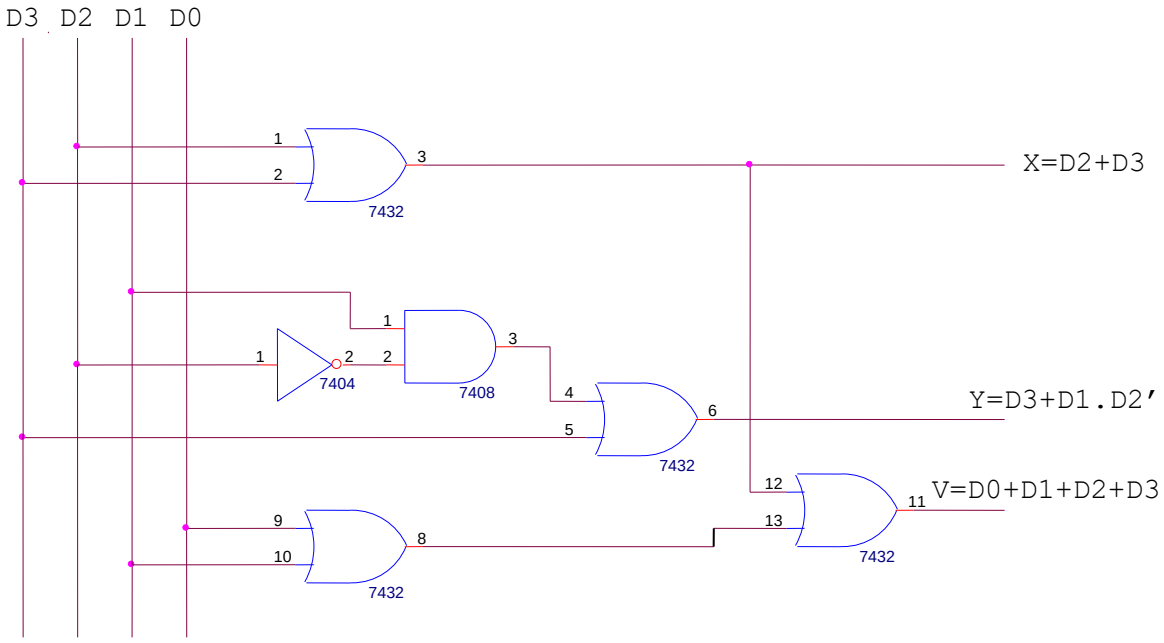


Figure 7.2: Logic diagram of the Priority Encoder

5. Select ICs from the equipment list.
6. Observe and note the output logic for all combination of inputs.

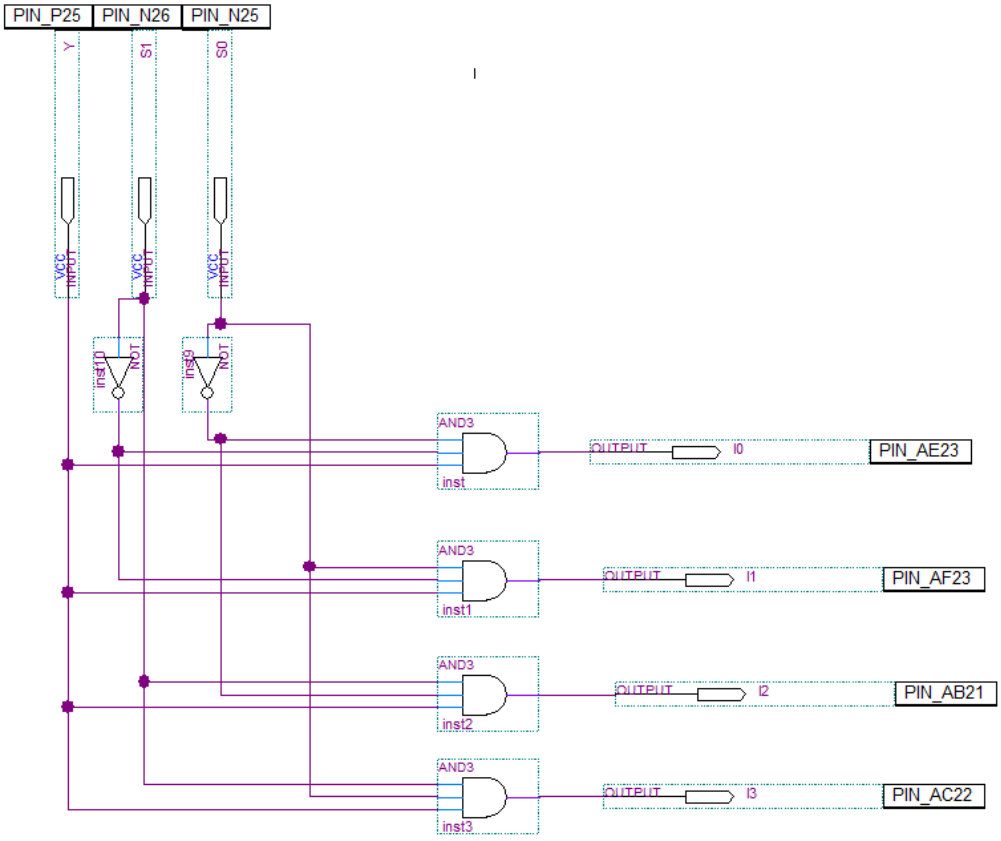
Procedure for FPGA:

DEMUX

1. Create a new project and create a new block diagram/schematic file.
2. Compile and simulate the schematic. If everything is ok, assign pins as follows:

INPUT			OUTPUT		
Signal	**Switch No.**	**Pin No.**	**Signal**	**LED No.**	**Pin No.**
S0	SW0	PIN_N25	I0	LEDR0	PIN_AE23
S1	SW1	PIN_N26	I1	LEDR1	PIN_AF23
Y	SW2	PIN_P25	I2	LEDR2	PIN_AB21
			I3	LEDR3	PIN_AC22

3. After assigning pins, the final schematic should look like the following one:



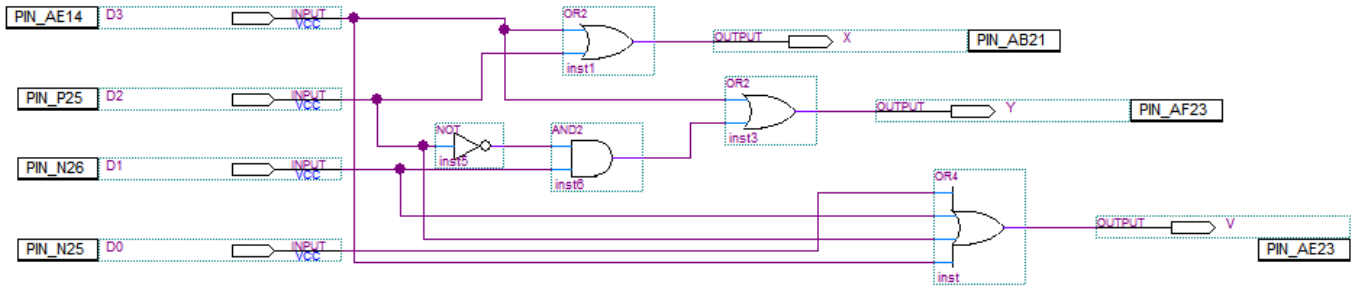
4. Test the functionality of the designed circuit using switches and LEDs on the FPGA board.

Priority Encoder

1. Create a new project and create a new block diagram/schematic file.
2. Compile and simulate the schematic. If everything is ok, assign pins as follows:

INPUT			OUTPUT		
Signal	**Switch No.**	**Pin No.**	**Signal**	**LED No.**	**Pin No.**
D0	SW0	PIN_N25	V	LEDR0	PIN_AE23
D1	SW1	PIN_N26	Y	LEDR1	PIN_AF23
D2	SW2	PIN_P25	X	LEDR2	PIN_AB21
D3	SW3	PIN_AE14			

3. After assigning pins, the final schematic should look like the following one:



4. Test the functionality of the designed circuit using switches and LEDs on the FPGA board

Experiment: 8
Experiment name: *Design of Flip-flop using basic gates.*

Caution:

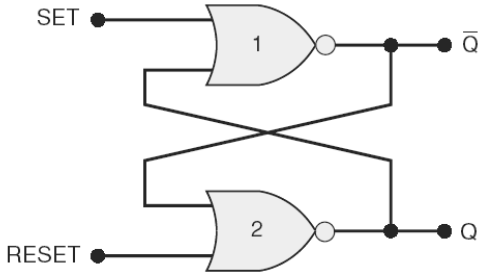
1. Remember to properly identify the pin numbers so that no accidents occur.
2. Properly bias the ICs with appropriate voltages to appropriate pins.

Equipment:

1. Trainer Board
2. IC 7400, 7402, 7432, 7408, 7404
3. Microprocessor Data handbook

Nor Gate Latch

Two cross-coupled NOR gates can be used as a **NOR gate latch**. The arrangement, is similar to the NAND latch except that the Q and Q' outputs have reversed positions.

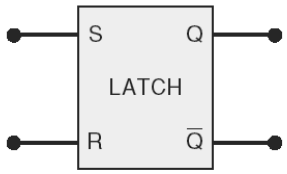


(a)

Set	Reset	Output
0	0	No change
1	0	Q = 1
0	1	Q = 0
1	1	Invalid*

*Produces Q = Q̄ = 0.

(b)



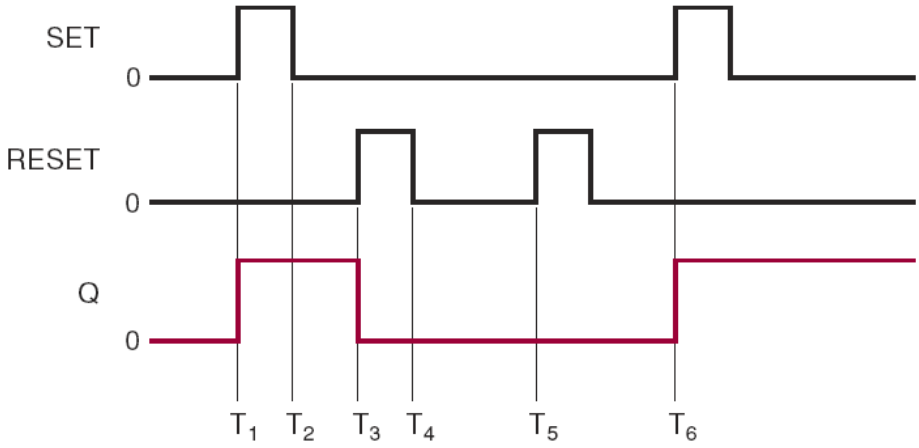
(c)

The analysis of the operation of the NOR latch can be performed in exactly the same manner as for the NAND latch. The results are given in the function table are summarized as follows:

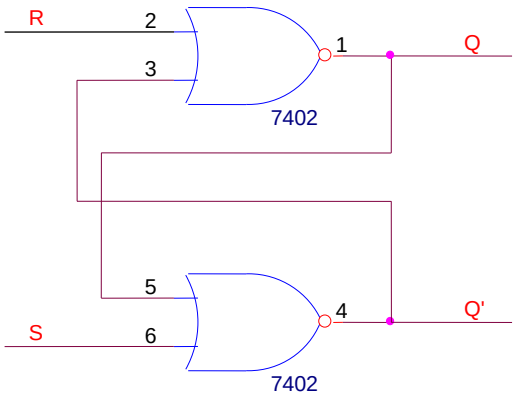
1. SET = RESET = 0. This is the normal resting state for the NOR latch, and it has no effect on the output state. Q and Q' will remain in whatever state they were in prior to the occurrence of this input condition.
2. SET = 1, RESET = 0.this will always set $Q=1$, where it will remain even after SET returns to 0.
3. RESET = 1. SET = 0, this will always clear Q=0, where it will remain even after RESET returns to 0.
4. SET = 1, RESET = 1, this condition tries to set and reset the latch at the same time, and it produces $Q = Q' = 0$. If the inputs are returned to 0 simultaneously, the resulting output state is unpredictable. This input condition should not be used.

The NOR gate latch operates exactly like the NAND latch except that the SET and RESET inputs are active-HIGH rather than active-LOW, and the normal resting state is Q will be set HIGH by a HIGH pulse on the SET input, and it will be cleared LOW by a HIGH pulse on the RESET input.

Timing Diagram



Logic Diagram



Procedure:

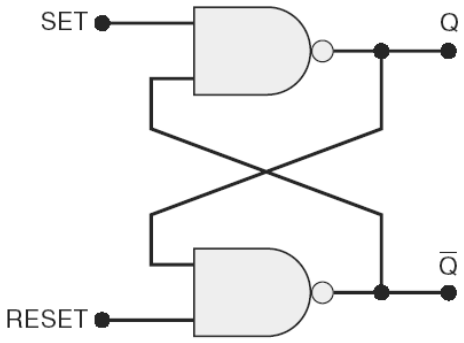
1. Draw the logic diagram to implement SR Flip-flop.
2. Fill up the table with different combination of inputs.

S	R	Q	Q'
1	0		
0	0		
0	1		
0	0		
1	1		

3. Observe the combination for which no change and invalid or race conditions arise.

NAND Gate Latch

The most basic FF circuit can be constructed from either two NAND gates or two NOR gates. The NAND gate version, called a **NAND gate latch** or simply a **latch**, is shown in Figure 5-3(a). The two NAND gates are cross-coupled so that the output of NAND-1 is connected to one of the inputs of NAND-2, and vice versa. The gate outputs, labeled Q and Q', respectively, are the latch outputs.



(a)

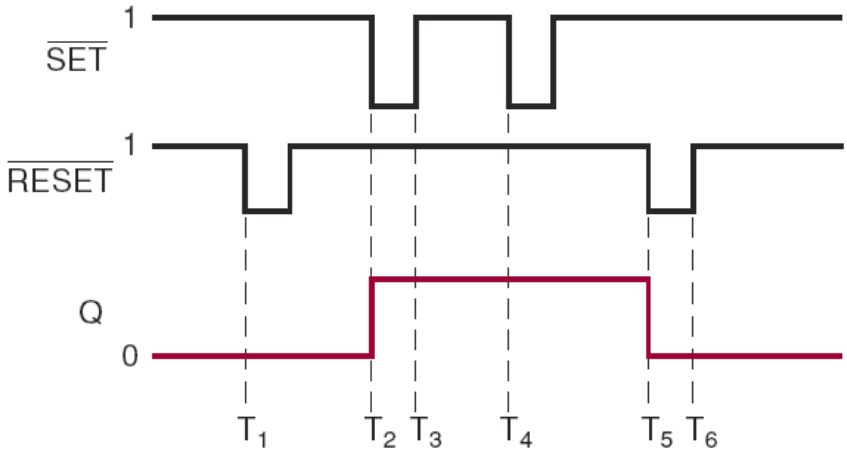
Set	Reset	Output
1	1	No change
0	1	Q = 1
1	0	Q = 0
0	0	Invalid*

*Produces Q = $\overline{Q}$ = 1.

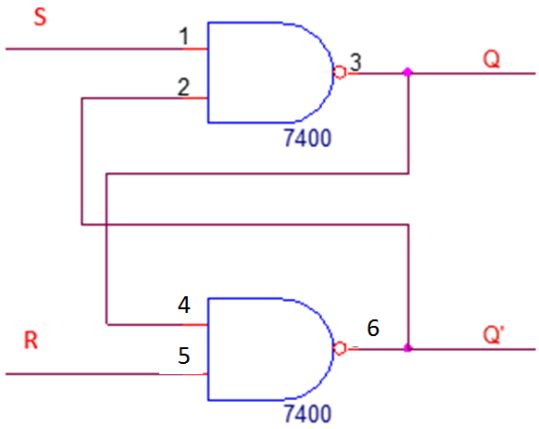
(b)

1. SET = RESET = 1. This condition is the normal resting state, and it has no effect on the output state. The Q and Q' outputs will remain in whatever state they were in prior to this input condition
2. SET = 0, RESET = 1.this will always set $Q=1$, where it will remain even after RESET returns to 0.
3. SET = 1. RESET = 0, this will always clear Q=0, where the output will remain even after RESET returns HIGH. This is called clearing or resetting the latch.
4. SET = 0, RESET = 0, this condition tries to set and reset the latch at the same time, and it produces $Q = Q' = 1$. If the inputs are returned to 0 simultaneously, the resulting output state is unpredictable. This input condition should not be used.

Timing Diagram



Logic Diagram



Procedure:

1. Draw the logic diagram to implement SR Flip-flop.
2. Fill up the table with different combination of inputs.

S	R	Q	Q'
1	0		
0	0		
0	1		
0	0		
1	1		

3. Observe the combination for which no change and invalid or race conditions arise.

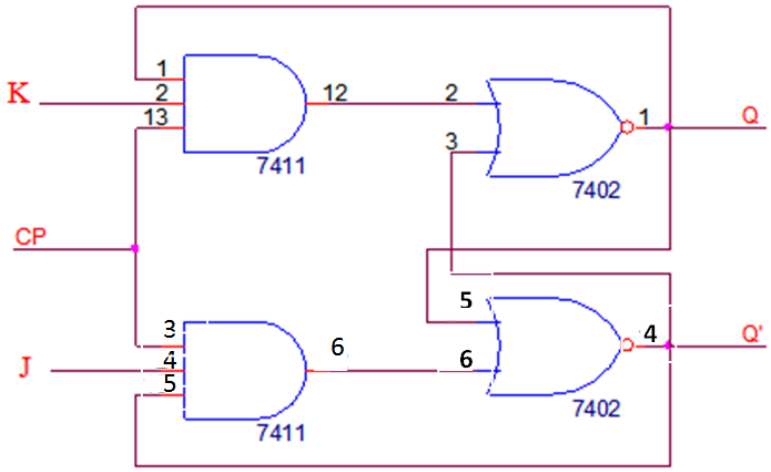
JK FlipFlop

A JK flip-flop is a refinement of the RS flip-flop in that the indeterminate state of the RS type is defined in the JK type. Inputs J and K behave like inputs S and R to set and clear the flip-flop, respectively. The input marked J is for set and the input marked K is for reset. When both inputs J and K are equal to 1, the flip-flop switches to its complement state, that is, if Q = 1, it switches to Q = 0, and vice versa.

A JK flip-flop constructed with two cross-coupled NOR gates and two AND gates is shown in Figure. Output Q is ANDed with K and CP inputs so that the flip-flop is cleared during a clock pulse only if Q was previously 1. Similarly, output Q' is ANDed with J and CP inputs so that the flop-flop is set with a clock pulse only when Q' was previously 1. When both J and K are 1, the input pulse is transmitted through one AND gate only: the one whose input is connected to the flip-flop output that is presently equal to 1. Thus, if Q = 1, the output of the upper AND gate becomes 1 upon application of the clock pulse, and the flip-flop is cleared. If Q' = 1, the output of the lower AND gate becomes 1 and the flip-flop is set. In either case, the output state of the flip-flop is complemented. The behavior of the JK flip-flop is demonstrated in the characteristic table.

It is very important to realize that because of the feedback connection in the JK flipflop, a CP pulse that remains in the 1 state while both J and K are equal to 1 will cause the output to

complement again and repeat complementing until the pulse goes back to 0. To avoid this undesirable operation, the clock pulse must have a time duration that is shorter than the propagation delay time of the flip-flop. This is a restrictive requirement, since the operation of the circuit depends on the width of the pulse. For this reason, JK flip-flops are never constructed as shown in Figure. The restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction, as discussed in the next section. The same reasoning applies to the T flip-flop.



Procedure:

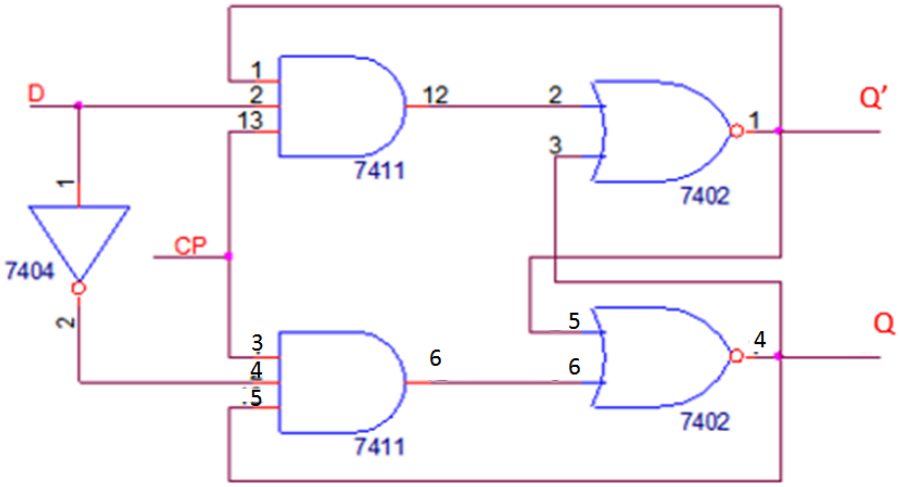
1. Draw the logic diagram to implement J-K Flip-flop.
2. Fill up the table with different combination of inputs.

Q	J	K	$Q(t+1)$
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

3. Observe the combination for which no change and invalid or race conditions arise.

D FlipFlop

Design of a D Flip-flop from a J-K Flip-flop.



Procedure:

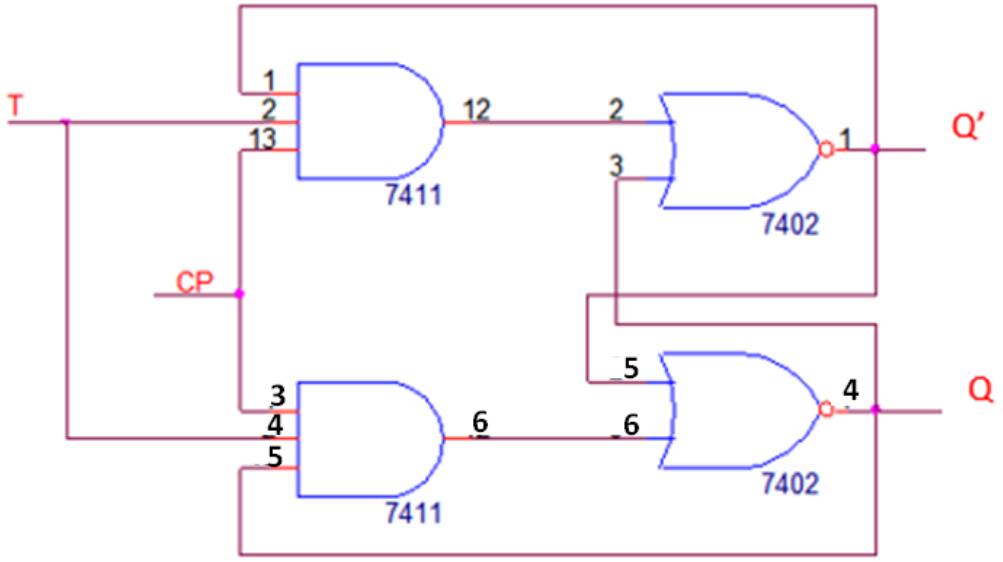
1. Draw the logic diagram to implement D Flip-flop.
2. Fill up the table with different combination of inputs.

Q	D	$Q(t+1)$
0	0	
0	1	
1	0	
1	1	

3. Observe the combination for which no change and invalid or race conditions arise.

T Flipflop

Design of a T Flip-flop from a J-K Flip-flop.



Procedure:

1. Draw the logic diagram to implement T Flip-flop.
2. Fill up the table with different combination of inputs.

Q	T	$Q(t+1)$
0	0	
0	1	
1	0	
1	1	

3. Observe the output logic.

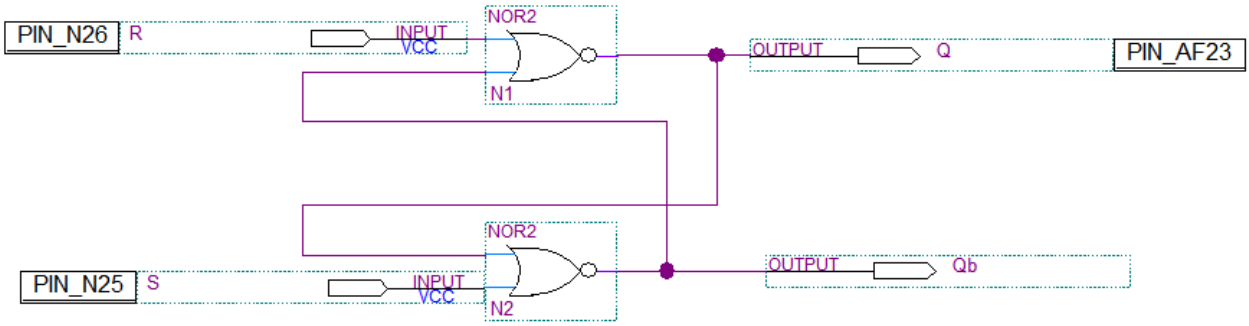
Procedure for FPGA:

SR Latch

1. Create a new project and create a new block diagram/schematic file
2. Compile and simulate the schematic. If everything is ok, assign pins as follows:

INPUT			OUTPUT		
Signal	**Switch No.**	**Pin No.**	**Signal**	**LED No.**	**Pin No.**
R	SW1	PIN_N26	Q	LEDR1	PIN_AF23
S	SW0	PIN_N25	Q0	LEDR0	PIN_AE23

3. After assigning pins, the final schematic should look like the following one:



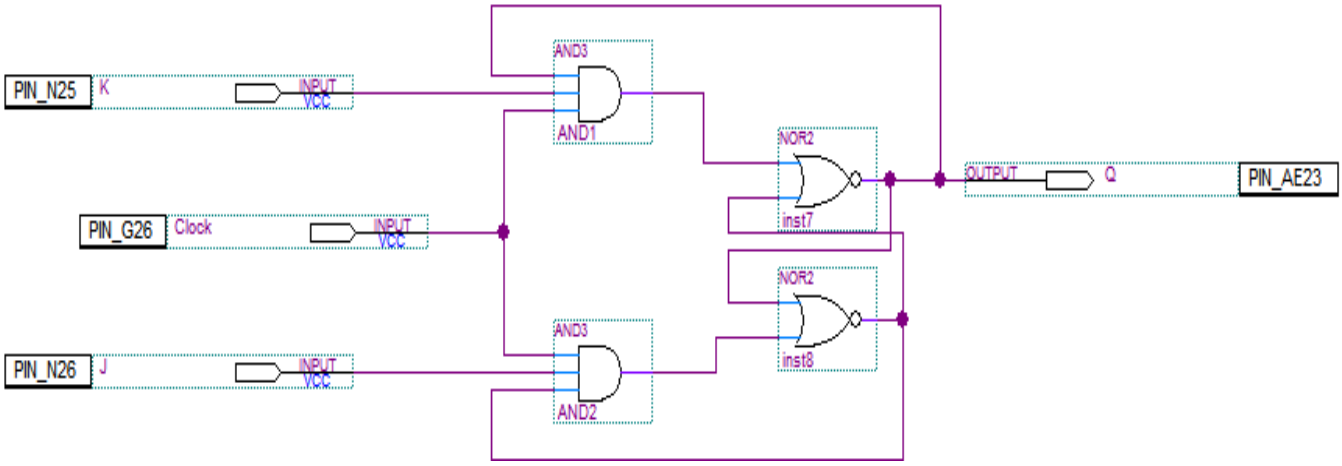
4. Test the functionality of the designed circuit using switches and LEDs on the FPGA board.

JK Flipflop

1. Create a new project and create a new block diagram/schematic file.
2. Compile and simulate the schematic. If everything is ok, assign pins as follows:

INPUT			OUTPUT		
Signal	**Switch No.**	**Pin No.**	**Signal**	**LED No.**	**Pin No.**
Clock	Pushbutton[0]	PIN_G26	Q	LEDR0	PIN_AE23
J	SW1	PIN_N26			
K	SW0	PIN_N25			

3. After assigning pins, the final schematic should look like the following one:



4. Test the functionality of the designed circuit using switches and LEDs on the FPGA board.

ANNEXURE I

Installing USB-Blaster driver software on Windows 7

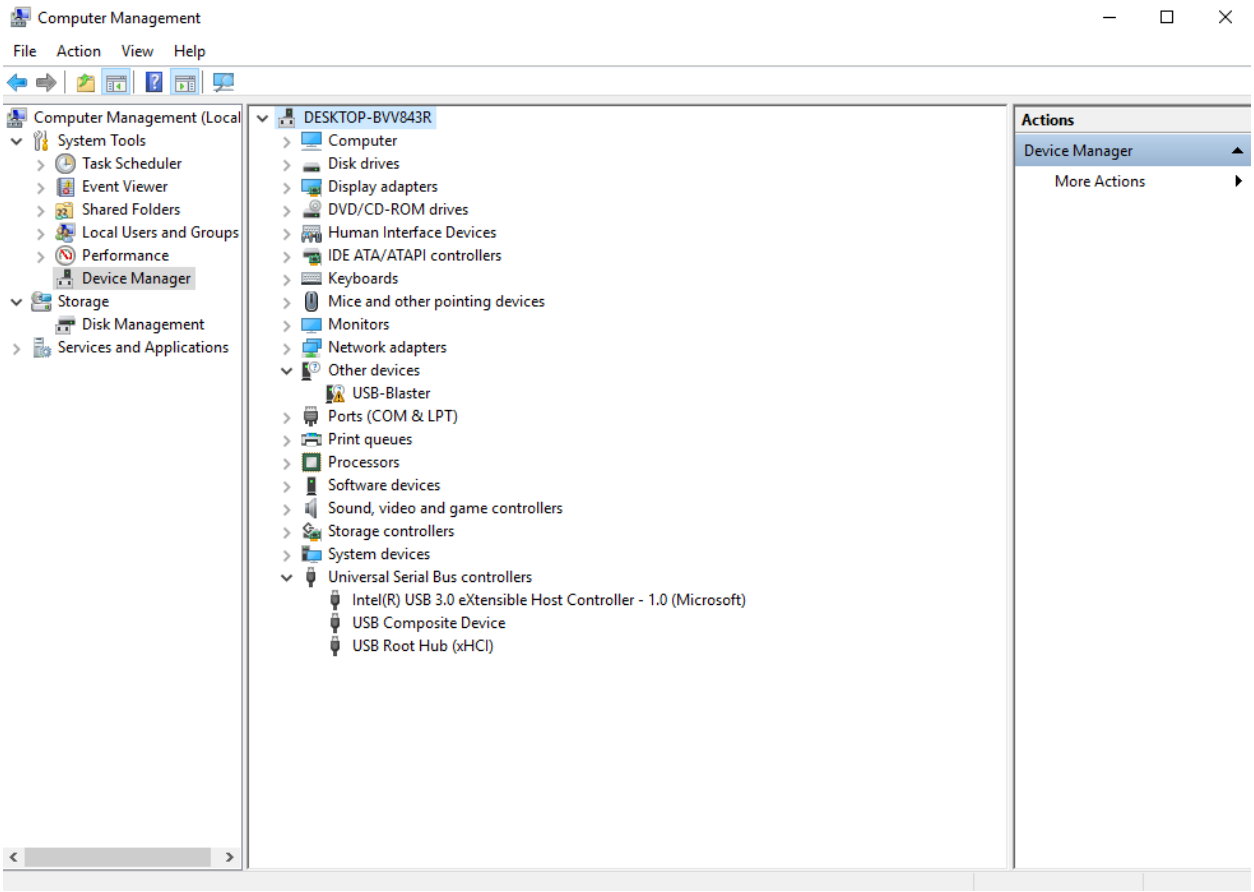
1. Set the **RUN/PROG** switch to the **RUN** position.

2. Connect the supplied USB cable to the **USB-Blaster port** of the FPGA and to a USB port of the PC. Also connect the 9V power supply adapter and turn the power switch **ON**.

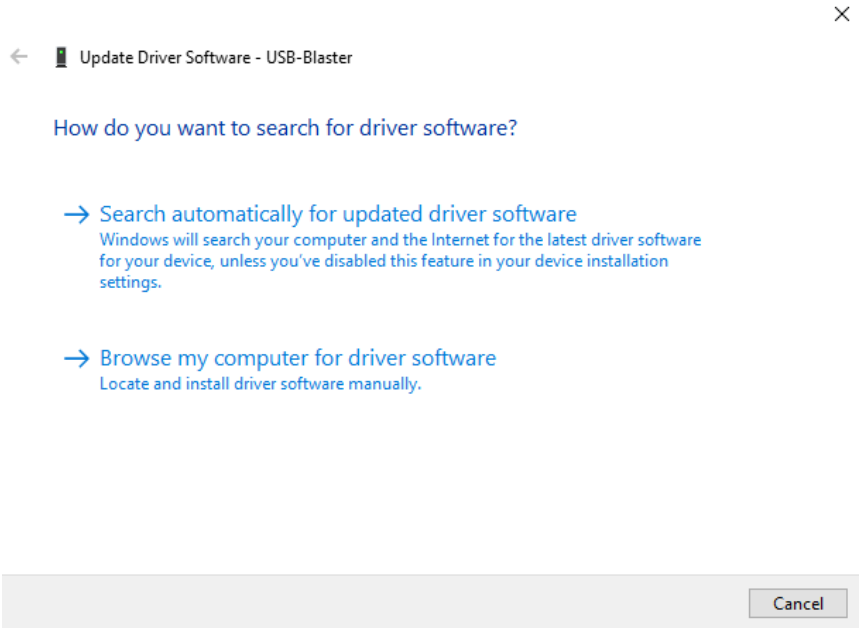
At this point you should observe the following:

- All user LEDs are flashing
- All 7-segment displays are cycling through the numbers 0 to F
- The LCD display shows Welcome to the Altera DE2 Board

3. Open **Device Manager**.



4. Note that, **USB-Blaster** is listed under **Other devices**. Right click on it and select **Update Driver Software**. **Update Driver Software –USB-Blaster** window will open up.

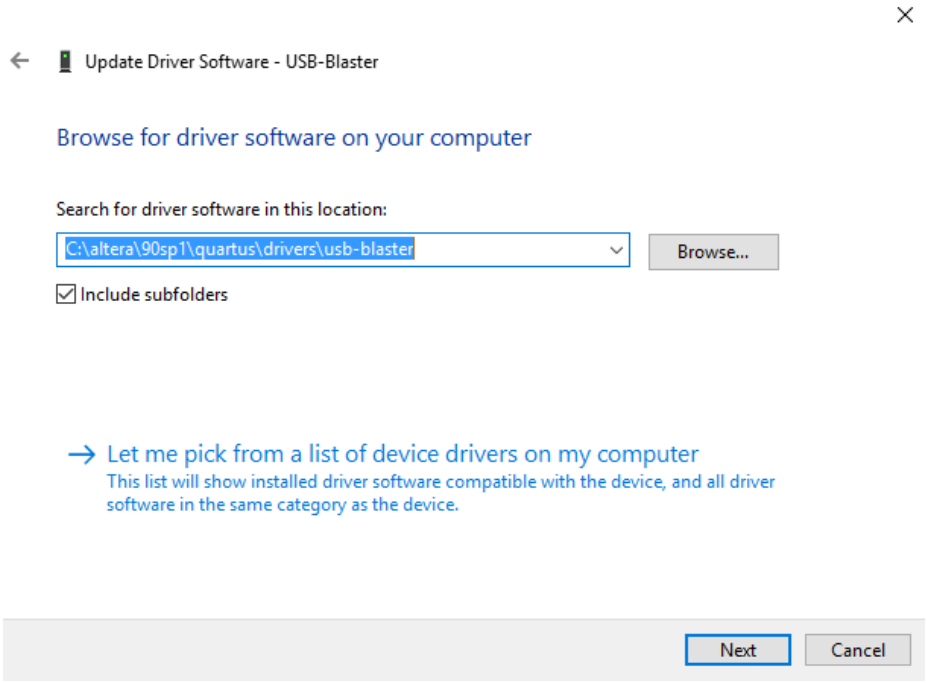


5. Select **Browse my computer for driver software**.

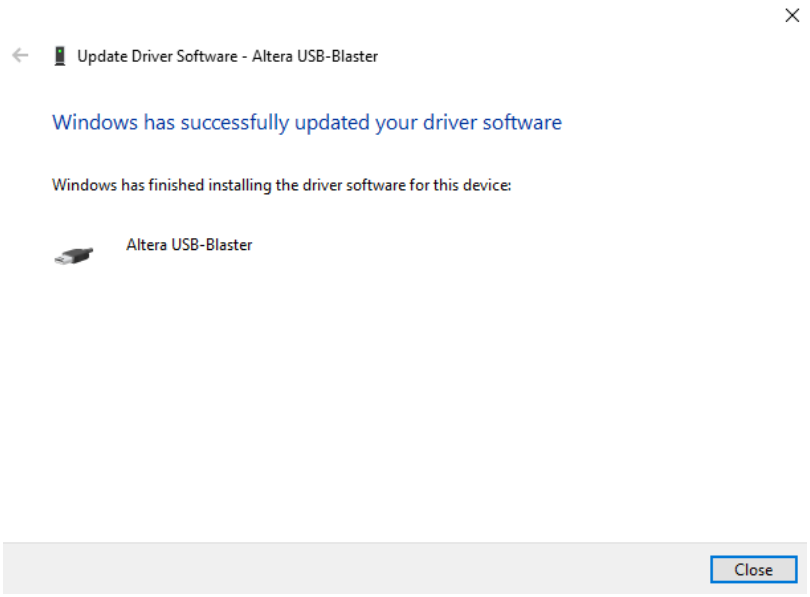
6. Find the location of **USB-Blaster driver software** from the installation directory of **Quartus II**. It will be under

<your installation directory>\altera\90sp1\quartus\drivers\usb-blaster

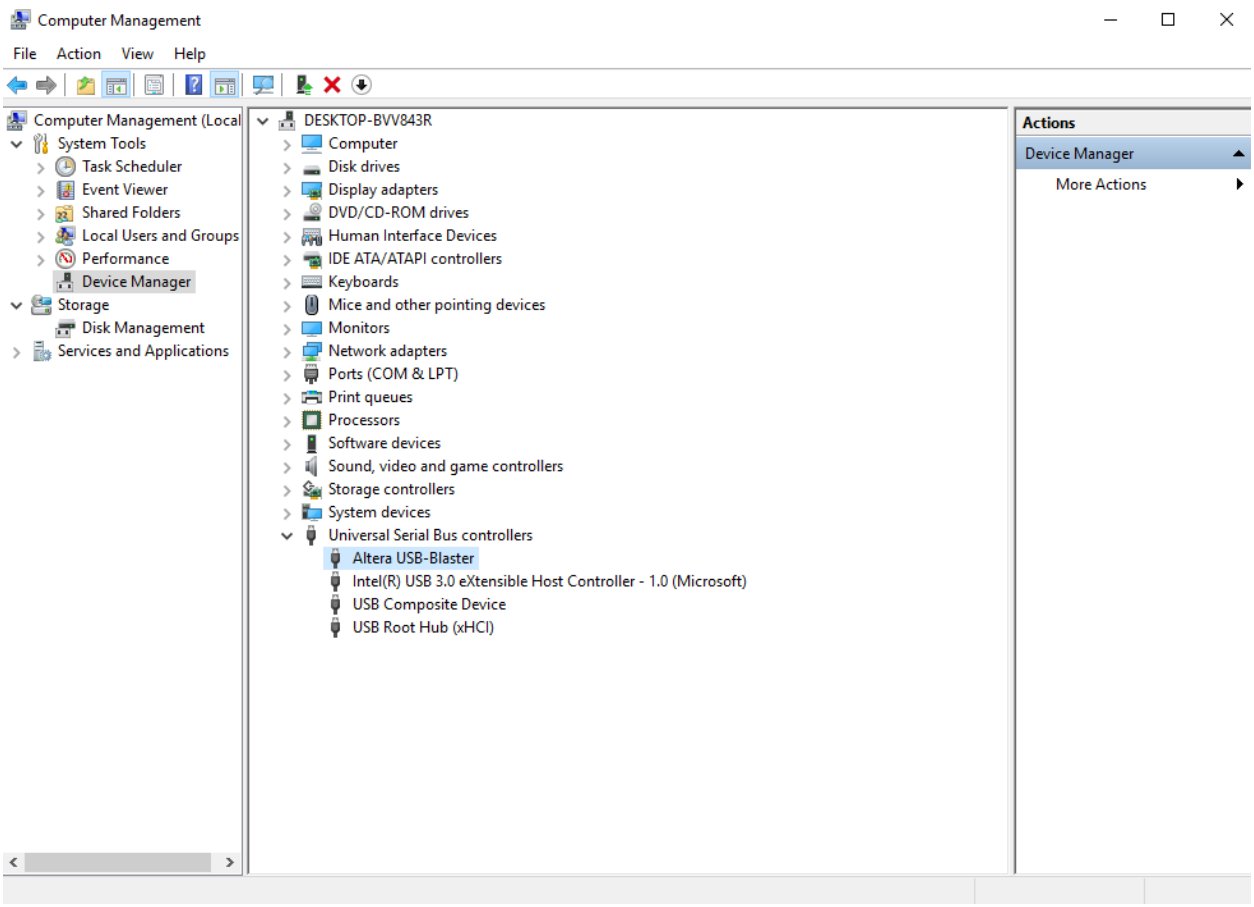
for**Quartus II 9.0 sp1 web edition**.



7. Select **Install this software anyway** if Windows Security prompt appears.



8. After successful installation, **Altera USB-Blaster** will appear under **Universal Serial Bus controllers** in **Device Manager**window:


Computer Management
File Action View Help
Computer Management (Local
System Tools
Task Scheduler
Event Viewer
Shared Folders
Local Users and Groups
Performance
Device Manager
Storage
Disk Management
Services and Applications
DESKTOP-BVV843R
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
Human Interface Devices
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
Print queues
Processors
Software devices
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers
Altera USB-Blaster
Intel(R) USB 3.0 eXtensible Host Controller - 1.0 (Microsoft)
USB Composite Device
USB Root Hub (xHCI)
Actions
Device Manager
More Actions

ANNEXURE II

Using the LEDs and Switches

The DE2 board provides four pushbutton switches. Each of these switches is debounced using a Schmitt Trigger circuit. The four outputs called *KEY0*, …, *KEY3* of the Schmitt Trigger device are connected directly to the Cyclone II FPGA. Each switch provides a high logic level (3.3 volts) when it is not pressed, and provides a low logic level (0 volts) when depressed. Since the pushbutton switches are debounced, they are appropriate for use as clock or reset inputs in a circuit. There are also 18 toggle switches (sliders) on the DE2 board. These switches are not debounced, and are intended for use as level-sensitive data inputs to a circuit. Each switch is connected directly to a pin on the Cyclone II FPGA. When a switch is in the DOWN position (closest to the edge of the board) it provides a low logic level (0 volts) to the FPGA, and when the switch is in the UP position it provides a high logic level (3.3 volts).

There are 27 user-controllable LEDs on the DE2 board. Eighteen red LEDs are situated above the 18 toggle switches, and eight green LEDs are found above the pushbutton switches (the 9th green LED is in the middle of the 7-segment displays). Each LED is driven directly by a pin on the Cyclone II FPGA; driving its associated pin to a high logic level turns the LED on, and driving the pin low turns it off.

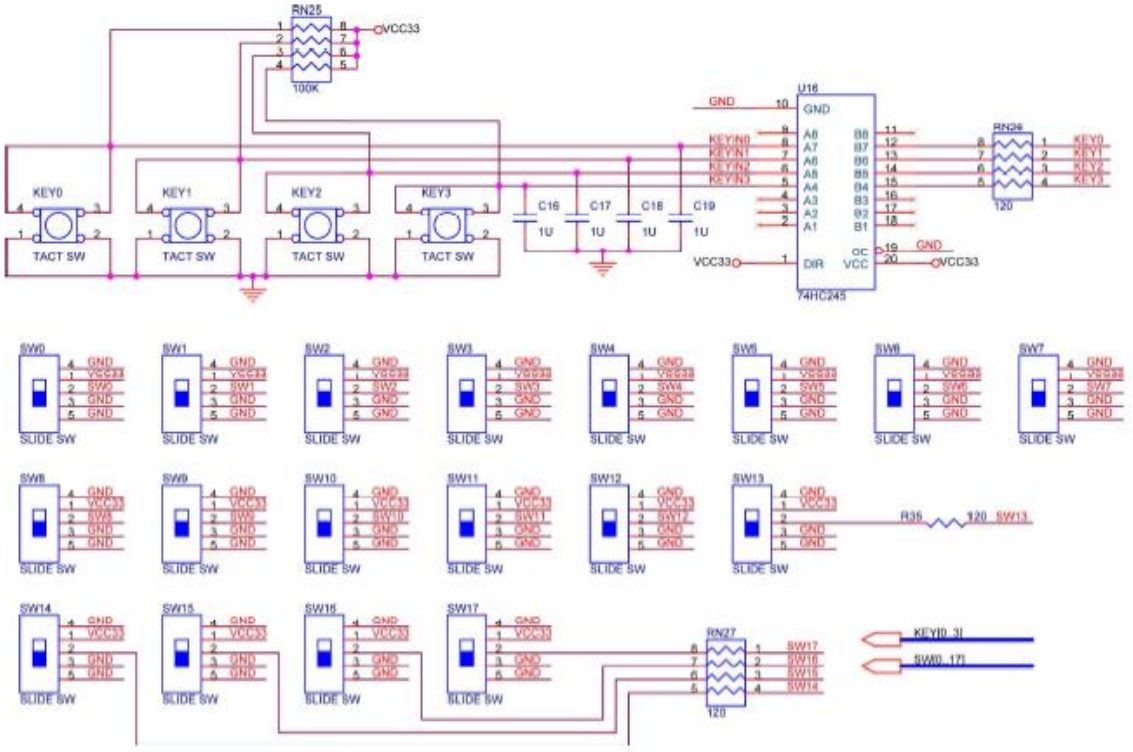


Figure1: Schematic diagram of push button and toggle switches

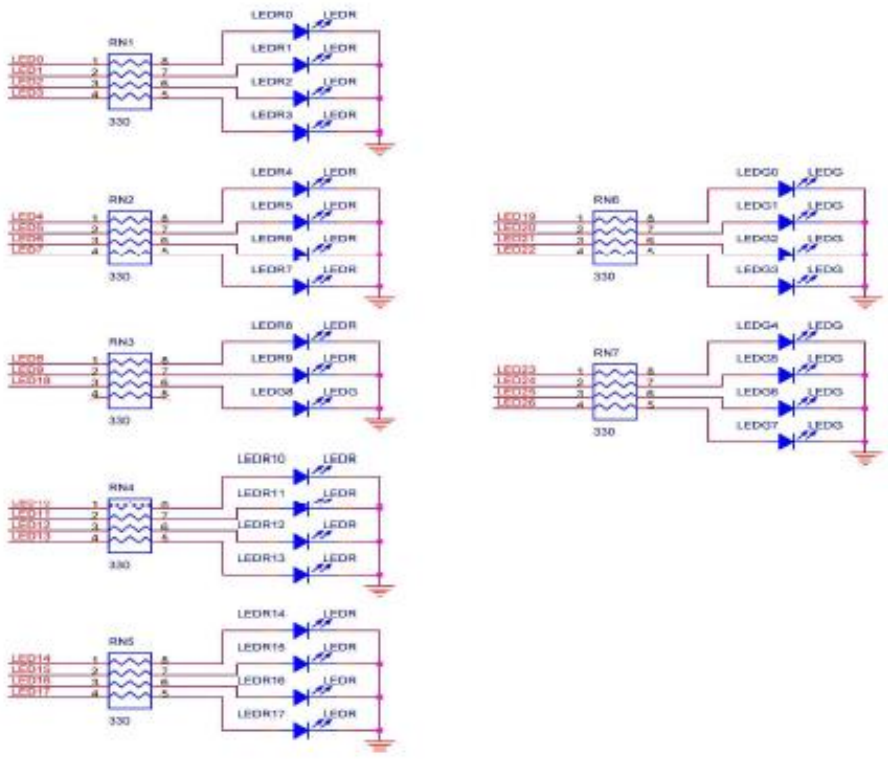


Figure 2: Schematic diagram of LEDs

Signal Name	FPGA Pin No.	Description
SW[0]	PIN_N25	Toggle Switch[0]
SW[1]	PIN_N26	Toggle Switch[1]
SW[2]	PIN_P25	Toggle Switch[2]
SW[3]	PIN_AE14	Toggle Switch[3]
SW[4]	PIN_AF14	Toggle Switch[4]
SW[5]	PIN_AD13	Toggle Switch[5]
SW[6]	PIN_AC13	Toggle Switch[6]
SW[7]	PIN_C13	Toggle Switch[7]
SW[8]	PIN_B13	Toggle Switch[8]
SW[9]	PIN_A13	Toggle Switch[9]
SW[10]	PIN_N1	Toggle Switch[10]
SW[11]	PIN_P1	Toggle Switch[11]
SW[12]	PIN_P2	Toggle Switch[12]
SW[13]	PIN_T7	Toggle Switch[13]
SW[14]	PIN_U3	Toggle Switch[14]
SW[15]	PIN_U4	Toggle Switch[15]
SW[16]	PIN_V1	Toggle Switch[16]
SW[17]	PIN_V2	Toggle Switch[17]

Table1: Pin Assignments for toggle switches

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_G26	Pushbutton[0]
KEY[1]	PIN_N23	Pushbutton[1]
KEY[2]	PIN_P23	Pushbutton[2]
KEY[3]	PIN_W26	Pushbutton[3]

Table 2: Pin Assignments for push button switches

Signal Name	FPGA Pin No.	Description
LEDR[0]	PIN_AE23	LED Red[0]
LEDR[1]	PIN_AF23	LED Red[1]
LEDR[2]	PIN_AB21	LED Red[2]
LEDR[3]	PIN_AC22	LED Red[3]
LEDR[4]	PIN_AD22	LED Red[4]
LEDR[5]	PIN_AD23	LED Red[5]
LEDR[6]	PIN_AD21	LED Red[6]
LEDR[7]	PIN_AC21	LED Red[7]
LEDR[8]	PIN_AA14	LED Red[8]
LEDR[9]	PIN_Y13	LED Red[9]
LEDR[10]	PIN_AA13	LED Red[10]
LEDR[11]	PIN_AC14	LED Red[11]
LEDR[12]	PIN_AD15	LED Red[12]
LEDR[13]	PIN_AE15	LED Red[13]
LEDR[14]	PIN_AF13	LED Red[14]
LEDR[15]	PIN_AE13	LED Red[15]
LEDR[16]	PIN_AE12	LED Red[16]
LEDR[17]	PIN_AD12	LED Red[17]
LEDG[0]	PIN_AE22	LED Green[0]
LEDG[1]	PIN_AF22	LED Green[1]
LEDG[2]	PIN_W19	LED Green[2]
LEDG[3]	PIN_V18	LED Green[3]
LEDG[4]	PIN_U18	LED Green[4]
LEDG[5]	PIN_U17	LED Green[5]
LEDG[6]	PIN_AA20	LED Green[6]
LEDG[7]	PIN_Y18	LED Green[7]
LEDG[8]	PIN_Y12	LED Green[8]

Table 3: Pin Assignments for LEDs

Using the 7-segment Displays

The DE2 Board has eight 7-segment displays. These displays are arranged into two pairs and a group of four, with the intent of displaying numbers of various sizes. As indicated in the schematic in Figure 4.6, the seven segments are connected to pins on the Cyclone II FPGA. Applying a low logic level to a segment causes it to light up, and applying a high logic level

turns it off. Each segment in a display is identified by an index from 0 to 6, with the positions given in Figure 4.7. Note that the dot in each display is unconnected and cannot be used. Table 4.4 shows the assignments of FPGA pins to the 7-segment displays.

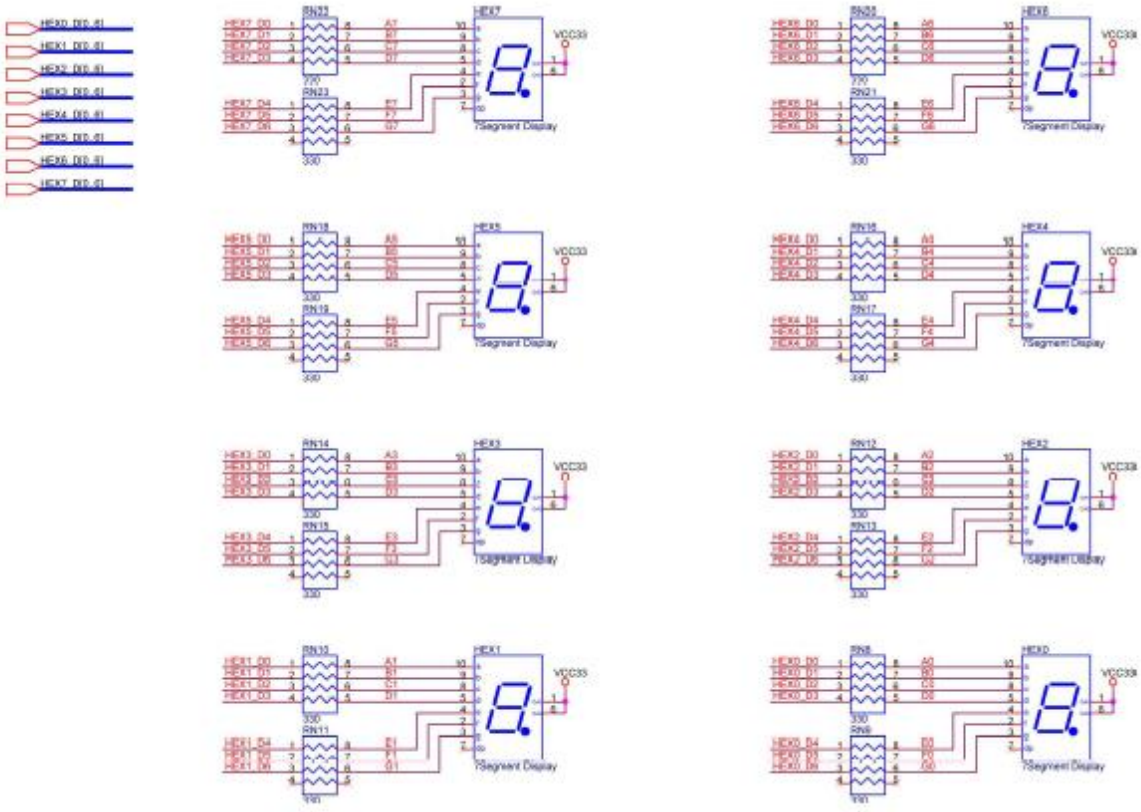


Figure 3: Schematic diagram of 7 segment displays

Signal Name	FPGA Pin No.	Description
HEX0[0]	PIN_AF10	Seven Segment Digit 0[0]
HEX0[1]	PIN_AB12	Seven Segment Digit 0[1]
HEX0[2]	PIN_AC12	Seven Segment Digit 0[2]
HEX0[3]	PIN_AD11	Seven Segment Digit 0[3]
HEX0[4]	PIN_AE11	Seven Segment Digit 0[4]
HEX0[5]	PIN_V14	Seven Segment Digit 0[5]
HEX0[6]	PIN_V13	Seven Segment Digit 0[6]
HEX1[0]	PIN_V20	Seven Segment Digit 1[0]
HEX1[1]	PIN_V21	Seven Segment Digit 1[1]
HEX1[2]	PIN_W21	Seven Segment Digit 1[2]
HEX1[3]	PIN_Y22	Seven Segment Digit 1[3]
HEX1[4]	PIN_AA24	Seven Segment Digit 1[4]
HEX1[5]	PIN_AA23	Seven Segment Digit 1[5]
HEX1[6]	PIN_AB24	Seven Segment Digit 1[6]
HEX2[0]	PIN_AB23	Seven Segment Digit 2[0]
HEX2[1]	PIN_V22	Seven Segment Digit 2[1]
HEX2[2]	PIN_AC25	Seven Segment Digit 2[2]
HEX2[3]	PIN_AC26	Seven Segment Digit 2[3]
HEX2[4]	PIN_AB26	Seven Segment Digit 2[4]
HEX2[5]	PIN_AB25	Seven Segment Digit 2[5]
HEX2[6]	PIN_Y24	Seven Segment Digit 2[6]
HEX3[0]	PIN_Y23	Seven Segment Digit 3[0]
HEX3[1]	PIN_AA25	Seven Segment Digit 3[1]
HEX3[2]	PIN_AA26	Seven Segment Digit 3[2]
HEX3[3]	PIN_Y26	Seven Segment Digit 3[3]
HEX3[4]	PIN_Y25	Seven Segment Digit 3[4]
HEX3[5]	PIN_U22	Seven Segment Digit 3[5]
HEX3[6]	PIN_W24	Seven Segment Digit 3[6]
HEX4[0]	PIN_U9	Seven Segment Digit 4[0]
HEX4[1]	PIN_U1	Seven Segment Digit 4[1]
HEX4[2]	PIN_U2	Seven Segment Digit 4[2]
HEX4[3]	PIN_T4	Seven Segment Digit 4[3]
HEX4[4]	PIN_R7	Seven Segment Digit 4[4]
HEX4[5]	PIN_R6	Seven Segment Digit 4[5]
HEX4[6]	PIN_T3	Seven Segment Digit 4[6]

HEX5[0]	PIN_T2	Seven Segment Digit 5[0]
HEX5[1]	PIN_P6	Seven Segment Digit 5[1]
HEX5[2]	PIN_P7	Seven Segment Digit 5[2]
HEX5[3]	PIN_T9	Seven Segment Digit 5[3]
HEX5[4]	PIN_R5	Seven Segment Digit 5[4]
HEX5[5]	PIN_R4	Seven Segment Digit 5[5]
HEX5[6]	PIN_R3	Seven Segment Digit 5[6]
HEX6[0]	PIN_R2	Seven Segment Digit 6[0]
HEX6[1]	PIN_P4	Seven Segment Digit 6[1]
HEX6[2]	PIN_P3	Seven Segment Digit 6[2]
HEX6[3]	PIN_M2	Seven Segment Digit 6[3]
HEX6[4]	PIN_M3	Seven Segment Digit 6[4]
HEX6[5]	PIN_M5	Seven Segment Digit 6[5]
HEX6[6]	PIN_M4	Seven Segment Digit 6[6]
HEX7[0]	PIN_L3	Seven Segment Digit 7[0]
HEX7[1]	PIN_L2	Seven Segment Digit 7[1]
HEX7[2]	PIN_L9	Seven Segment Digit 7[2]
HEX7[3]	PIN_L6	Seven Segment Digit 7[3]
HEX7[4]	PIN_L7	Seven Segment Digit 7[4]
HEX7[5]	PIN_P9	Seven Segment Digit 7[5]
HEX7[6]	PIN_N9	Seven Segment Digit 7[6]

Table 4: Pin diagram for seven segment displays

ANNEXURE III

Complements of a Number

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements for each base-r system: the radix complements and the diminished radix complement. The first is referred to as the r's complement and the second as the (r - 1)'s complement. When the value of the base *r* is substituted in the name, the two types are referred to as the **2's complement and l's complement for binary numbers**, and the **10's complement and 9's complement for decimal numbers**.

Diminished Radix Complement

Given a number *N* in base r having *n* digits, the (r - 1)'s complement of *N* is defined as $(r^n - 1) - N$. For decimal numbers, r = 10 and *r* - 1 = 9, so the 9's complement of *N* is $(10^n - 1) - N$. Now, 10^n represents a number that consists of a single 1 followed by *n 0's.* $10^n - 1$ is a number represented by *n* 9's. For example, if *n* = 4, we have 10^4 = 10,000 and 10^4- 1 = 9999. It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9. Some numerical examples follow.

The 9's complement of 546700 is 999999 - 546700 = 453299.
The 9's complement of 012398 is 999999 - 012398 = 987601.

For binary numbers, r = 2 and r - 1 = 1, so the 1's complement of *N* is $(2^n - 1) - N$. Again, 2^n is represented by a binary number that consists of a 1 followed by *n* 0's. $2^n - 1$ is a binary number represented by *n* 1's. For example, if *n* = 4, we have 2^4 = (0000), and 2^4 - I = (1111). Thus the 1's complement of a binary number is obtained by subtracting each digit from 1. However, when subtracting binary digits from 1, we can have either 1 - 0 = 1 or 1 - 1 = 0, which causes the bit to change from 0 to 1 or from 1 to O. Therefore, the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's. The following are some numerical examples. -

The 1's complement of 1011000 is 0100111.
The 1's complement of 0101101 is 1010010.

The *(r* - 1)'s complement of octal or hexadecimal numbers is obtained by subtracting each digit from 7 or F (decimal 15), respectively.

Radix Complement

The r's complement of an n-digit number, N in base *r* is defined as $r^n - N$ *for* $N \neq 0$. and 0 for *N* = 0. Comparing with the *(r* - 1)'s complement, we note that the r's complement obtained by adding 1 to the *(r* - 1)'s complement since $r^n - N = [(r^n - 1) - N] + 1$. Thus, the 10's complement of decimal 2389 is 7610 + 1 = 7611and is obtained by adding 1 to the 9's-complement value. The 2's complement of binary 101100 is 010011 + 1 = 010100 and is obtained by adding 1 to the 1's-complement value.

Since 10^n is a number represented by a 1 followed by n 0's, $10^n - N$, which is the 10's complement of *N,* can be formed also by leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and subtracting all higher significant digits from 9.

The 10's complement of 012398 is 987602.
The 10's complement of 246700 is 753300.

The 10's complement of the first number is obtained by subtracting 8 from 10 in the least significant position and subtracting all other digits from 9. The 10's complement of the second number is obtained by leaving the two least significant 0's unchanged, subtracting 7 from 10, and subtracting the other three digits from 9.

Similarly, the 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged, and replacing 1's with 0's and 0's with 1's in all other higher significant digits.

The 2's complement of 1101100 is 0010100.

The 2's complement of 0110111 is 1001001.

The 2's complement of the first number is obtained by leaving the two least significant 0's and the first 1 unchanged, and then replacing 1's with 0's and 0's with 1's with other four most-significant digits. The 2's complement of the second number is obtained by leaving the least significant I unchanged and complementing all other digits.

In the previous definitions, it was assumed that the numbers do not have a radix point. If the original number *N* contains a radix point, the point should be removed temporarily in order to form the *r's* or *(r* - I)'s complement. The radix point is then restored to the complemented number in the same relative position. It is also worth mentioning that the complement of the complement restores the number to its original value. The *r's* complement of *N* is $r'' \sim N$. The complement of the complement is $r^n - (r^n - N) = N$, giving back the original number.

REFERENCE

1. Digital logic and computer design by M. Morris Mano.

2. Digital fundamentals by Thomas L. Floyd.

3. Fundamentals of Digital Logic with Verilog Design by Stephen Brown and Zvonko Vranesic.

4. Digital systems principles and applications by Ronald J. Tocci.