

13EC437 ECAD & VLSI Lab
(Lab Manual)
Verilog Programs
For IV Year I Sem ECE

Prepared by Dr. K. Swaminathan
(swami.nitt@gmail.com)

E-CAD AND VLSI LAB :

COURSE OBJECTIVES:

1. To learn the HDL programming language.
2. To learn the simulation of basic gates using the basic programming language.
3. To learn the simulation of combinational circuits using programming language.
4. To learn the simulation of sequential circuits using programming language.
5. To learn the synthesis and layouts of analog and digital CMOS circuits.
6. To develop an ability to simulate and synthesize various digital circuits.

COURSE OUTCOMES:

The students will be able to

1. Simulate various digital circuits.
2. Simulate and synthesize various CMOS circuits.
3. Understand the layout design rules for both static CMOS and dynamic clocked CMOS Circuits.
4. Develop an ability of designing of analog and digital CMOS circuits.

List of Experiments

List of Experiments

Design and implementation of the following CMOS digital/analog circuits using Cadence/ Mentor Graphics/ Synopsys/ GEDA/Equivalent CAD tools. The design shall include Gate-level design. Transistor –level design, Hierarchical design, Verilog HDL/VHDL design, Logic synthesis, Simulation and verification, scaling of CMOS inverter for different technologies, study of secondary effects (temperature, power supply and process corners), circuit optimization with respect to area, performance and/or power, Layout Extraction of parasitics and back annotation, modifications in circuit parameters and layout consumption, DC/ transient analysis, Verification of layouts (DRC, LVS)

E-CAD programs:

Programming can be done using any compiler. Down load the programs on FPGA/CPLD boards and performance testing may be done using pattern generator (32 channels) and logic analyzer apart from verification by simulation with any of the front end tools.

- 1. HDL code to realize all the logic gates**
- 2. Design of 2-to-4 decoder**
- 3. Design of 8-to-3 encoder (without and with parity)**
- 4. Design of 8-to-1 Multiplexer/ Demultiplexer**
- 5. Design of 4 bit binary to gray converter**
- 6. Design of comparator**
- 7. Design of Full adder using 3 modeling styles**
- 8. Design of flip flops: SR, D, JK, T**
- 9. Design of 4-bit binary, BCD counters (synchronous/ asynchronous reset) or any sequence counter**
- 10. Finite State Machine Design**

VLSI programs:

- 1. Introduction to layout design rules**
- 2. Layout, physical verification, placement & route for complex design, static timing analysis, IR drop analysis and crosstalk analysis of the following:**

Basic logic gates

- a. CMOS inverter**
- b. CMOS NOR/ NAND gates**
- c. CMOS XOR and MUX gates**
- d. CMOS 1-bit full adder**
- e. Static / Dynamic logic circuit (register cell)**
- f. Latch**
- g. Pass transistor**
- h. Layout of any combinational circuit (complex CMOS logic gate)- Learning about data paths**
- i. Introduction to SPICE simulation and coding of NMOS/CMOS circuit**
- j. SPICE simulation of basic analog circuits: Inverter / Differential amplifier**
- k. Analog Circuit simulation (AC analysis) – CS & CD amplifier**
- l. System level design using PLL**

1. Simulation using all the modeling styles and Synthesis of all the logic gates using Verilog HDL

Objective: Implement and verify the functionality of basic gates using Xilinx ISE

Scope:

- a) To realize the logical operation of the basic gates using Verilog and implement the same on Xilinx FPGA

Electronic Design Automation Tools and Apparatus required:

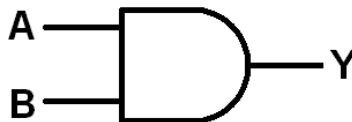
- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

Theory

Logic gates:

A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions low (0) or high (1), represented by different voltage levels.

- a) **AND Gate:** The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB



Boolean expression of AND gate is given as $Y = (A \cdot B)$

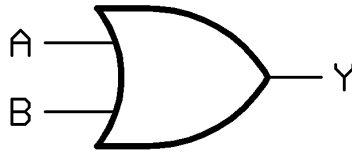
AND gate - Truth Table

A	B	Y

Verilog program for AND gate:

Simulation Results:

b) **OR Gate:** The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.



Boolean expression of OR gate is given as $Y = (A+B)$

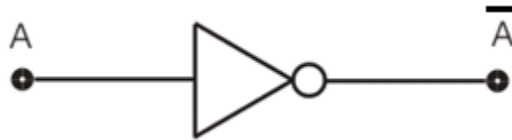
OR gate - Truth Table

A	B	Y

Verilog program for OR gate:

Simulation Results:

c) **NOT Gate:** The NOT gate produces an inverted version of the input at its output. It is also known as an *inverter*. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs.



Boolean expression of NOT gate is given as $A = A'$

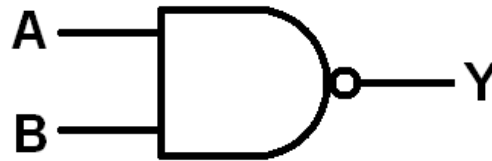
NOT gate - Truth Table

A	A''

Verilog program for NOT gate:

Simulation Results:

- d) **NAND Gate:** NAND gate is NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.



Boolean expression of NAND gate is given as $Y = (A \cdot B)'$

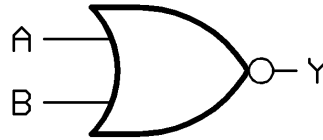
OR gate - Truth Table

A	B	Y

Verilog program for NAND gate:

Simulation Results:

e) **NOR Gate:** NOR gate is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.



Boolean expression of NOR gate is given as $Y = (A+B)'$

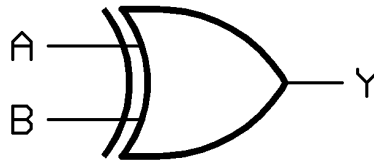
NOR gate - Truth Table

A	B	Y

Verilog program for NOR gate:

Simulation Results:

f) **XOR Gate :** The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high. An encircled plus sign (\oplus) is used to show the EOR operation.



Boolean expression of XOR gate is given as $Y = A.B' + A'.B$

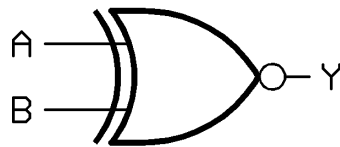
NOR gate - Truth Table

A	B	Y

Verilog program for XOR gate:

Simulation Results:

g) **XNOR Gate:** The 'Exclusive-NOR' gate circuit does the opposite to the EXOR gate. It will give a low output if either, but not both, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.



Note:

The NAND and NOR gates are called *universal functions* since with either one the AND and OR functions and NOT can be generated.

A function in *sum of products* form can be implemented using NAND gates by replacing all AND and OR gates by NAND gates.

A function in *product of sums* form can be implemented using NOR gates by replacing all AND and OR gates by NOR gates.

XNOR gate - Truth Table

A	B	Y

Verilog program for XNOR gate:

Simulation Results:

Result:

Viva Questions:

1. An AND gate and OR gate are given inputs X & 1 , what is expected output?
2. An XOR gate and XNOR gate are given inputs X & 1 , what is expected output?
3. How do you convert a XOR gate into NOT gate?
4. How do you convert a XOR gate into buffer?
5. How do you make a NAND gate out of an AND gate using inverters (NOT gates)?
6. How do you make a NOR gate out of an NAND gate using inverters (NOT gates)?
7. How do you make an AND gate out of an OR gate using inverters (NOT gates)?
8. What does connecting together the inputs of NAND and NOR gates do?
9. What is the difference between wire and reg?
10. What are the uses of user defined primitives (UDP) in Verilog?
11. What are applications of XOR and XNOR gates?
12. What is VLSI Design?
13. What do you mean by HDLs? Give examples.
14. What do you mean by universal gate?

2. Design of 2-to-4 decoder using Verilog HDL

Aim: To design the 2x4 decoder using Verilog and simulate the design

Objective: To develop the source code 2-to-4 decoder using Verilog HDL in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- To understand the design using procedural level coding and structural level coding of decoder using Verilog.
- To verify wave form viewer against the truth table of decoder
- To implement the decoder on FPGA development board

Electronic Design Automation Tools and Apparatus required:

- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

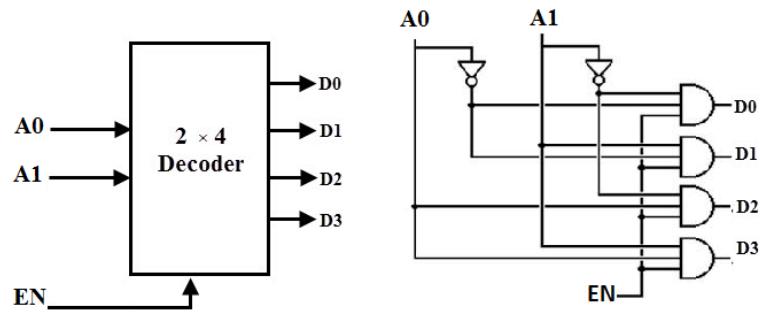
THEORY:

As a decoder is a combinational circuit takes an n-bit binary number and produces an output on one of 2^n output lines.

In a 2-to-4 binary decoder, two inputs are decoded into four outputs hence it consists of two input lines and 4 output lines. Only one output is active at any time while the other outputs are maintained at logic 0 and the output which is held active or high is determined the two binary inputs A1 and A0. The figure below shows the truth table for a 2-to-4 decoder. For a given input, the outputs D0 through D3 are active high if enable input EN is active high ($EN = 1$). When both inputs A1 and A0 are low (or $A1 = A0 = 0$), the output D0 will be active or High and all other outputs will be low.

When $A1 = 0$ and $A0 = 1$, the output D1 will be active and when $A1 = 1$ and $A0 = 0$, then the output D2 will be active. When both the inputs are high, then the output D3 will be high. If the enable bit is zero then all the outputs will be set to zero. This relationship between the inputs and outputs are illustrated in below truth table clearly.

Block diagram:



Inputs			Outputs			
EN	A1	A0	D3	D2	D1	D0
0	×	×	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

a) **Verilog coding for 2 to 4 decoder using dataflow model**

b) Verilog code 2 to 4 decoder using case statement (behavior level-procedural)

Simulation Results:

c) Verilog code for 2 to 4 decoder using structural level

Simulation Results:

Result:

Viva Questions:

1. What is the difference between Verilog and VHDL?
2. Differentiate between Simulation and Synthesis?
3. What are the differences between continuous and procedural assignments?
4. Why RTL synthesis is important?
5. What is the difference between \$display and \$monitor?
6. List out the applications of decoder?
7. Convert 2 to decoder into Half adder?
8. What is a "parallel" case statement?
9. Write the Verilog code for 3 to 8 decoder?
10. What is the function of a decoder's enable input?
11. Explain working of a Decoder
12. List all the system tasks and their purpose?
 - \$display
 - \$strobe
 - \$monitor
 - \$reset
 - \$stop
 - \$finish
 - \$time, \$stime, \$realtime
 - \$scope
 - \$showscopes
 - \$fopen
 - fclose
 - \$fdisply,
 - \$fstrobe

3. Design of 8-to-3 encoder (without and with priority) using Verilog HDL

Objective: To realize 8x3 encoder (without and with priority) using Verilog HDL in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- To understand the design using procedural level coding (using case statement) and structural level coding of encoder and priority encoder using Verilog.
- To verify wave form viewer against the truth table of encoder
- To implement the encoder on FPGA development board and verify the same.

Electronic Design Automation Tools used

- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

THEORY:

An encoder is a combinational logic circuit that essentially performs a “reverse” of decoder functions. An encoder has 2^N input lines and N output lines. In encoder the output lines generate the binary code corresponding to input value. An encoder accepts an active level on one of its inputs, representing digit, such as a decimal or octal digits, and converts it to a coded output such as BCD or binary. Encoders can also be devised to encode various symbols and alphabetic characters. The process of converting from familiar symbols or numbers to a coded format is called **encoding**. An encoder has a number of input lines, only one of which input is activated at a given time and produces an N-bit output code, depending on which input is activated.

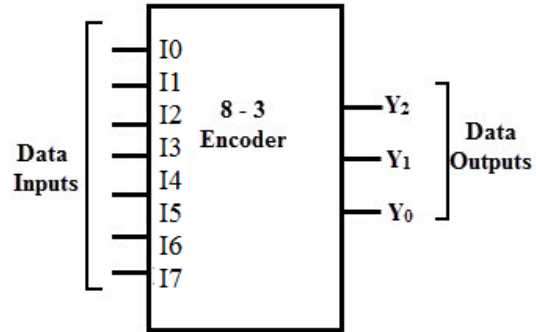
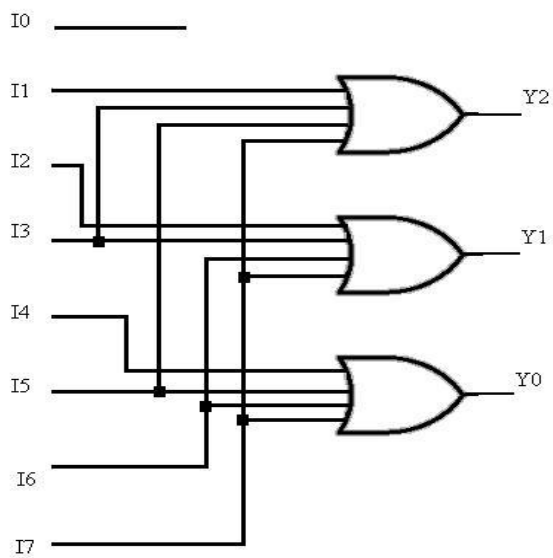
For an 8-to-3 binary encoder with inputs I0-I7 the logic expressions of the outputs Y0-Y2 are:

$$Y0 = I1 + I3 + I5 + I7$$

$$Y1 = I2 + I3 + I6 + I7$$

$$Y2 = I4 + I5 + I6 + I7$$

Circuit Diagram, Block diagram and Truth Table:



I0	I1	I2	I3	I4	I5	I6	I7	Y2	Y1	Y0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Verilog program for 8x3 encoder structural:

Simulation Results:

Verilog program for 8x3 encoder behavioral (using case statement)

Simulation Results:

Result:

Viva Questions:

1. How do I choose between a case statement and a multi-way if-else statement?
2. How do I avoid a priority encoder in an if-else tree?
3. What are the differences between if-else and the ("?:") conditional operator?
4. What is the importance of a default clause in a case construct?
5. What is the difference between full_case and parallel_case synthesis directive?
6. What is the difference in implementation with sequential and combinatorial processes, when the final else clause in a multiway if-else construct is missing?
7. What is the difference in using (== or !=) vs. (=== or !==) in decision making of a flow control construct in a synthesizable code?
8. Explain the differences and advantages of casex and casez over the case statement?
9. How does an encoder differ from a decoder?
10. How does a priority encoder differ from an ordinary encoder?
11. What is the function of a decoder's enable input?

4. Design of 8-to-1 multiplexer and 1-to-8 Demultiplexer

Objective: To realize 8-to-1 multiplexer and 1-to-8 Demultiplexer using Verilog HDL in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- To understand various abstract level of coding coding of 8-to-1 multiplexer and 1-to-8 Demultiplexer using Verilog.
- To verify wave form viewer against the truth table of 8-to-1 multiplexer and 1-to-8 Demultiplexer.
- To implement the 8-to-1 multiplexer and 1-to-8 Demultiplexer on FPGA development board and verify the same.

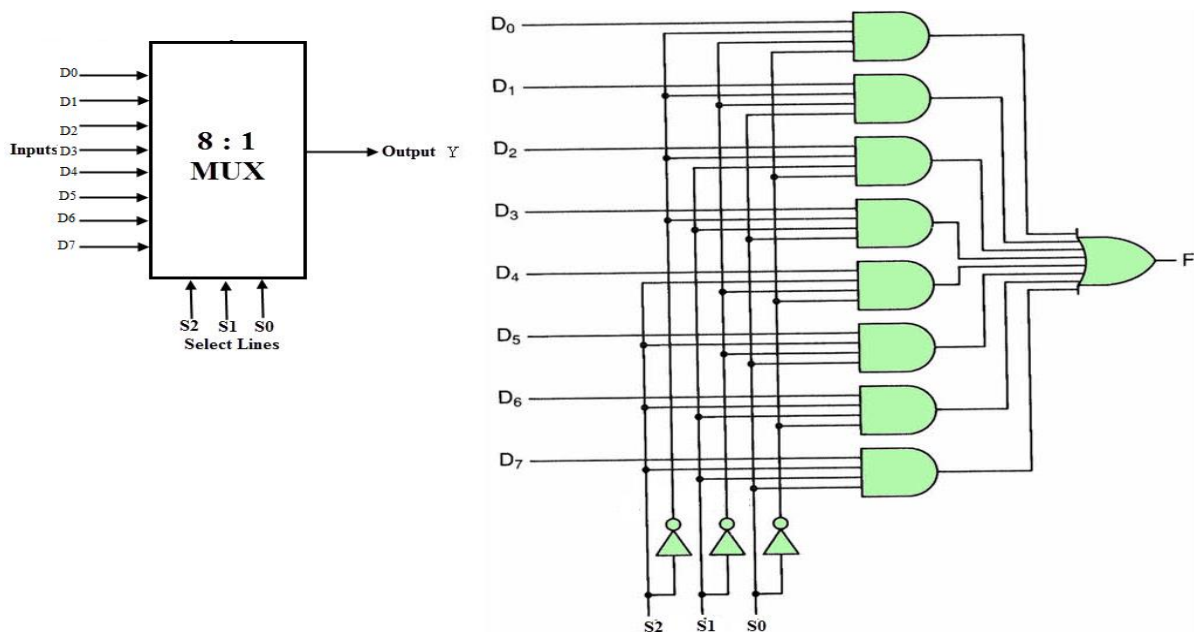
Electronic Design Automation Tools and Apparatus required:

- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

Theory:

An 8-to-1 multiplexer consists of eight data inputs D0 through D7, three input select lines S2 through S0 and a single output line Y. Depending on the select lines combinations, multiplexer decodes the inputs.

The below figure shows the block diagram of an 8-to-1 multiplexer with enable input that enable or disable the multiplexer. Since the number data bits given to the MUX are eight then 3 bits ($2^3=8$) are needed to select one of the eight data bits.



The truth table for an 8-to1 multiplexer is given below with eight combinations of inputs so as to generate each output corresponds to input.

Select Data Inputs			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

For example, if $S_2=0$, $S_1=1$ and $S_0=0$ then the data output Y is equal to D_2 . Similarly the data outputs D_0 to D_7 will be selected through the combinations of S_2 , S_1 and S_0 as shown in below figure.

From the above truth table, the Boolean equation for the output is given as

$$Y = D_0 \overline{S_2} \overline{S_1} \overline{S_0} + D_1 \overline{S_2} \overline{S_1} S_0 + D_2 \overline{S_2} S_1 \overline{S_0} + D_3 \overline{S_2} S_1 S_0 + D_4 S_2 \overline{S_1} \overline{S_0} + D_5 S_2 \overline{S_1} S_0 + D_6 S_2 S_1 \overline{S_0} + D_7 S_2 S_1 S_0$$

From the above Boolean equation, the logic circuit diagram of an 8-to-1 multiplexer can be implemented by using 8 AND gates, 1 OR gate and 7 NOT gates as shown in below figure. In the circuit, when enable pin is set to one, the multiplexer will be disabled and if it is zero then select lines will select the corresponding data input to pass through the output.

Verilog code for 8 to 1 mux using if else statement:

Simulation Results:

Verilog code for 8 to 1 mux using Case statement:

Simulation Results:

Verilog code for 8 to 1 mux – Gate level realization:

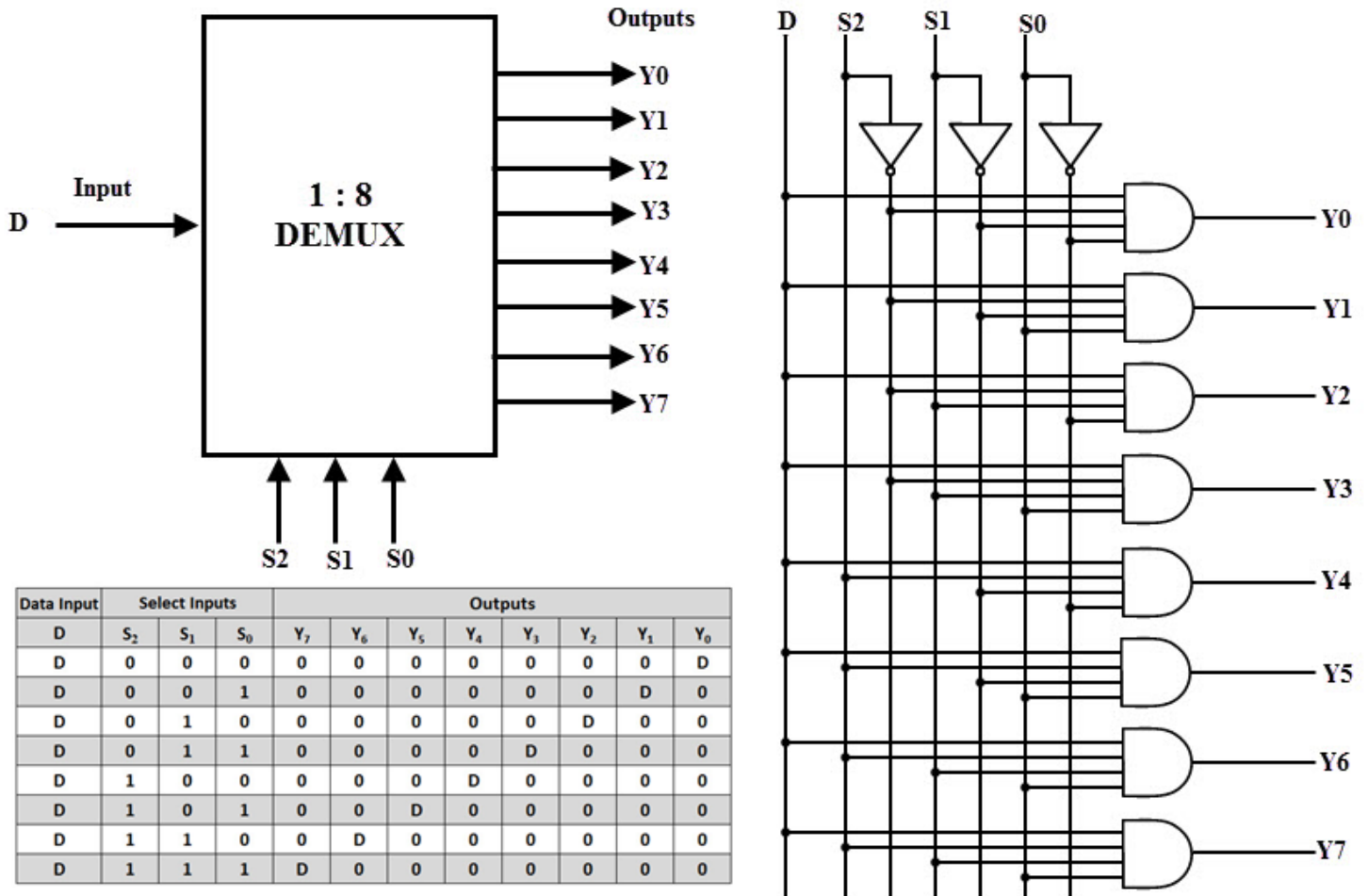
Simulation Results

Demultiplexor

A demultiplexer is a combinational logic circuit that receives the information on a single input and transmits the same information over one of 2^n possible output lines.. The action or operation of a demultiplexer is opposite to that of the multiplexer. As inverse to the MUX , demux is a one-to-many circuit. With the use of a demultiplexer , the binary data can be bypassed to one of its many output data lines. Demultiplexers are mainly used in Boolean function generators and decoder circuits.

The below figure shows the block diagram of a 1-to-8 demultiplexer that consists of single input D, three select inputs S2, S1 and S0 and eight outputs from Y0 to Y7.

It is also called as 3-to-8 demultiplexer due to three select input lines. It distributes one input line to one of 8 output lines depending on the combination of select inputs.



The truth table for this type of demultiplexer is shown below. The input D is connected with one of the eight outputs from Y0 to Y7 based on the select lines S2, S1 and S0.

For example, if $S_2S_1S_0=000$, then the input D is connected to the output Y0 and so on.

From this truth table, the Boolean expressions for all the outputs can be written as follows.

$$Y_0 = D \overline{S_2} \overline{S_1} \overline{S_0}$$

$$Y_1 = D \overline{S_2} \overline{S_1} S_0$$

$$Y_2 = D \overline{S_2} S_1 \overline{S_0}$$

$$Y_3 = D \overline{S_2} S_1 S_0$$

$$Y_4 = D S_2 \overline{S_1} \overline{S_0}$$

$$Y_5 = D S_2 \overline{S_1} S_0$$

$$Y_6 = D S_2 S_1 \overline{S_0}$$

$$Y_7 = D S_2 S_1 S_0$$

From these obtained equations, the logic diagram of this demultiplexer can be implemented by using eight AND gates and three NOT gates as shown in below figure. The different combinations of the select lines , select one AND gate at given time , such that data input will appear at a particular output.

Applications of Demultiplexer

Since the demultiplexers are used to select or enable the one signal out of many, these are extensively used in microprocessor or computer control systems such as

- Selecting different IO devices for data transfer
- Choosing different banks of memory
- Depends on the address, enabling different rows of memory chips

Enabling different functional units.

Other than these, demultiplexers can be found in a wide variety of application such as

Synchronous data transmission systems

- Boolean function implementation
- Data acquisition systems
- Combinational circuit design
- Automatic test equipment systems
- Security monitoring systems (for selecting a particular surveillance camera at a time), etc.

Verilog code for 1 to 8 Demux – Gate level realization:

Simulation Results:

Verilog code for 1 to 8 Demux – using assign statement:

Simulation Results:

Results:

Questions:

1. What logic is inferred when there are multiple assign statements targeting the same wire?
2. What do conditional assignments get inferred into?
3. What is the logic that gets synthesized when conditional operators in a single continuous assignment are nested?
4. What value is inferred when multiple procedural assignments made to the same reg variable in an always block?
5. Why should a non-blocking assignment be used for sequential logic, and what would happen if a blocking assignment were used? Compare it with the same code in a combinatorial block.
6. What are the different approaches of connecting ports in a hierarchical design? What are the pros and cons of each?
7. What is difference between Verilog full case and parallel case?
8. Explain the working of 4:1 MUX with block diagram.
9. State some applications of MUX.
10. How is the 4-to-2 encoder different from a 4-to-1 multiplexer?
11. Explain the difference between a DEMUX and a MUX.
12. True or False: The circuit for a DEMUX is basically the same as for a decoder.
13. Draw the logic diagram of 2-bit comparator using gates.
14. Write Verilog codes for Decoder, Encoder, MUX and DEMUX.
15. Write syntax of module if there are two inputs (A and B) and one output (C).

5. Design of 4-bit binary to gray converter/ Gray to Binary Code Converter

Objective: To realize 4-bit binary to gray converter/ Gray to Binary Code Converter using Verilog HDL in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- To understand various abstract level of coding 4-bit binary to gray converter/ Gray to Binary Code Converter using Verilog.
- To verify wave form viewer against the truth table of 4-bit binary to gray converter/ Gray to Binary Code Converter.
- To implement the 4-bit binary to gray converter/ Gray to Binary Code Converter on FPGA development board and verify the same.

Electronic Design Automation Tools used

- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

Theory: Binary to Gray converter

The logical circuit which converts binary code to equivalent gray code is known as **binary to gray code converter**. The gray code is a non weighted code. The successive gray code differs in one bit position only that means it is a unit distance code. It is also referred as cyclic code. It is not suitable for arithmetic operations. It is the most popular of the unit distance codes. It is also a reflective code. An n-bit Gray code can be obtained by reflecting an n-1 bit code about an axis after 2^{n-1} rows, and putting the MSB of 0 above the axis and the MSB of

Decimal Number	4 bit Binary Number <u>ABCD</u>	4 bit Gray Code <u>G₁G₂G₃G₄</u>
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

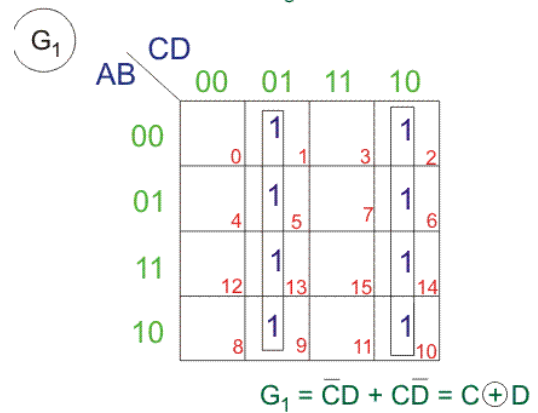
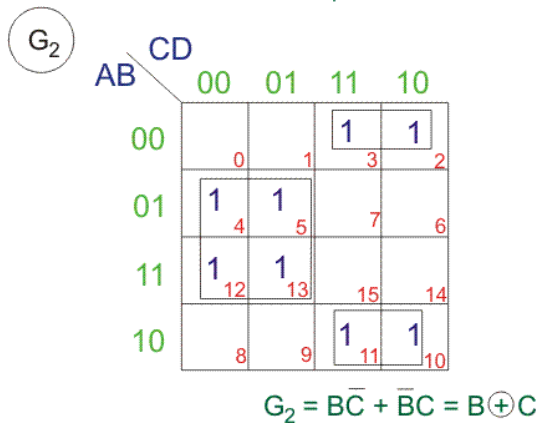
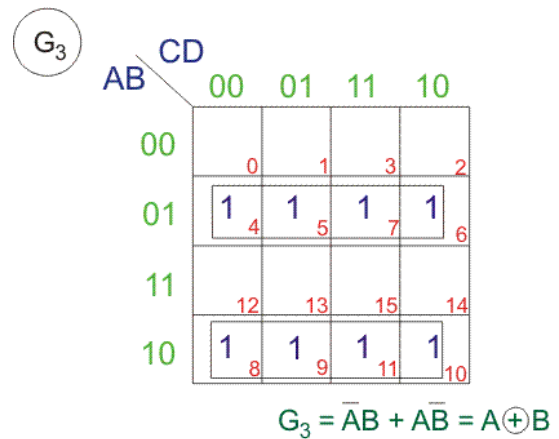
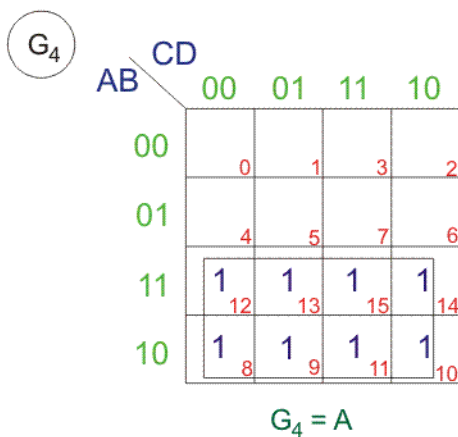
1 below the axis. Reflection of Gray codes is shown below. The 4 bits binary to gray code conversion table is given below,

That means, in 4 bit gray code, (4-1) or 3 bit code is reflected against the axis drawn after $(2^{4-1})^{\text{th}}$ or 8th row. The bits of 4 bit gray code are considered as $G_4G_3G_2G_1$. Now from conversion table, From above SOPs, let us draw K -maps for G_4 , G_3 , G_2 and G_1 .

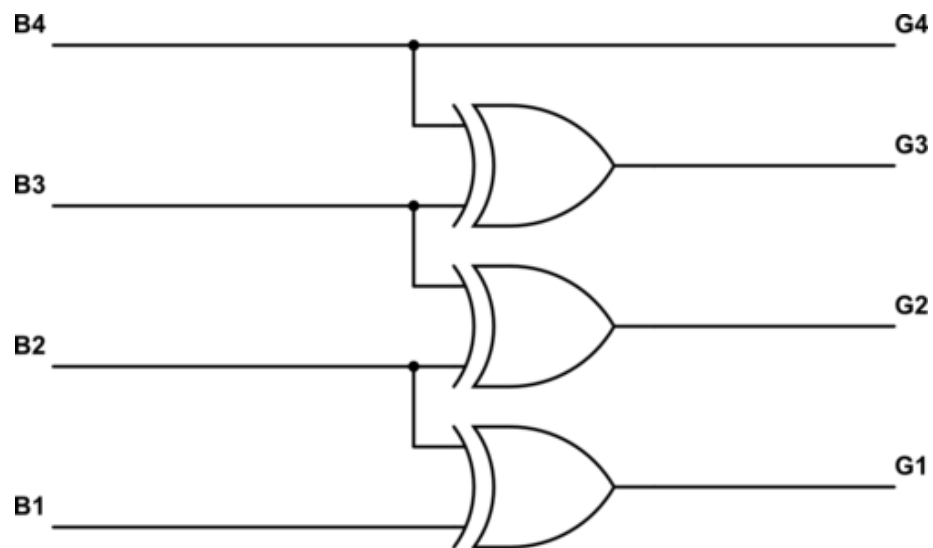
$$G_4 = \sum m(8, 9, 10, 11, 12, 13, 14, 15), \quad G_3 = \sum m(4, 5, 6, 7, 8, 9, 10, 11)$$

$$G_2 = \sum m(2, 3, 4, 5, 10, 11, 12, 13), \quad G_1 = \sum m(1, 2, 5, 6, 9, 10, 13, 14)$$

K-Map



Circuit Diagram : Binary to Gray Converter



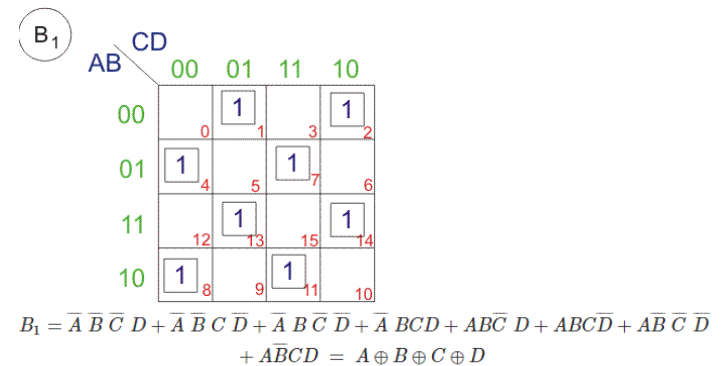
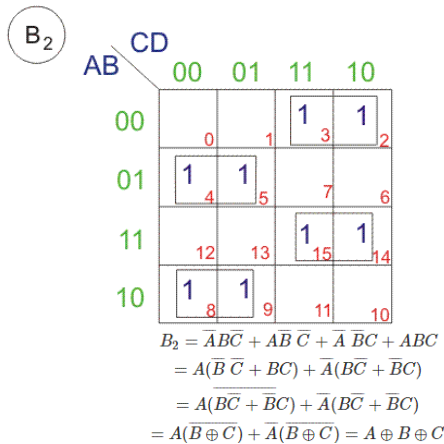
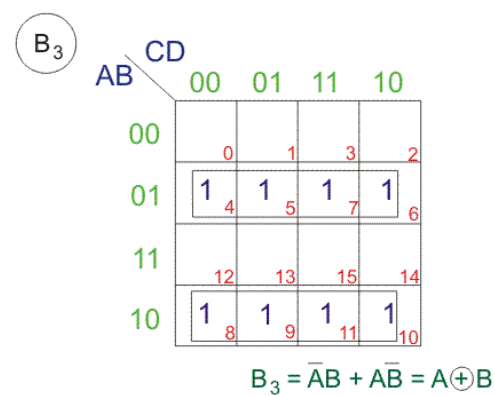
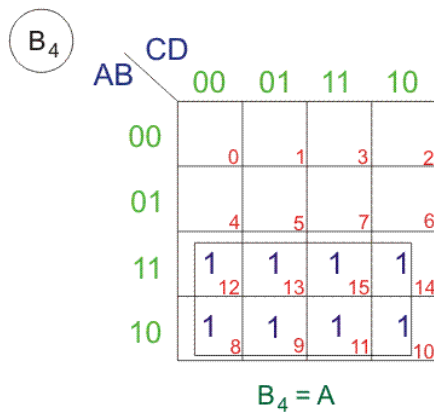
Gray to Binary Code Converter

In **gray to binary code converter**, input is a multiplies gray code and output is its equivalent binary code.

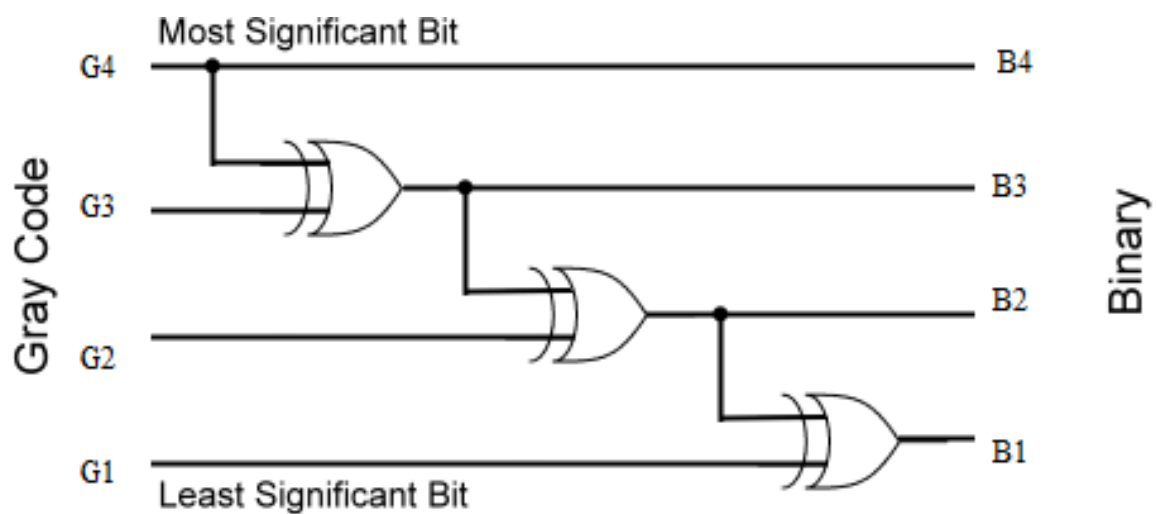
Let us consider a 4 bit gray to binary code converter. To design a 4 bit gray to binary code converter, we first have to draw a conversion table. From above gray code we get,

4 bit Gray Code	4 bit Binary Code
A B C D	B ₄ B ₃ B ₂ B ₁
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 1	0 0 1 0
0 0 1 0	0 0 1 1
0 1 1 0	0 1 0 0
0 1 1 1	0 1 0 1
0 1 0 1	0 1 1 0
0 1 0 0	0 1 1 1
1 1 0 0	1 0 0 0
1 1 0 1	1 0 0 1
1 1 1 1	1 0 1 0
1 1 1 0	1 0 1 1
1 0 1 0	1 1 0 0
1 0 1 1	1 1 0 1
1 0 0 1	1 1 1 0
1 0 0 0	1 1 1 1

K-Map



Circuit diagram Gray to Binary encoder:



Verilog Code for Binary to Gray convertor:

Simulation Results:

Verilog code for Gray to Binary convertor:

Simulation Results:

Result:

Questions:

1. What are the different approaches of connecting ports in a hierarchical design? What are the pros and cons of each?
2. Can there be full or partial no-connects to a multi-bit port of a module during its instantiation?
3. What happens to the logic after synthesis, that is driving an unconnected output port that is left open (that is, noconnect) during its module instantiation?
4. What value is sampled by the logic from an input port that is left open (that is, no-connect) during its module instantiation?
5. How is the connectivity established in Verilog when connecting wires of different widths?
6. Can I use a Verilog function to define the width of a multi-bit port, wire, or reg type?
7. Explain the difference between Binary and Gray code.
8. List out the advantages and disadvantages of Binary and Gray code.
9. How to convert Binary to Gray code and vice-versa?
10. Explain the Test Bench Waveform of 4 bit binary to Gray converter

6. Design of 4-bit Magnitude Comparator

Design of Comparator using Verilog and simulate the design

Objective: To realize 4-bit Magnitude Comparator using Verilog HDL in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- To understand the realization of 4-bit Magnitude Comparator using Verilog.
- To verify wave form viewer result against the truth table of 4-bit Magnitude Comparator
- To 4-bit Magnitude Comparator on FPGA development board and verify the same.

Theory

A magnitude comparator is a combinational circuit that compares two numbers A & B to

determine whether:

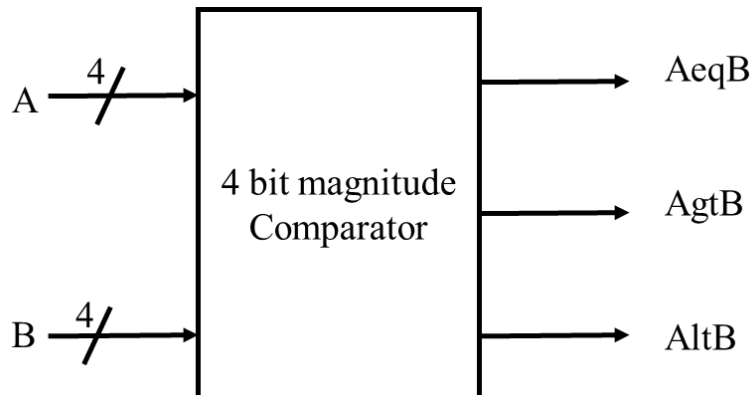
$A > B$, or $A = B$, or $A < B$

Inputs: First 4-bit number A

Second 4-bit number B

Outputs: 3 output signals (GT, EQ, LT), where:

1. $AgtB = 1$ IFF $A > B$
2. $AeqB = 1$ IFF $A = B$
3. $AltB = 1$ IFF $A < B$



Verilog code for 4 bit magnitude comparator

Simulation Results:

Result:

Questions:

1. Draw the logic diagram of 2-bit comparator using gates.
2. Which basic logic gate is suitable to design comparator? Why?
3. What are the guidelines for coding priority encoders?
4. What is the difference between “==” and “===”?
5. What are the differences between assignment in initial and always blocks?
6. What are the differences between blocking and non blocking statements?
7. What are the differences between Task and Functions?
8. What is difference between freeze deposit and force?
9. What are Different types of Verilog Simulators?
10. What are the logical operators in Verilog?

7. Design of Full adder using 3 modeling styles

Objective: To develop the source code for full adder by using three different modelling styles of Verilog in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- To understand the level of abstraction of Verilog HDL coding such as behavior level, gate level/structural level and data flow modelling.
- To verify the design using wave form viewer against the truth table
- To implement the full adder on FPGA development board

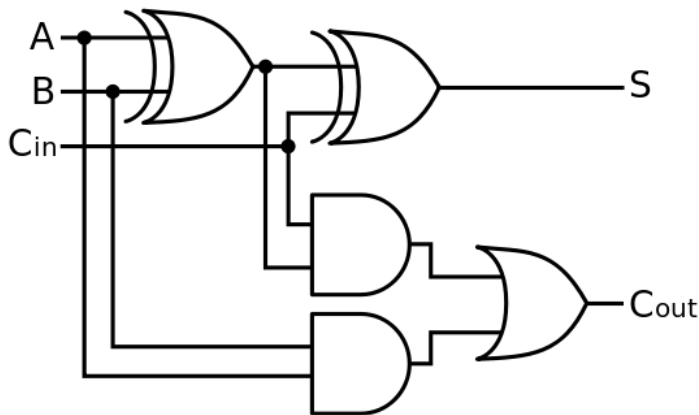
Electronic Design Automation Tools and Apparatus required:

- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

Theory:

A **full adder** adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as A , B , and C_{in} ; A and B are the operands, and C_{in} is a bit carried in from the previous less-significant stage. The full adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. bit binary numbers. The circuit produces a two-bit output. Output carry and sum typically represented by the signals C_{out} and S

Circuit diagram:



The truth table for the full adder

Inputs			Outputs	
A	B	C _{in}	C _{out}	S

a. Gate level modeling

The module is implemented in terms of logic gates and interconnections between these gates. Designer should know the gate-level diagram of the design.

```
wire Z, Z1, OUT, OUT1, OUT2, IN1, IN2;  
  
and a1(OUT1, IN1, IN2);  
nand na1(OUT2, IN1, IN2);  
xor x1(OUT, OUT1, OUT2);  
not (Z, OUT);  
buf final (Z1, Z);
```

- Essentially describes the topology of a circuit
- All instances are executed concurrently just as in hardware
- Instance name is not necessary
- The first terminal in the list of terminals is an output and the other terminals are inputs
- Not the most interesting modeling technique for our class

b. Data Flow Modeling

Module is designed by specifying the data flow, where the designer is aware of how data flows between hardware registers and how the data is processed in the design

- The continuous assignment is one of the main constructs used in dataflow modeling
 - `assign out = i1 & i2;`
 - `assign addr[15:0] = addr1[15:0] ^ addr2[15:0];`
 - `assign {c_out, sum[3:0]}=a[3:0]+b[3:0]+c_in;`
- A continuous assignment is always active and the assignment expression is evaluated as soon as one of the right-hand-side variables change

- Assign statements describe hardware that operates concurrently – ordering does not matter
- Left-hand side must be a scalar or vector net. Right-hand side operands can be wires, (registers, integers, and real)

c. Behavioral level Modeling:

This is the highest level of abstraction. A module can be implemented in terms of the design algorithm. The designer no need to have any knowledge of hardware implementation.

Verilog program for full adder – Gate level modeling

Simulation result:

Verilog program for full adder – dataflow modeling

Simulation result:

Verilog program for full adder behavior level modeling

Simulation Results:

Result:

Questions

1. How do you test your design using HDL?
2. What is the difference between wire and reg?
3. Can I model combinational logic using always statements? How?
4. Why can I not instantiate a module inside an 'if' statement (or an always block, for that matter)?
5. How do you implement full adder using two half adders using Verilog HDL code? What is the level of abstraction if you implement full adder using two half adders?
6. What are the applications of adders?
7. What happens to the logic after synthesis, that is driving an unconnected output port that is left open during its module instantiation?
8. Which assignment statement will be used in Behavioral modeling?
9. Draw logic diagram of Half adder and write its logical expressions.
10. Draw logic diagram of Full adder and write its logical expressions.
11. Explain dataflow type of modelling of Verilog.
12. Explain behavioral type of modelling of Verilog.

8. Design of Latches and flip flops: D-latch SR, D,JK, T

Aim: Design of Latches and flip flops (D-latch SR, D,JK, T) using Verilog and simulates the design

Objective: To develop the source code for Latches and flip flops (D-latch SR, D,JK, T) using Verilog in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- To understand the Verilog HDL coding Latches and flip flops (D-latch SR, D,JK, T)
- To verify the design using wave form viewer against the truth table
- To implement the Latches and flip flops (D-latch SR, D,JK, T) on FPGA development board

Electronic Design Automation Tools and Apparatus required:

- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

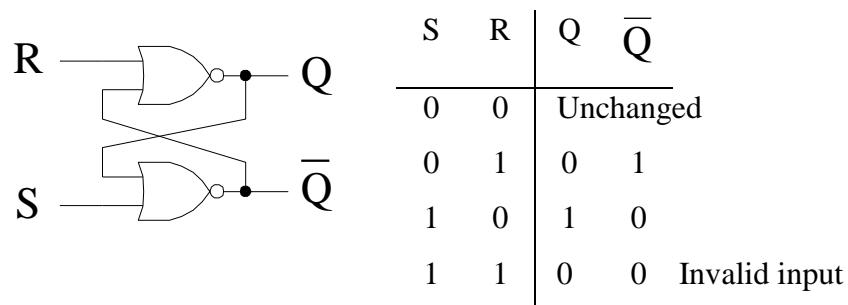
Theory:

LATCH AND FLIP-FLOP

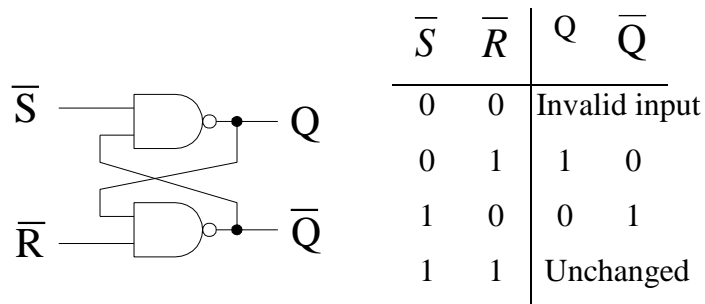
Latch and flip-flop are memory devices and implemented using bistable circuit - its output will remain in either 0 or 1 state. The output state of a latch is controlled by its excitation input signals. A flip-flop (FF) is predominately controlled by a clock and its output state is determined by its excitation input signals. Note that if the clock is replaced by a gated control signal, the flip-flop becomes a gated latch.

a. RS (reset-set) latch circuit

When S (set) is set to 1, the output Q will be set to 1. Likewise, when R (reset) is set to 1, the output Q will be set to 0. It is invalid to set both S and R to 1.



NOR gate implementation

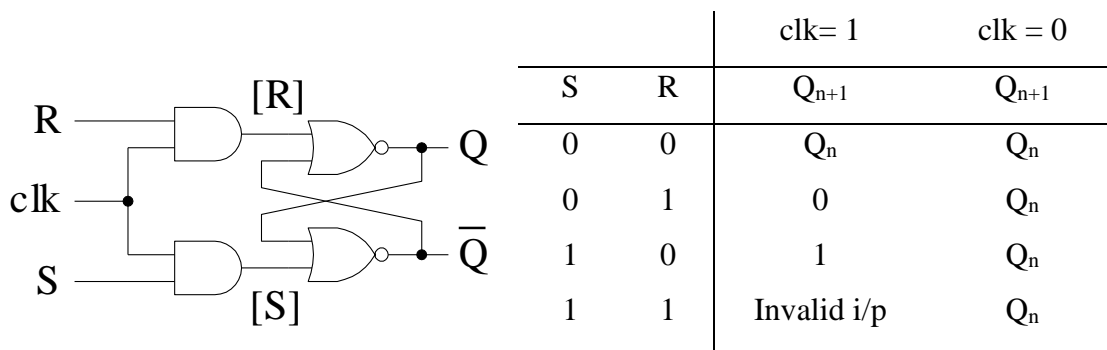


NAND gate implementation

Note that the input is **active high** for NOR gate implementation, whereas the input is **active low** for NAND gate implementation.

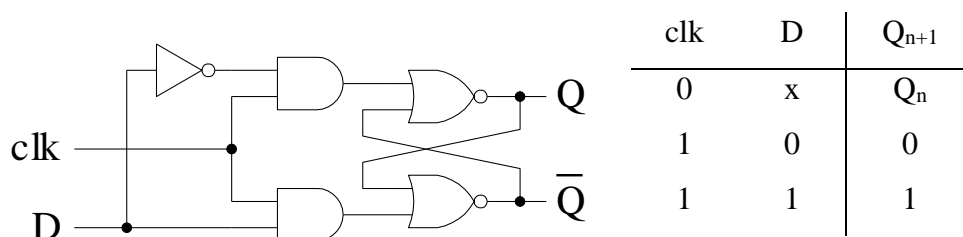
b. Clocked RS FF

The major problem of RS latch is its susceptibility to voltage noise which could change the output states of the FF. With the clocked RS FF, the problem is remedied. With the clock held low, [S] & [R] held low, the output remains unchanged. With the clock held high, the output follows R & S. Thus the output will be latched its states when the clock goes low.



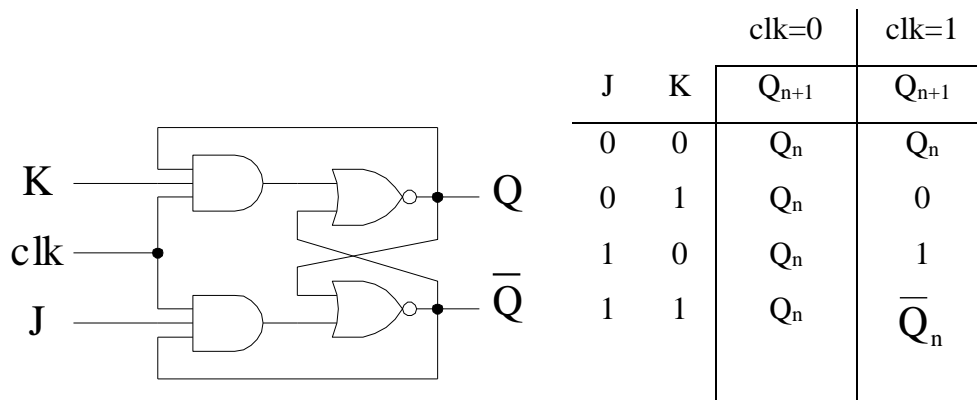
c. D-type FF

The D-type FF remedies the indeterminate state problem that exists when both inputs to a clocked RS FF are high. The schematic is identical to a RS FF except that an inverter is used to produce a pair of complementary input.



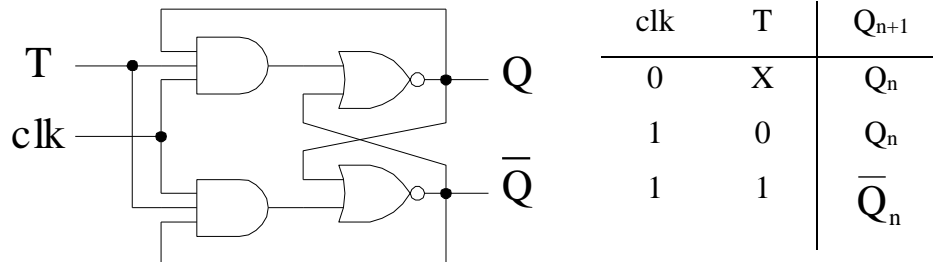
d. JK FF

The JK FF is a refinement of the RS FF in that the undetermined state of the RS type is defined in the JK type. Inputs J and K behave like inputs S and R to set and reset (clear) the FF, respectively. The input marked J is for *set* and the input marked K is *reset*.

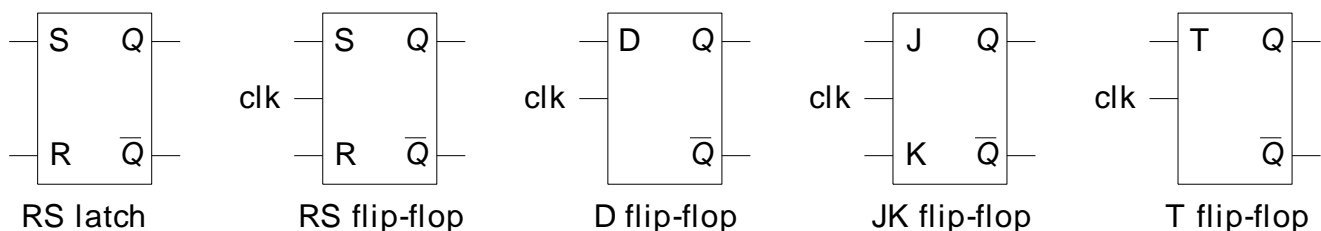


e. T-type FF

The toggle (T) FF has a clock input which causes the output state changed for each clock pulse if T is in its active state. It is useful in counter design.



Logic symbols of various latch and level-triggered flip-flops

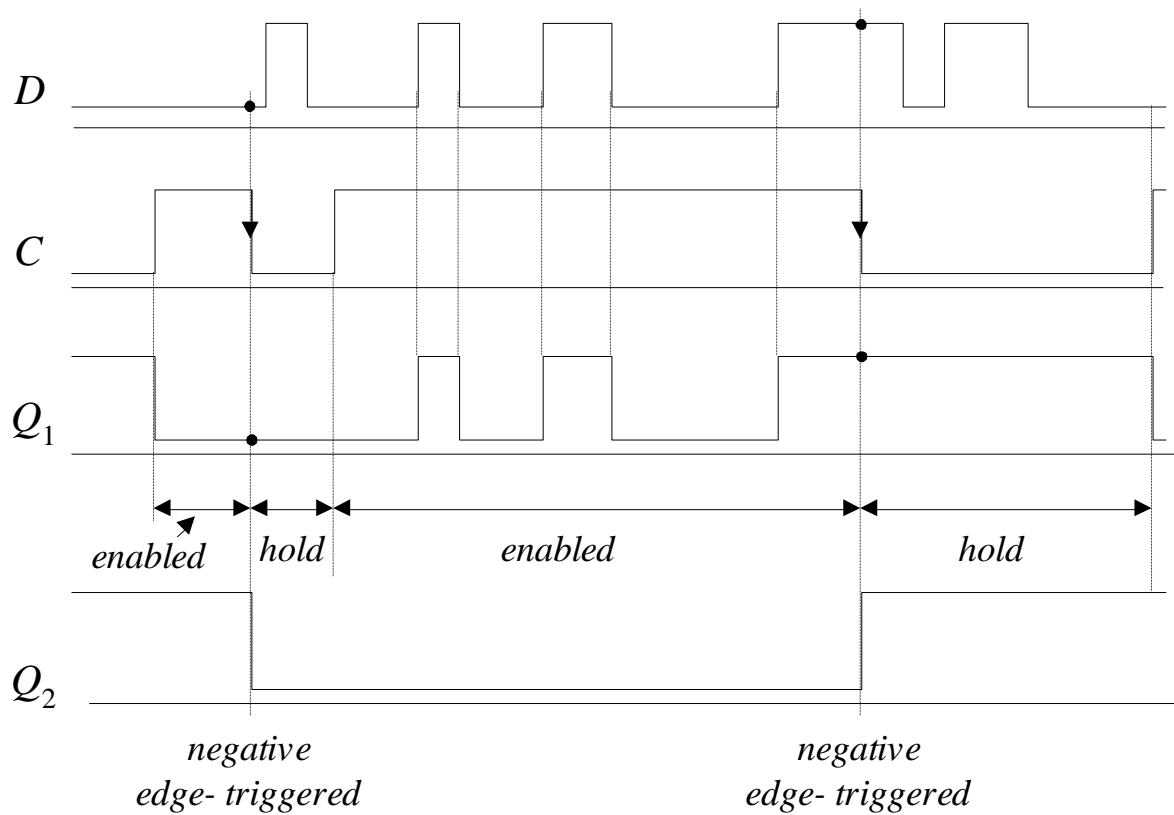
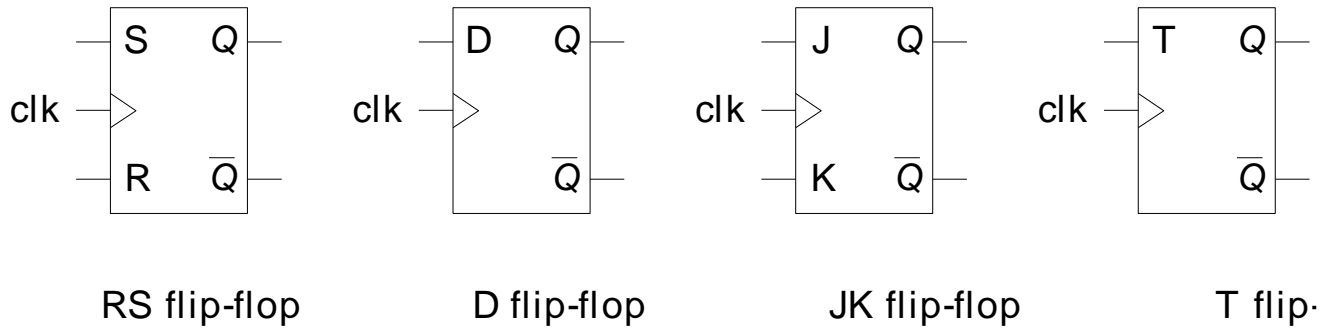


f. Edge-triggered FF

Clocked FF is a **level-triggered** device, its output responses to the input during the clock active period and this is referred to as the "0" and "1" catching problem. For sequential synchronous circuit, the data transfer

is required to be synchronized with the clock signal. Additional circuit is included in the FF to ensure that it will only response to the input at the transition edge of the clock pulse. These type of devices are called *edge-triggered* FFs.

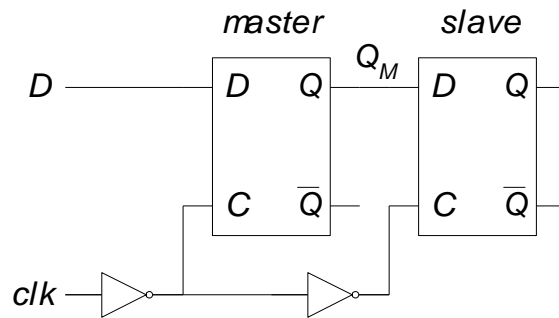
Logic symbols of various edge-triggered flip-flops



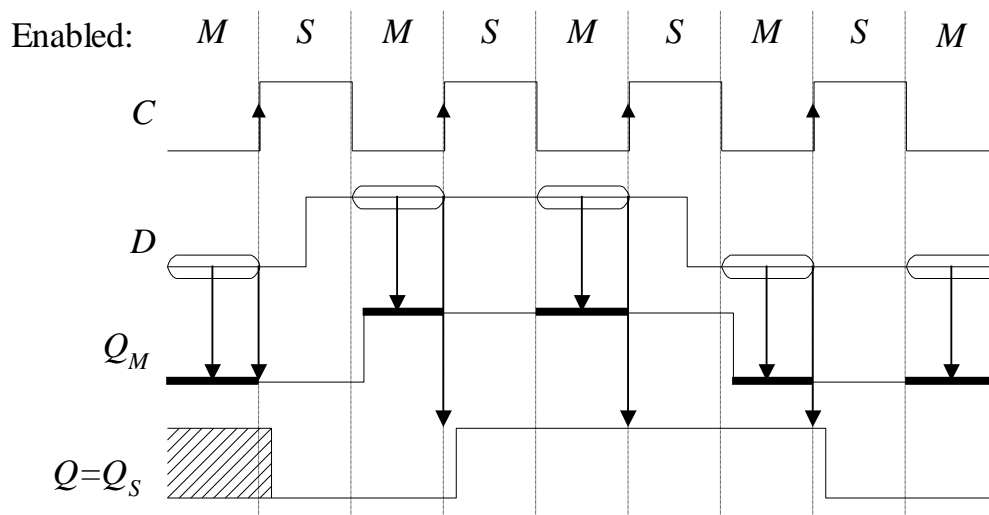
Timing diagram for a gated D latch and a negative edge-triggered D FF

g. Master-slave FF

A master-slave type FF consists of two FFs, a master stage and a slave stage. The output states responding to the inputs are transmitted to slave output through the master stage in different time slots of the clock pulse. Hence the output will not be affected by the undesirable changes at the input after the output of the master FF has been latched to the slave FF.



Master-slave D flip-flop



Timing diagram

h. FF timing parameters

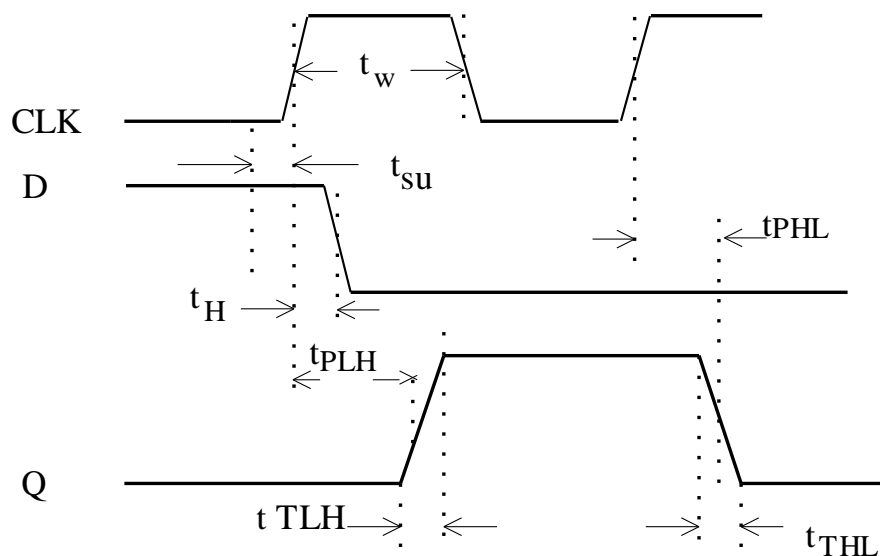
Propagation delay: propagation delay for a FF is the amount of time it takes for the output of the FF to change its state from a clock trigger or asynchronous set or reset. It is defined from the 50% point of the input pulse to the 50% point of the output pulse. Propagation delay is specified as t_{PHL} - the propagation time from a HIGH to a LOW, and as t_{PLH} - the propagation time from a LOW to a HIGH.

Output transition time: the output transition time is defined as the rise time or fall time of the output. The t_{TLH} is the 10% to 90% time, or LOW to HIGH transition time. The t_{THL} is the 90% to 10% time, or the HIGH to LOW transition time.

Setup time: the setup time is defined as the interval immediately preceding the active transition of the clock pulse during which the control or data inputs must be stable (at a valid logic level). Its parameter symbol is t_{su} .

Hold time: the hold time is the amount of time that the control or data inputs must be stable after the clock trigger occurs. The t_H is the parameter symbol for hold time.

Minimum pulse width: the minimum pulse width is required to guarantee a correct state change for the flip-flop. It is usually denoted by t_w .



Verilog program for D-latch

Simulation Results:

Verilog program for D-flip flop with sync reset

Simulation Results:

Verilog program for D-flip flop with async reset

Simulation results:

Verilog program for T-flip flop with sync reset

Simulation results:

Verilog program for SR-flip flop

Simulation Results

Verilog program for JK-flip flop

Simulation Results

Questions:

1. What are the considerations to be taken choosing between flop-flops vs. latches in a design?
2. Sequential design must meet the set up time and hold time requirement. Why?
3. Which one is better, asynchronous or synchronous reset for the storage elements?
4. Why should a non-blocking assignment be used for sequential logic, and what would happen if a blocking assignment were used? Compare it with the same code in a combinatorial block
5. Explain the Verilog statement `always@(posedge clk)` and `always@(negedge clk)`.
6. What is triggering in sequential circuit? How do you represent the various triggering mechanisms using Verilog?
7. Write a Verilog code for D flip flop with gated clock?
8. What is the race condition in Verilog?
9. What is the difference between swapping the contents of two registers with temp register and without temp register?
10. What exactly expression `reg[8*13:1] string_val;` signifies?
11. What is the difference between Concurrent & Sequential statements?
12. Give at least four differences between Latch and Flip-flop.
13. What is disadvantage of SR flip-flop?
14. What is disadvantage of JK flip-flop?
15. What is race around condition? What is done to remove race around condition

9. Design of 4-bit binary, BCD counters (Synchronous/Asynchronous reset) or any sequence Counter

Objective: To develop the source code for BCD counters (Synchronous/Asynchronous reset) using Verilog in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- a. To understand the Verilog HDL coding of 4-bit binary, BCD counters.
- b. To verify the design using wave form viewer against the truth table
- c. To implement the of 4-bit binary, BCD counters on FPGA development board

Electronic Design Automation Tools and Apparatus required:

- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

Theory

The counters which use clock signal to change their transition are called “Synchronous counters”. This means the synchronous counters depends on their clock input to change state values. In synchronous counters, all flip flops are connected to the same clock signal and all flip flops will trigger at the same time.

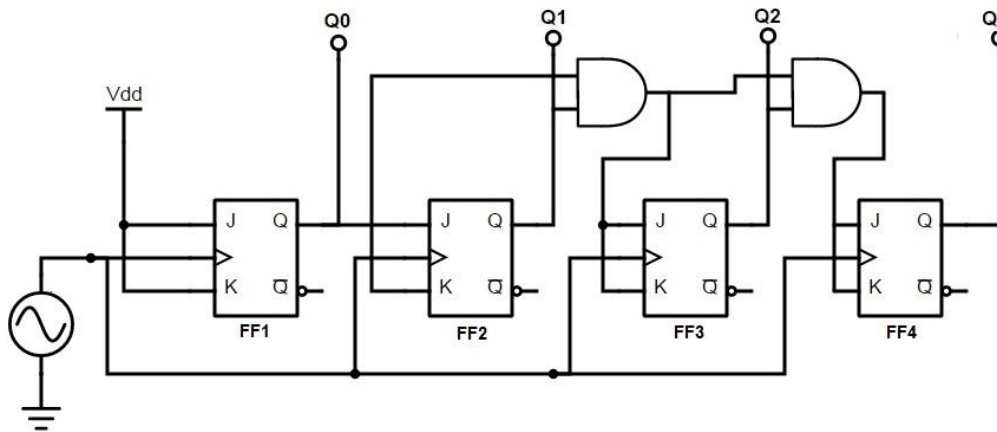
Synchronous counters are also known as ‘Simultaneous counters’. There is no propagation delay and no ripple effect in synchronous counters.

4 bit Synchronous UP Counter

The 4 bit up counter shown in below diagram is designed by using JK flip flop. External clock pulse is connected to all the flip flops in parallel.

For designing the counters JK flip flop is preferred. The significance of using JK flip flop is that it can toggle its state if both the inputs are high, depending on the clock pulse. The inputs of first flip flop are connected to HIGH (logic 1), which makes the flip flop to toggle, for every clock pulse entered into it. So the synchronous counter will work with single clock signal and changes its state with each pulse. The output of first JK flip flop (Q) is connected to the input of second flip flop. The AND gates (which are connected externally) drives the inputs of other two flip flops. The inputs of these AND gates , are supplied from previous stage flip flop outputs.

If inputs of FF2 are connected directly to the Q1 output of FF1, the counter would not function properly. This is because, the Q1 value is high at count of 2_{10} , this means that the FF2 flip flop will toggle for the 3rd clock pulse. This results in wrong counting operation, gives the count as 7_{10} instead of 4_{10} .



To prevent this problem AND gates are used at the input side of FF2 and FF3. The output of the AND gate will be high only when the Q0, Q1 outputs are high. So for the next clock pulse, the count will be 00012. Similarly, the flip flop FF3 will toggle for the fourth clock pulse when Q0, Q1 and Q2 are high. The Q3 output will not toggle till the 8th clock pulse and will again remain high until 16th clock pulse. After the 16th clock pulse, the q outputs of all flip flops will return to 0.

Operation

In the up counter the 4 bit binary sequence starts from 0000 and increments up to 1111. Before understanding the working of the above up counter circuit know about JK Flip flop.

In the above circuit as the two inputs of the flip flop are wired together. So, there are only two possible conditions that can occur, that is, either the two inputs are high or low.

If the two inputs are high then JK flip-flop toggles and if both are low JK flip flop remembers i.e. it stays in the previous state.

Let us see the operation. Here clock pulse indicates edge triggered clock pulse.

1. In the first clock pulse, the outputs of all the flip flops will be at 0000.
2. In the second clock pulse, as inputs of J and k are connected to the logic high, output of JK flip flop (FF0) change its state. Thus the output of the first flip-flop (FF0) changes its state for every clock pulse. This can be observed in the above shown sequence. The LSB changes its state alternatively. Thus producing -0001
3. In the third clock pulse next flip flop (FF1) will receive its J K inputs i.e (logic high) and it changes its state. At this state FF0 will change its state to 0. And thus input on the FF1 is 0. Hence output is -0010

4. Similarly, in the fourth clock pulse FF1 will not change its state as its inputs are in low state, it remains in its previous state. Though it produces the output to FF2, it will not change its state due to the presence of AND gate. FF0 will again toggle its output to logic high state. Thus Output is 0011.
5. In the fifth clock pulse, FF2 receives the inputs and changes its state. While, FF0 will have low logic on its output and FF1 will also be low state producing 0100.

This process continuous up to 1111.

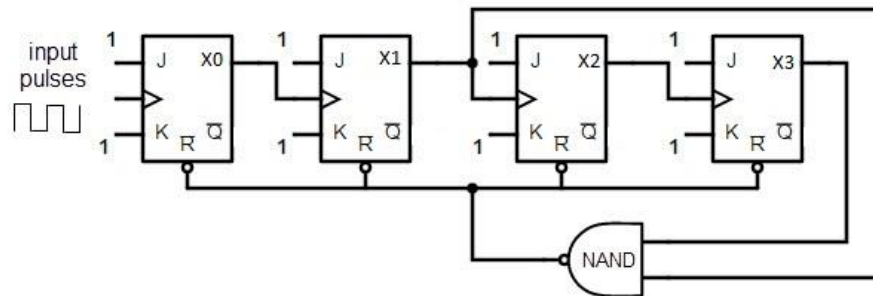
After reaching zero again the three flip flops toggles to logic low i.e 0000 and again count starts.

Timing diagram for up counter is shown below.

BCD or Decade Counter Circuit

A binary coded decimal (BCD) is a serial digital counter that counts ten digits .And it resets for every new clock input. As it can go through 10 unique combinations of output, it is also called as “Decade counter”. A BCD counter can count 0000, 0001, 0010, 1000, 1001, 1010, 1011, 1110, 1111, 0000, and 0001 and so on.

A 4 bit binary counter will act as decade counter by skipping any six outputs out of the 16 (24) outputs. There are some available ICs for decade counters which we can readily use in our circuit, like 74LS90. It is an asynchronous decade counter.



The above figure shows a decade counter constructed with JK flip flop. The J output and K outputs are connected to logic 1. The clock input of every flip flop is connected to the output of next flip flop, except the last one.

The output of the NAND gate is connected in parallel to the clear input ‘CLR’ to all the flip flops. This ripple counter can count up to 16 i.e. 24.

Decade Counter Operation

When the Decade counter is at REST, the count is equal to 0000. This is first stage of the counter cycle. When we connect a clock signal input to the counter circuit, then the circuit will count the binary sequence. The first clock pulse can make the circuit to count up to 9 (1001). The next clock pulse advances to count 10 (1010).

Then the ports X1 and X3 will be high. As we know that for high inputs, the NAND gate output will be low. The NAND gate output is connected to clear input, so it resets all the flip flop stages in decade counter. This means the pulse after count 9 will again start the count from count 0.

Truth Table of Decade Counter

Input Pulses	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
0	0	0	0	0 (resets)

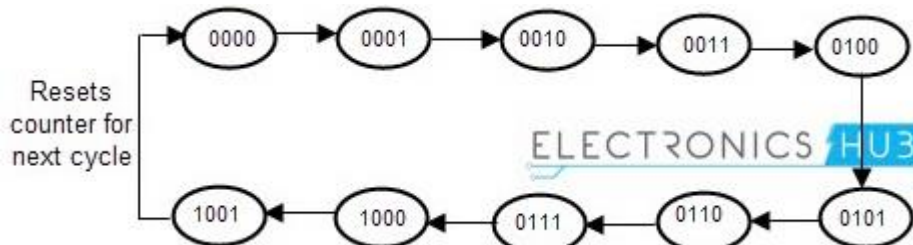
The above table describes the counting operation of Decade counter. It represents the count of circuit for decimal count of input pulses. The NAND gate output is zero when the count reaches 10 (1010).

The count is decoded by the inputs of NAND gate X1 and X3. After count 10, the logic gate NAND will trigger its output from 1 to 0, and it resets all flip flops.

State Diagram of Decade Counter

The state diagram of Decade counter is given below. If we observe the decade counter circuit diagram, there are four stages in it, in which each stage has single flip flop in it. So it is capable of counting 16 bits or 16 potential states, in which only 10 are used. The count starts from 0000 (zero) to 1001 (9) and then the NAND gate will reset the circuit.

Multiple counters are connected in series, to count up to any desired number. The number that a counter



circuit can count is called “Mod” or “Modulus”. If a counter resets itself after counting n bits is called “Mod- n counter” “Modulo- n counter”, where n is an integer.

The Mod n counter can calculate from 0 to $2n-1$. There are several types of counters available, like Mod 4 counter, Mod 8 counter, Mod 16 counter and Mod 5 counters etc.

Verilog program for 4 bit binary counter

Simulation Results:

Verilog program BCD counter

Simulation Results:

Questions:

1. Differentiate between synchronous and asynchronous counter?
2. What logic gets synthesized when I use an integer instead of a reg variable as a storage element? Is use of integer recommended?
3. What value is inferred when multiple procedural assignments made to the same reg variable in an always block?
4. How does the sensitivity list of a combinatorial always block affect pre- and post- synthesis simulation? Is this still an issue lately?
5. What value is inferred when multiple procedural assignments made to the same reg variable in an always block?
6. What does the logic in a function get synthesized into? What are the area and timing implications of calling functions in RTL?
7. What are a few important considerations while writing a Verilog function?
8. Design a 4 bit gray counter
9. What do you mean by binary counter?
10. What is BCD counter?
11. Convert 4257 to its BCD equivalent.

10. Finite State Machine Design

Objective: To develop the source code for Finite State Verilog in order to obtain the simulation, synthesis, place and route to implement into FPGA.

Scope:

- To understand the Verilog coding of Mealy and Moore Finite state machine
- To verify the Mealy and Moore Finite state machine using wave form viewer against the truth table
- To implement the Mealy and Moore Finite state machine on FPGA development board

Electronic Design Automation Tools and Apparatus required:

- Xilinx Spartan 3 FPGA
- Xilinx ISE Simulator tool
- Xilinx XST Synthesis tool
- Xilinx Project Navigator 8.1i
- JTAG cable
- Adaptor 5v/4A

Theory:

Basically a FSM consists of combinational, sequential and output logic. Combinational logic is used to decide the next state of the FSM, sequential logic is used to store the current state of the FSM. The output logic is a mixture of both combo and seq logic as shown in the figure below.

Types of State Machines

There are many ways to code these state machines, but before we get into the coding styles, let's first understand the basics a bit. There are two types of state machines:

- Mealy State Machine : Its output depends on current state and current inputs. In the above picture, the blue dotted line makes the circuit a mealy state machine.
- Moore State Machine : Its output depends on current state only. In the above picture, when blue dotted line is removed the circuit becomes a Moore state machine.

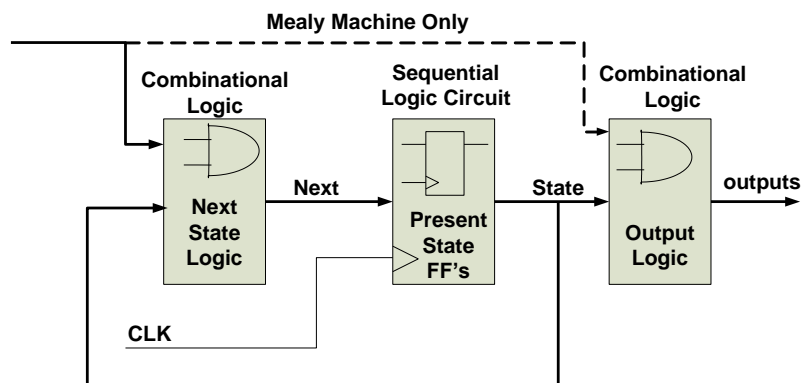


Fig: Mealy & Moore FSMs

Combinational always blocks are always blocks that are used to code combinational logic functionality and are strictly coded using blocking assignments. A combinational always block has a combinational sensitivity list, a sensitivity list without "posedge" or "negedge" Verilog keywords.

Sequential always blocks are always blocks that are used to code clocked or sequential logic and are always coded using nonblocking assignments. A sequential always block has an edge-based sensitivity list.

Encoding Style

Since we need to represent the state machine in a digital circuit, we need to represent each state in one of the following ways:

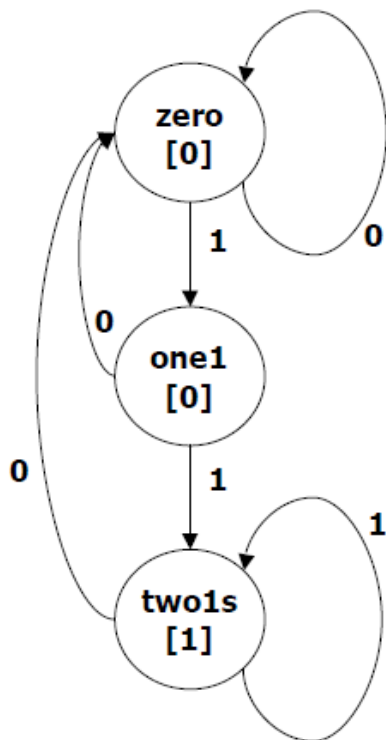
Binary encoding: each state is represented in binary code (i.e. 000, 001, 010....)

Gray encoding: each state is represented in gray code (i.e. 000, 001, 011,...)

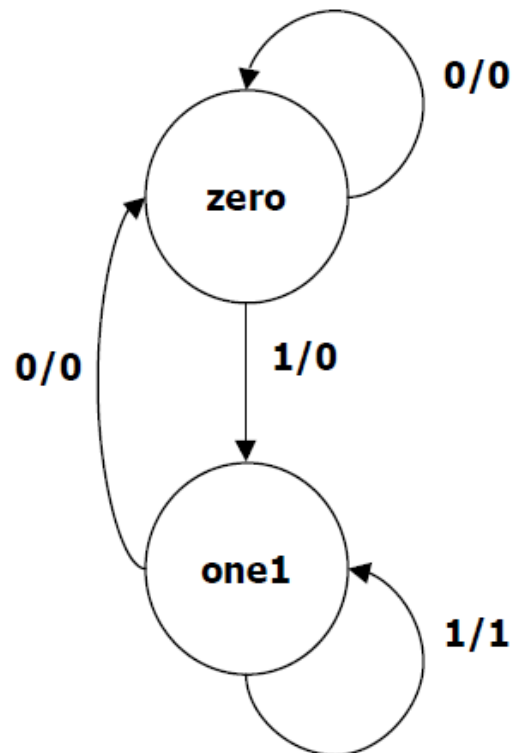
One Hot: only one bit is high and the rest are low (i.e. 0001, 0010, 0100, 1000)

One Cold: only one bit is low, the rest are high (i.e. 1110, 1101, 1011, 0111)

State diagram for Moore and Mealy machine



Detect 2 Consecutive 1 inputs (Moore)



Detect 2 Consecutive 1 inputs (Mealy)

Verilog coding for moore FSM

Simulation Results:

Verilog coding for Mealy Machine:

Simulation Results:

Viva Questions:

1. Differentiate between Mealy and Moore State machine?
2. Why is one-hot encoding preferred for FSMs designed for high-speed designs?
3. How do the `ifdef, `ifndef, `elsif, `endif constructs aid in minimizing area?
4. Illustrate the differences between binary encoding and onehot encoding mechanisms state machines
5. Explain a reversed case statement, and how it can be useful to infer a one-hot statemachine?
6. What are the differences between synchronous and asynchronous state machines?
7. Write a Moore and Mealy state mechine using verilog to detect sequence 11001.
8. How can I override a module's parameter values during instantiation?
9. What are the rules governing parameter assignments?
10. How do I prevent selected parameters of a module from being overridden during instantiation?
11. What are the differences between using `define, and using either parameter or defparam for specifying variables?
12. What are the pros and cons of specifying the parameters using the defparam construct vs. specifying during instantiation?
13. What is the difference between the specparam and parameter constructs?
14. What are derived parameters? When are derived parameters useful, and what are their limitations?

Layout Design Rules

Types of Design Rules

The design rules primary address two issues:

1. The geometrical reproduction of features that can be reproduced by the mask making and lithographical process and
2. The interaction between different layers.

There are primarily two approaches in describing the design rules.

1. Linear scaling is possible only over a limited range of dimensions.
2. Scalable design rules are conservative .This results in over dimensioned and less dense design.
3. This rule is not used in real life.

1. Scalable Design Rules (e.g. SCMOS, λ -based design rules):

In this approach, all rules are defined in terms of a single parameter λ . The rules are so chosen that a design can be easily ported over a cross section of industrial process, making the layout portable .Scaling can be easily done by simply changing the value of.

The key disadvantages of this approach are:

2. Absolute Design Rules (e.g. μ -based design rules) :

In this approach, the design rules are expressed in absolute dimensions (e.g. $0.75\mu\text{m}$) and therefore can exploit the features of a given process to a maximum degree. Here, scaling and porting is more demanding, and has to be performed either manually or using CAD tools .Also, these rules tend to be more complex especially for deep submicron.

The fundamental unity in the definition of a set of design rules is the minimum line width .It stands for the minimum mask dimension that can be safely transferred to the semiconductor material .Even for the same minimum dimension, design rules tend to differ from company to company, and from process to process. Now, CAD tools allow designs to migrate between compatible processes.

Layer Representations

With increase of complexity in the CMOS processes, the visualization of all the mask levels that are used in the actual fabrication process becomes inhibited. The layer concept translates these masks to a set of conceptual layout levels that are easier to visualize by the circuit designer. From the designer's viewpoint, all CMOS designs have the following entities:

- Two different substrates and/or wells: which are p-type for NMOS and n-type for PMOS.
- Diffusion regions (p+ and n+): which defines the area where transistors can be formed.
- These regions are also called active areas. Diffusion of an inverse type is needed to implement Contacts to the well or to substrate. These are called select regions.
- Transistor gate electrodes : Polysilicon layer
- Metal interconnect layers
- Interlayer contacts and via layers.

The layers for typical CMOS processes are represented in various figures in terms of:

- A color scheme (Mead-Conway colors).
- Other color schemes designed to differentiate CMOS structures.
- Varying stipple patterns
- Varying line styles



















Layer Description	Representation				
metal	 m1	 m2	 m3	 m4	 m5
well	 nw				
polysilicon	 poly				
contacts & vias	 ct	 v12,v23,v34,v45	 nwc	 pwc	
active area and FETs	 ndif	 pdif	 nfct	 pfct	
select	 nplus	 pplus	 prb		

Figure: Mead Conway Color coding for layers.

An example of layer representations for CMOS inverter using above design rules is shown below-

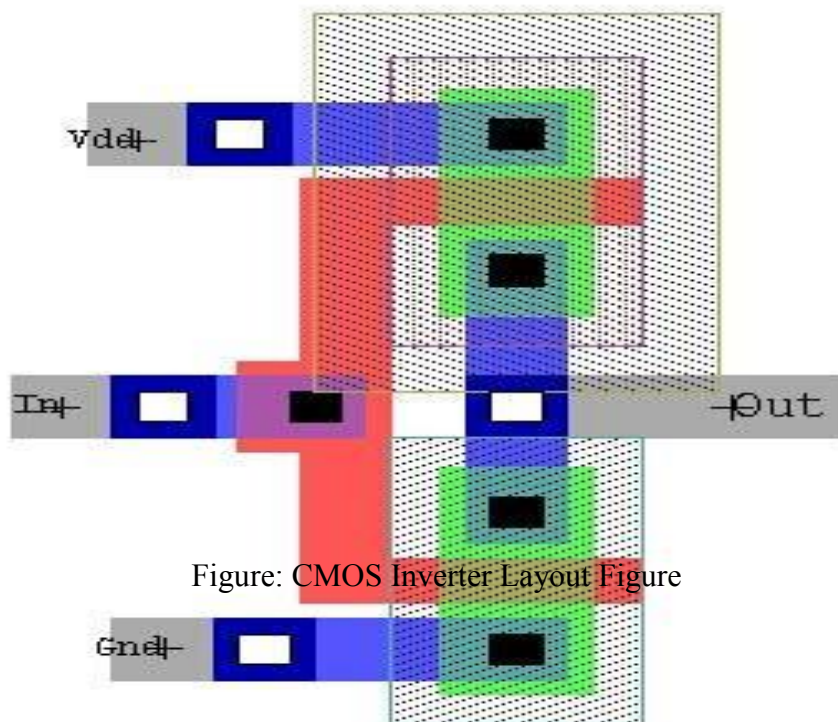


Figure: CMOS Inverter Layout Figure

STICK DIAGRAMS

Another popular method of symbolic design is "**Sticks**" layout. In this, the designer draws a freehand sketch of a layout, using colored lines to represent the various process layers such as diffusion, metal and polysilicon. Where polysilicon crosses diffusion, transistors are created and where metal wires join diffusion or polysilicon, contacts are formed.

This notation indicates only the relative positioning of the various design components. The absolute coordinates of these elements are determined automatically by the editor using a compactor. The compactor translates the design rules into a set of constraints on the component positions, and solve a constrained optimization problem that attempts to minimize the area or cost function.

The advantage of this symbolic approach is that the designer does not have to worry about design rules, because the compactor ensures that the final layout is physically correct. The disadvantage of the symbolic approach is that the outcome of the compaction phase is often unpredictable. The resulting layout can be less dense than what is obtained with the manual approach. In addition, it does not show exact placement, transistor sizes, wire lengths, wire widths, tub boundaries.

For example, stick diagram for CMOS Inverter is shown below.

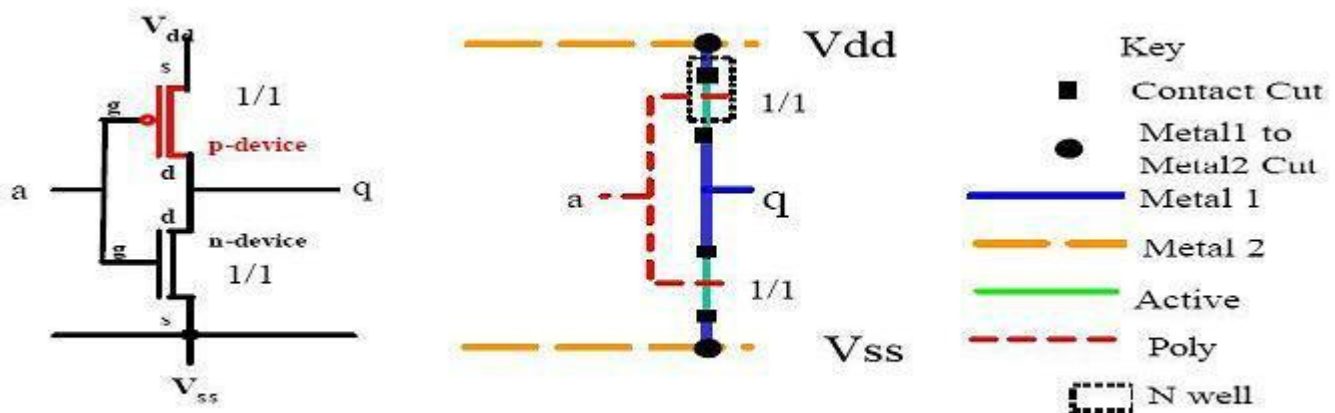


Figure : Stick Diagram of a CMOS Inverter

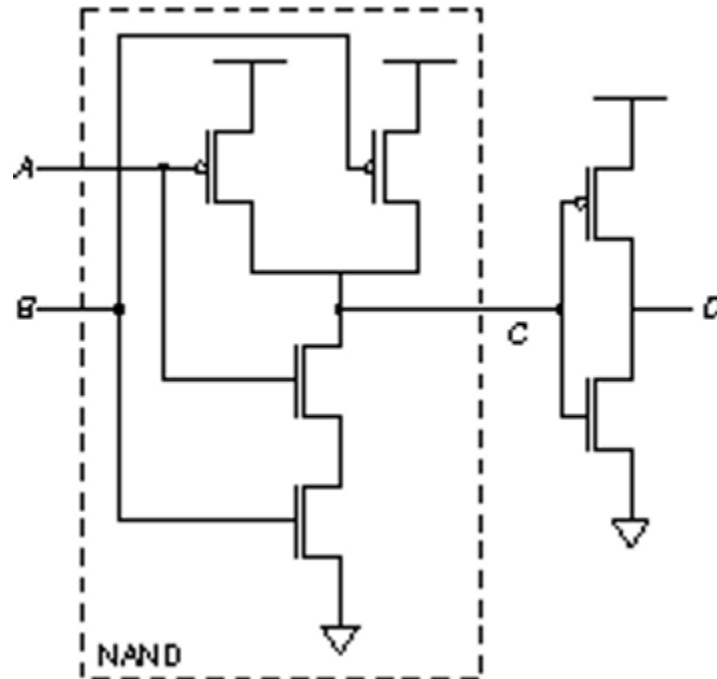
EXPERIMENT-1(a)

CMOS AND GATE

AIM: To design and simulate the CMOS AND gate and perform LVS.

TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:



PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using IC schematic.
2. Enter into Simulation mode.
3. Setup the library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using IC layout.
9. Perform DRC, LVS, PEX.

Schematic diagram:

LAYOUT diagram:

OUTPUT WAVEFORM:

RESULT:-

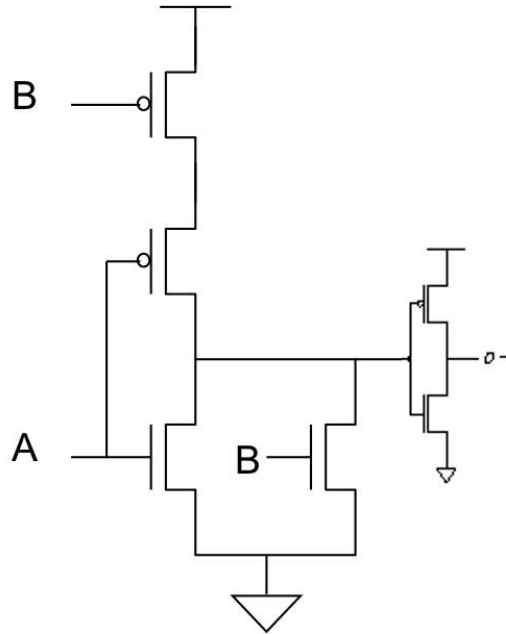
EXPERIMENT-1(b)

CMOS OR GATE

AIM: To design and simulate the CMOS OR gate and perform LVS.

TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:



PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using IC schematic.
2. Enter into Simulation mode.
3. Setup the library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using IC layout.
9. Perform DRC, LVS, PEX.

Schematic diagram:

LAYOUT diagram:

OUTPUT WAVEFORM:

RESULT:-

Viva Questions:

1. Why is NAND gate preferred over NOR gate for fabrication?
2. In CMOS technology why do we design the size of PMOS to be higher than the NMOS.
3. How do you size NMOS and PMOS transistors to increase the threshold voltage?
4. For CMOS logic, give the various techniques you know to minimize power consumption?
5. Why do we gradually increase the size of inverters in buffer design? Why not give the output of a circuit to one large inverter?
6. Give the expression for CMOS switching power dissipation?
7. What do you mean by Layout?
8. What are the types of Design rules?
9. Why Design Rules are used?
10. What is a 'well'?

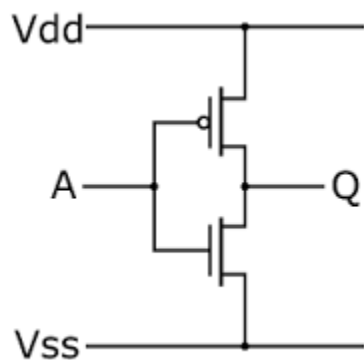
EXPERIMENT-2

CMOS INVERTER

AIM: To design and simulate the CMOS inverter and perform LVS.

TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:



Theory:

Inverter consists of nMOS and pMOS transistor in series connected between Vdd and Vss. The gate of the two transistors are shorted and connected to the input. When the input to the inverter $A = 0$, Nmos transistor is OFF and pMOS transistor is ON. The output is pull-up to Vdd. When the input $A = 1$, nMOS transistor is ON and pMOS transistor is OFF. The Output is Pull-down to Vss.

PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using IC schematic.
2. Enter into Simulation mode.
3. Setup the library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using IC layout.
9. Perform DRC, LVS, PEX.

Schematic diagram:

LAYOUT diagram:

OUTPUT WAVEFORM:

RESULT:-

Viva Questions

1. Explain CMOS Fabrication steps.
2. What are the advantages of CMOS over NMOS and PMOS?
3. What is Latch Up? Explain Latch Up with cross section of a CMOS Inverter. How do you avoid Latch Up?
4. What happens when the PMOS and NMOS are interchanged with one another in an inverter?
5. Explain sizing of the inverter?
6. What are the limitations in increasing the power supply to reduce delay?
7. Why do we gradually increase the size of inverters in buffer design when trying to drive a high capacitive load?
8. All of us know how an inverter works. What happens when the PMOS and NMOS are interchanged with one another in an inverter?

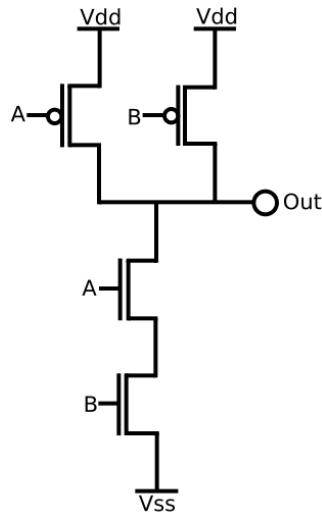
EXPERIMENT-3(a)

CMOS NAND GATE

AIM: To design and simulate the CMOS NAND gate and perform LVS.

TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:



PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using IC schematic.
2. Enter into Simulation mode.
3. Setup the library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using IC layout.
9. Perform DRC, LVS, PEX.

Schematic diagram:

LAYOUT diagram:

OUTPUT WAVEFORM:

RESULT:-

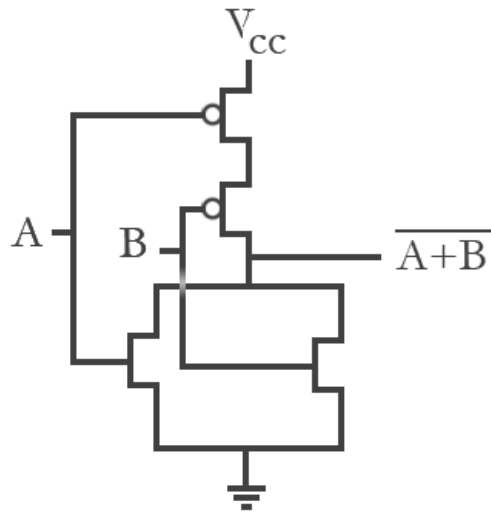
EXPERIMENT-3(b)

CMOS NOR GATE

AIM: To design and simulate the CMOS NOR gate and perform LVS.

TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:



PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using IC schematic.
2. Enter into Simulation mode.
3. Setup the library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using IC layout.
9. Perform DRC, LVS, PEX.

Schematic diagram:

LAYOUT diagram:

OUTPUT WAVEFORM:

RESULT:-

Viva Questions

1. Draw the symbols for n -channel and p -channel DE-MOSFET and E-only MOSFET.
2. What are the main CMOS fabrication technologies available?
3. Compare CMOS and Bipolar technologies?
4. Implement NAND gate using CMOS and give the corresponding equation also.
5. What is the function of 'thinox' in a MOSFET?
6. What is Noise Margin? Explain the procedure to determine Noise Margin
7. What happens to delay if you increase load capacitance?
8. What happens to delay if we include a resistance at the output of a CMOS circuit?
9. How does Resistance of the metal lines vary with increasing thickness and increasing length?
10. What is Body Effect?

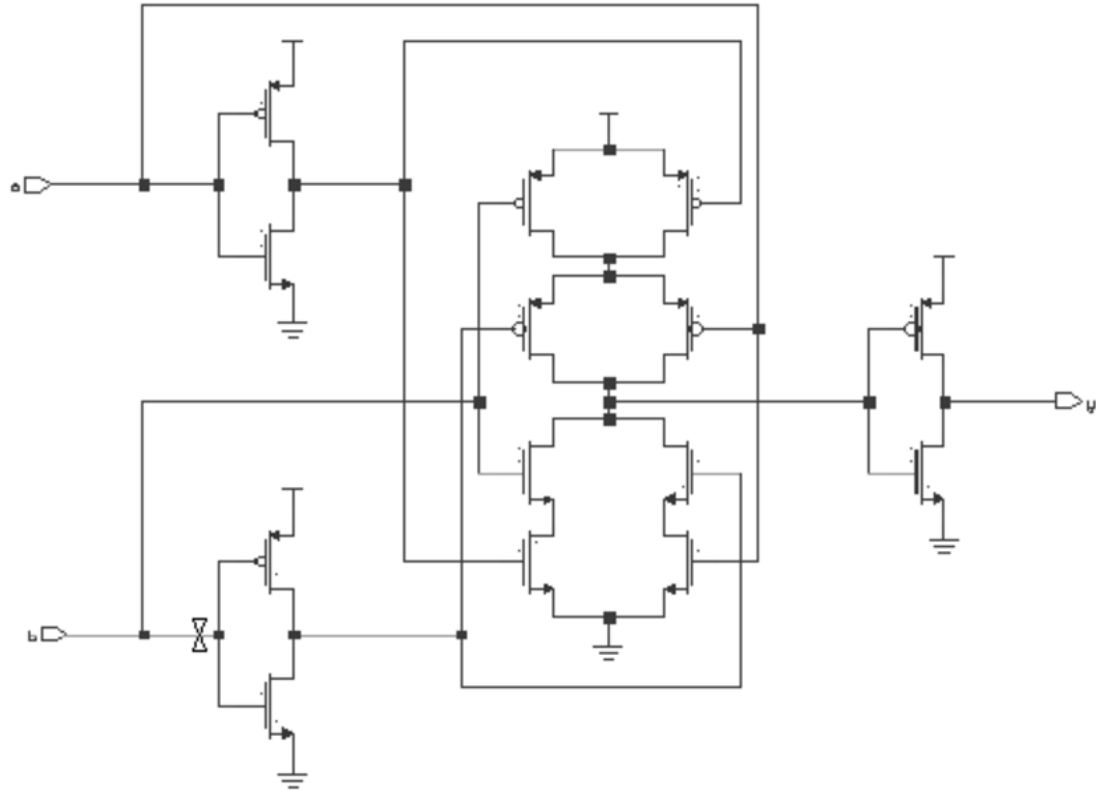
EXPERIMENT-4(a)

CMOS XOR GATE

AIM: To design and simulate the CMOS XOR gate and perform LVS.

TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:



PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using IC schematic.
2. Enter into Simulation mode.
3. Setup the library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using IC layout.
9. Perform DRC, LVS, PEX.

Schematic diagram:

LAYOUT diagram:

OUTPUT WAVEFORM:

RESULT:-

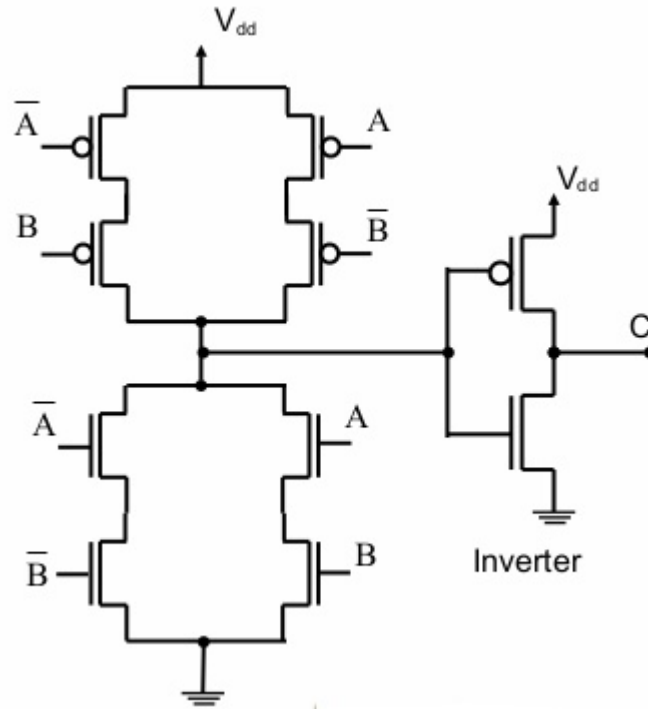
EXPERIMENT-4(b)

CMOS XNOR GATE

AIM: To design and simulate the CMOS XNOR gate and perform LVS.

TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:



PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using IC schematic.
2. Enter into Simulation mode.
3. Setup the library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using IC layout.
9. Perform DRC, LVS, PEX.

Schematic diagram:

LAYOUT diagram:

OUTPUT WAVEFORM:

RESULT:-

Viva Questions (Exp.-4)

1. Write the truth tables of XOR and XNOR, also give their logical symbols.
2. Implement XOR/XNOR gate using CMOS and give their corresponding equations also.
3. What do you mean by 'Active area'?
4. Give applications of XOR/XNOR gates.
5. What is Mentor Graphics? Applications?
6. Why is the substrate in NMOS connected to Ground and in PMOS to VDD?
7. What is the fundamental difference between a MOSFET and BJT ?
8. Which transistor has higher gain. BJT or MOS and why?

EXPERIMENT-5

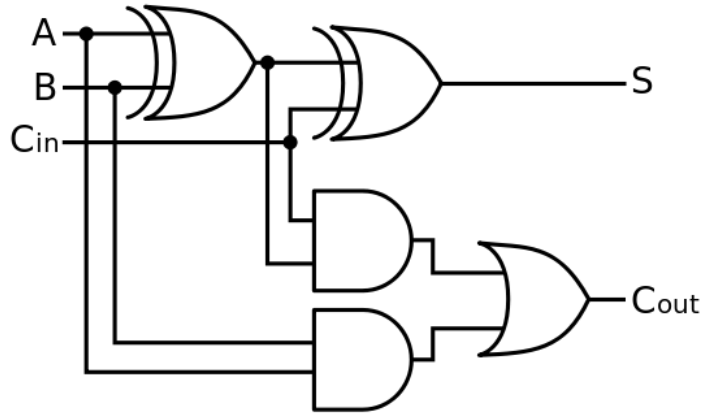
CMOS FULL ADDER

AIM: To design and simulate the CMOS full adder.

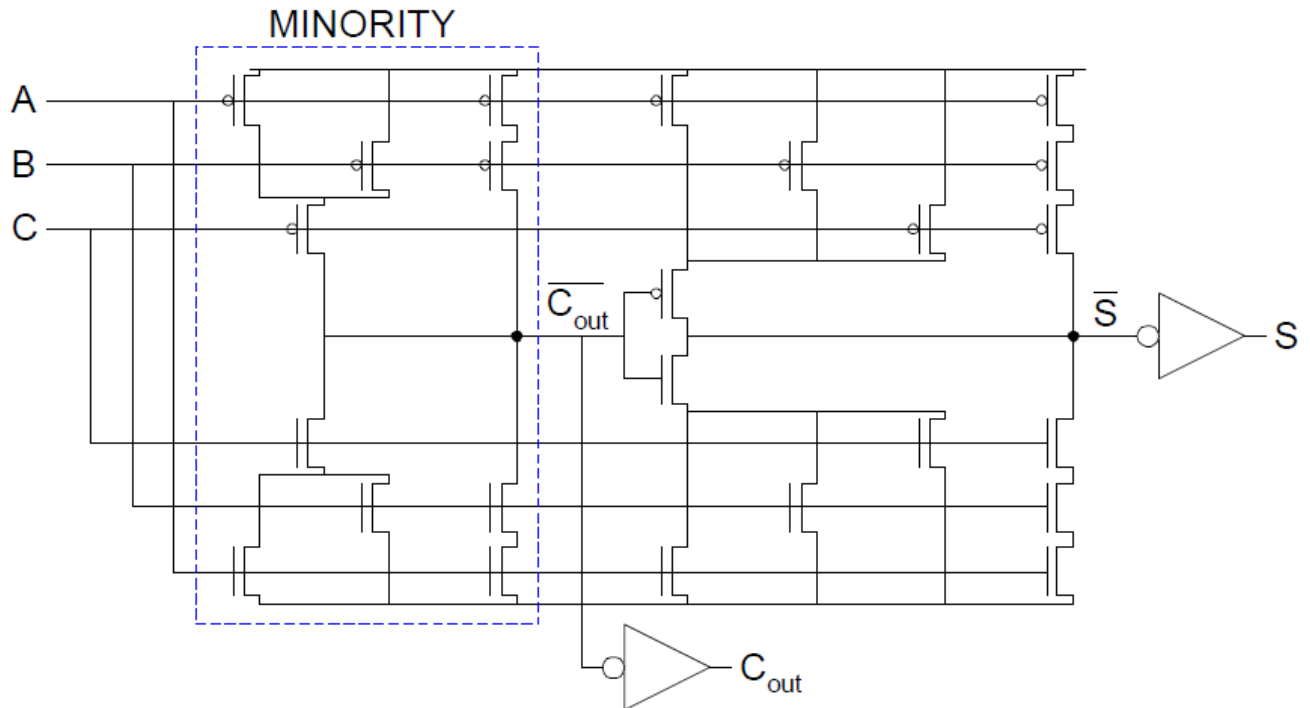
TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:

a) Using gates



b) Using CMOS



PROCEDURE:

- Connect the Circuit as shown in the circuit diagram using IC schematic.
- Enter into Simulation mode.
- Setup the library.
- Setup the required analysis.
- Probe the required Voltages
- Run the simulation.
- Observe the waveforms in EZ wave.

OUTPUT WAVEFORMS:**RESULT:**

Viva Questions

- 1.** What is the difference between 'Xilinx' and 'Mentor Graphics' softwares?
- 2.** Write logical expressions for Half and Full Adders.
- 3.** What is the advantage of using CMOS technology to implement Full Adder?
- 4.** Write expression for Full Subtractor.
- 5.** Draw diagram of Full Adder using logic gates.

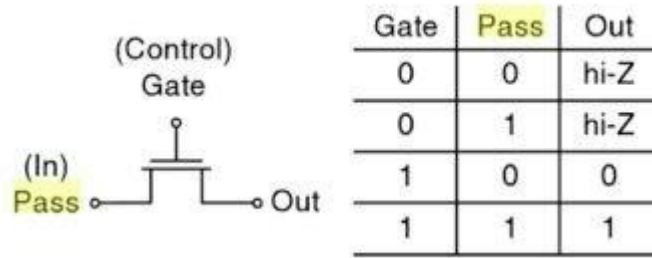
EXPERIMENT-6

NMOS PASS TRANSISTOR

AIM: To design and simulate the pass transistor circuit and perform LVS.

TOOLS: Mentor Graphics ICstudio, ICschematic, IClayout, Calibre.

CIRCUIT DIAGRAM:



PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using IC schematic.
2. Enter into Simulation mode.
3. Setup the library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using IC layout.
9. Perform DRC, LVS, PEX.

Schematic diagram:

LAYOUT diagram:

OUTPUT WAVEFORM:

RESULT:-

Viva Questions

1. Explain working of NMOS PASS TRANSISTOR.
2. What are the advantages and disadvantages of NMOS pass transistor?
3. Draw NMOS Pass Transistor Logic and Transmission Gate Logic.
4. NMOS is strong inwhile PMOS is strong in
5. In CMOS technology, in digital design, why do we design the size of PMOS to be higher than the NMOS. What determines the size of PMOS wrt NMOS. Though this is a simple question try to list all the reasons possible?
6. Why PMOS and NMOS are sized equally in a Transmission Gates?
7. Give 5 important Design techniques you would follow when doing a Layout for Digital Circuits?
8. What is metastability? When/why it will occur? Different ways to avoid this?