Department of CSE
Pandit Deendayal Energy University (PDEU)
Digital Electronics and Computer Organization Lab (DECO)
20CP203P

Lab 1: Basic GATE operations
(Week 01-Aug-2022 to 07-Aug-2022)

In this lab, you will learn how Logisim works and will simulate some sample blocks.
This will help you learn more about digital system design.

Logisim (http://www.cburch.com/logisim/) is a Java application, it can be obtained from http://sourceforge.net/projects/circuit/. Note: JDK installation is prerequisite before using the Logisim. Simply download and run it (tutorial is available help->tutorial).

Learn the interface of the software and learn how basic gates works, how to give input and output signals.

1. Design the following logic functions using basic gates and test the output using truth table.
   - 1.1 Y = ABC + DE + F
   - 1.2 Z = A'B' + AB' + A'B + AB
   - 1.3 K = (A'+B') (A+B') (A'+B) (A+B)
2. Design the verify the functionality using existing library of basic gates in logisim.
   - 2.1 Y = ((A+B).(C+D+E))'
   - 2.2 F = ((A.B')'+((A)'(B)')')'
   - 2.3 K = (((A)'+B)'. ((A)'+(B)')')'
3. For each of the following problem statement, design a circuit, describe a Truth table and verify the outcome using Logisim.
   - 3.1 Implement the functionality of XOR gate using only NAND gates.
   - 3.2 Implement the functionality of XNOR gate using only NOR gates.
   - 3.3 Implement the functionality of AND gate using only NAND gates.
   - 3.4 Implement the functionality of OR gate using only NOR gates.
   - 3.5 Implement the functionality of NOT gate using only NAND gate.

Submission Instructions:
Prepare the submission file according to the following steps.

1. Copy (Snip/snapshot) the Design circuit and past into the word file.
2. Prepare a corresponding Truth table.
3. Repeat step 1 and 2 for all the problems.
4. For each problem the corresponding circuit design figure must be number accordingly. For example (Fig 1.1 The Circuit Design of XOR using NAND Only).
5. Convert it into pdf file, name it as **RollNo_Assignment# (Example: 20BCP001_ Assignment1.pdf)**.

- Print the corresponding PDF file and prepare a file (Hard copy).

- **The assignment of the previous lab will be verified in the very next lab. Therefore, it is mandatory to bring the file in each lab.**
- Late submission will lead to penalty.
- Any form of plagiarism/copying from peer or internet sources will lead penalty.
- Following of all instructions at submission time is mandatory. Missing of any instructions at submission time will lead to penalty of marks.

Department of CSE
Pandit Deendayal Energy University (PDEU)
Digital Electronics and Computer Organization Lab (DECO)
20CP203P

Lab 2: Introduction to Verilog
(Week 02)

In this Lab, we shall start Verilog. To run Verilog, we shall use http://www.edaplayground.com. After opening you will find Design or Testbench window pane. In the Design window you need write the Verilog Code and in the Testbench window, the testbench verification code will be written.

1. Write a Verilog code to implement AND gate. Write the corresponding Testbench code for the verification of your Verilog code.

   Truth Table of AND Gate:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

2. Write a Verilog code to implement OR gate. Write the corresponding Testbench code for the verification of your Verilog code.

   Truth Table of OR Gate:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

3. Write a Verilog code to implement XOR gate. Write the corresponding Testbench code for the verification of your Verilog code.

   Truth Table of XOR Gate:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

4. Write a Verilog code to implement NOR gate. Write the corresponding Testbench code for the verification of your Verilog code.

   Truth Table of NOR Gate:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

5. Write a Verilog code to implement NAND gate. Write the corresponding Testbench code for the verification of your Verilog code.

   Truth Table of NAND Gate:

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

6. Write a Verilog code to implement XNOR gate. Write the corresponding Testbench code for the verification of your Verilog code.

   Truth Table of XNOR Gate:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

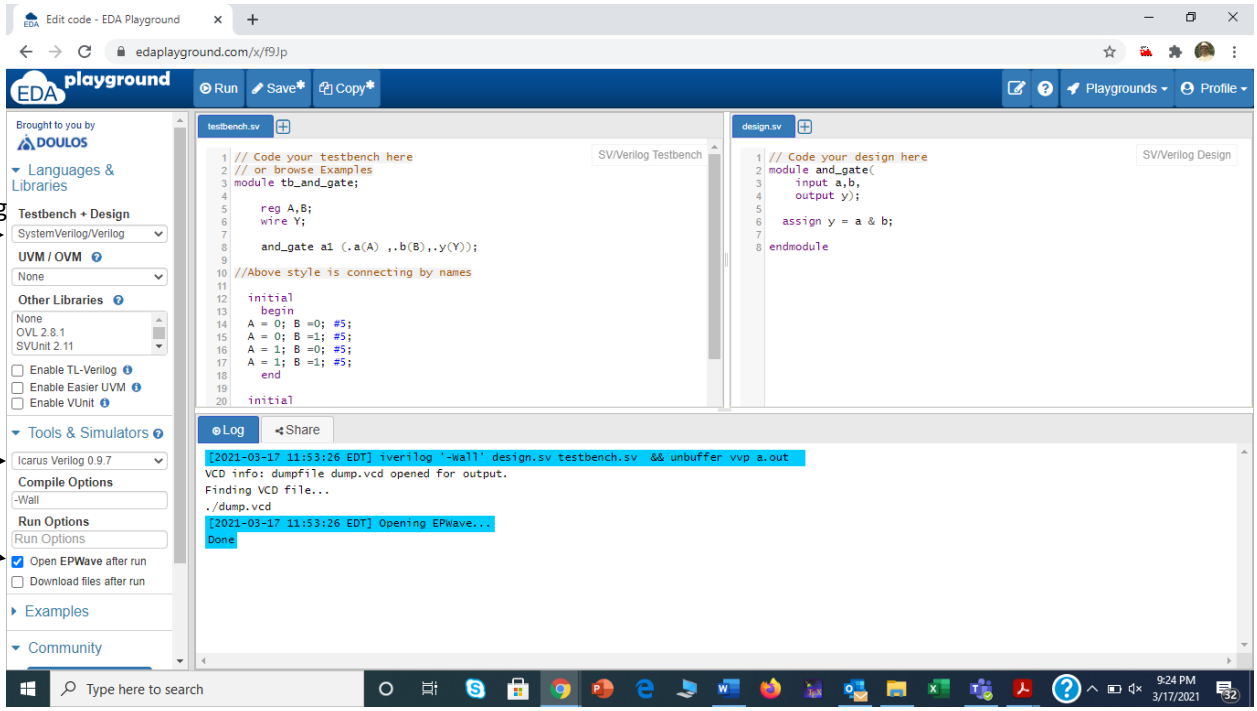
Procedure to run the programs in EDAPlayground:

- **Open** http://www.edaplayground.com
- **Go to Design Window to write the Verilog Code.**
- **Go to Testbench Window to write the Testbench Verification Code.**
- **All the other setup should be done according to the following Screenshot**

(1)
Select SystemVerilog

(2)
Select Icarus Verilog 0.9.7 in Tools and Simulator

(3)
Make it sure the check box has been clicked “Open EPWave after run”



- **Sample Verilog Code for AND Gate:**

```
module and gate(
    input a,b,
    output y);
    assign y = a & b;
endmodule
```

- **Sample Testbench Code for the AND Gate:**

```
module tb_and_gate;

    reg A,B;
    wire Y;
    and_gate a1 (.a(A) ,.b(B),.y(Y));

initial
    begin
        A = 0; B =0; #5;
        A = 0; B =1; #5;
        A = 1; B =0; #5;
        A = 1; B =1; #5;
    end
initial
```

```
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end

endmodule
```

Submission Instructions:

Prepare the submission file according to the following steps.

1. Copy (Snip/snapshot) the Design circuit and past into the word file.
2. Prepare a corresponding Truth table.
3. Repeat step 1 and 2 for all the problems.
4. For each problem the corresponding circuit design figure must be number accordingly. For example (Fig 1.1 The Circuit Design of XOR using NAND Only).
5. Convert it into pdf file, name it as **RollNo_Assignment# (Example: 20BCP001_ Assignment1.pdf)**.

- Print the corresponding PDF file and prepare a file (Hard copy).
- **The assignment of the previous lab will be verified in the very next lab. Therefore, it is mandatory to bring the file in each lab.**
- Late submission will lead to penalty.
- Any form of plagiarism/copying from peer or internet sources will lead penalty.
- Following of all instructions at submission time is mandatory. Missing of any instructions at submission time will lead to penalty of marks.

Department of Computer Science and Engineering
Pandit Deendayal Energy University
Digital Electronics and Computer Organization Lab – **20CP203P**

Lab Assignment 3

In this Lab, we shall learn 'Module Instantiation' as well as writing a Verilog code using a 'Structural' and 'Behavioral' method. You will write the code for one module; the same module will be used in other module of your Verilog Code.

A module provides a template from which you can create actual objects. When a module is invoked, Verilog creates a unique object from the template. Each object has its own name, variables, parameters, and I/O interface. The process of creating objects from a module template is called instantiation, and the objects are called instances.

Previously, we have used these instances in the Testbench code. Now we shall use it in Verilog Code also.

Question: 1 Implement the following expression using the Verilog Hardware Description Language (HDL) then compare with truth table whether your circuit produced same output or not? Moreover, verify your circuit against the waveform.

1. F(A, B, C)= A'BC + AB'C + ABC
2. F(A, B, C, D)=ABCD' + A'BC D+ AB'CD' + ABC
3. F(A, B, C, D, E, F) = ABC + DE + F
4. F(A, B) = (A'+B') (A+B') (A'+B) (A+B)
5. F(A, B) = ((A.B')'+((A)'(B)')')'
6. F(A, B) = (((A)'+B)'. ((A)'+(B)')')'

Question: 2 Implement the XOR gate using Behavioural and Structural code (shown in Fig: 1) of Verilog Hardware Description Language.

```
 1   module Lab4_Build_XOR(a, b, c);
 2   input a, b;
 3   output c;
 4   wire a_not, b_not;
 5   wire x, y;
 6   not(a_not, a);
 7   not(b_not, b);
 8   and(x, a_not, b);
 9   and(y, a, b_not);
10   or(c, x, y);
11   endmodule
```

Fig. 1: Example Structural code of XOR gate operation.

Question: 3: Implement the following expression using the Verilog Hardware Description Language (HDL) (Structural Coding).

1. F (A, B, C) = (A'+B'+C')(A+B'+C')(A'+B+C')(A'+B')`
2. F (A, B, C, D, E) = ((A+B).(C+D+E))'

Question: 3 Implement the following expression using universal NAND and NOR gate. Write down Verilog Structural and Behavioral code for that expression.

1. F (A, B, C) = (AB'C) + (AB'C')
2. F (A, B) = A'B' + AB' + A'B + AB

Question: 4 Write down three modules in a single Verilog code to design AND, OR and NOT gate. Now use those modules to design the XOR gate. Use AND, OR and NOT gate as the instances to implement the XOR gate. Write the corresponding Testbench code for the verification of your XOR gate.

Truth Table of XOR Gate:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Question: 5 Consider the following expression:

Y = A'.B'.C' + A'.B.C' + A.B'.C' + A.B'.C

(i) Generate the Truth Table manually for the same.

(ii) Design one three input 'And' gate module and one four input 'OR' gate module using Verilog. Instantiate those two modules to design the above-mentioned expression. Design the corresponding TestBench code for the verification purpose.

(iii) Now Minimize the given expression.

(iv) Design again 'AND' gate and 'OR' gate module with required number of inputs and instantiate them to implement the minimized expression. Design the corresponding TestBench code for the verification purpose.

- **Sample Verilog Design Code for Instantiation of module:**

```
module not_gate (input e, output f);
  assign f = ~e;
```

```
endmodule

module and_gate (input a, b, output c);
  assign c = a & b;
endmodule

module or_gate (input p, q, output r);
  assign r = p | q;
endmodule

module build_xor (input m, n, output o);
  wire x, y, a_not, b_not;
  not_gate n1 (.e(m),.f(a_not));
  not_gate n2 (.e(n),.f(b_not));
  and_gate a1 (.a(a_not),.b(n),.c(x));
  and_gate a2 (.a(m),.b(b_not),.c(y));
  or_gate o1 (.p(x),.q(y),.r(o));

endmodule
```

- **Sample TestBench Code for Question 1:**

```
module tb_xor_str;
   reg A,B;
   wire C;
 build_xor x1 (.m(A), .n(B), .o(C));
 initial
   begin
 A = 0; B =0; #5;
 A = 0; B =1; #5;
 A = 1; B =0; #5;
 A = 1; B =1; #5;
   end
 initial
 begin
   $dumpfile("dump.vcd");
   $dumpvars(1);
 end
endmodule
```

Submission Instructions:

- Prepare the submission file according to the following process:
    1. Copy the Verilog code, the Test Bench Code in a Word File.
    2. Take the ScreenShot of Waveform and paste into the same word file.

3. Repeat Step 1 and 2 for all the programs.
4. Copy and Paste all the Verilog code, Testbench Code and Waveform into a single word file as 1_verilog, 1_TestBench, 1_Waveform, 2_verilog, 2_TestBench, 2_Waveform… etc.
5. Convert it into pdf file, Print it and prepare a file for the verification in the next lab.

Department of Computer Science and Engineering
Pandit Deendayal Energy University
Digital Electronics and Computer Organization Lab – **20CP203P**

Lab 4: Design a Half Adder, and Full Adder using Half Adder

Question 1 Write Verilog code for half adder. Test using waveform.

Question 2. Design a four-bit combinational circuit 2's complement using exclusive-OR gates and half adder. Write Verilog code in Edaplayground and then test using waveform.

Question 3. Write Verilog code for full adder. Test using waveform.

Question 4. Write Verilog code for full adder using module instantiation of half adder. Test using waveform.

Question 5. Write Verilog code for half adder using only NAND gate. Test using waveform.

Question 6. Write Verilog code for half adder using only NOR gate. Test using waveform.

Description:

Half Adder (HA):

Half adder is the simplest of all adder circuits. Half adder is a combinational arithmetic circuit that adds two numbers and produces a sum bit (s) and carry bit (c) both as output. The addition of 2 bits is done using a combination circuit called a Half adder. The input variables are augend and addend bits and output variables are sum & carry bits. A and B are the two input bits.

let us consider two input bits A and B, then sum bit (s) is the X-OR of A and B. it is evident from the function of a half adder that it requires one X-OR gate and one AND gate for its construction.

Truth Table:

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

Table 4.1: Truth table for "Sum" and "Carry".

Here we perform two operations Sum and Carry, thus we need two K-maps one for each to derive the expression.

Logical Expression:

For Sum:

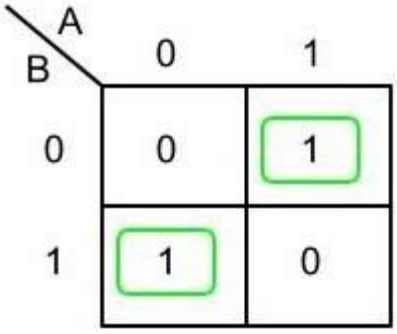


Fig 4.1: K-map for “Sum”

Sum = A XOR B

For Carry:

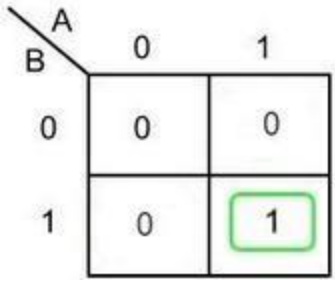


Fig 4.2: K-map for “Carry”

Carry = A AND B

Implementation:

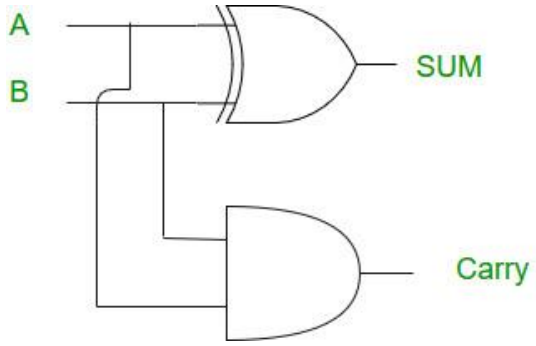


Fig 4.3: Circuit Diagram of "Half-Adder".

Full Adder is the adder that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another. we use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit. A 1-bit full adder adds three operands and generates 2-bit results.

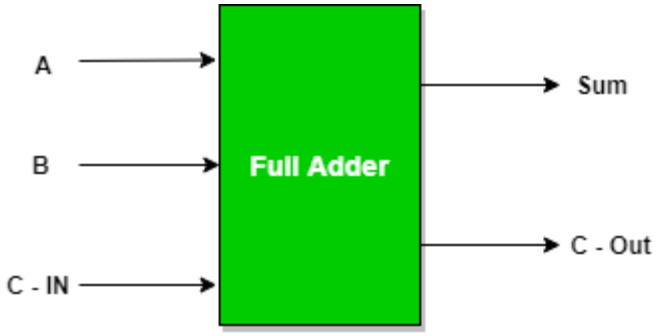


Fig 4.4: Black box view of "Full Adder".

Inputs			Outputs	
A	B	C – IN	Sum	C – Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 4.2: Truth table of "Full Adder"

Logical Expression for SUM: = A' B' C-IN + A' B C-IN' + A B' C-IN' + A B C-IN = C-IN (A' B' + A B) + C-IN' (A' B + A B') = C-IN XOR (A XOR B) = (1,2,4,7)

Logical Expression for C-OUT: = A' B C-IN + A B' C-IN + A B C-IN' + A B C-IN = A B + B C-IN + A C-IN = (3,5,6,7)

Another form in which C-OUT can be implemented: = A B + A C-IN + B C-IN (A + A') = A B C-IN + A B + A C-IN + A' B C-IN = A B (1 +C-IN) + A C-IN + A' B C-IN = A B + A C-IN + A' B C-IN = A B + A C-IN (B + B') + A' B C-IN = A B C-IN + A B + A B' C-IN + A' B C-IN = A B (C-IN + 1) + A B' C-IN + A' B C-IN = A B + A B' C-IN + A' B C-IN = AB + C-IN (A' B + A B')

Therefore, COUT = AB + C-IN (A EX – OR B)

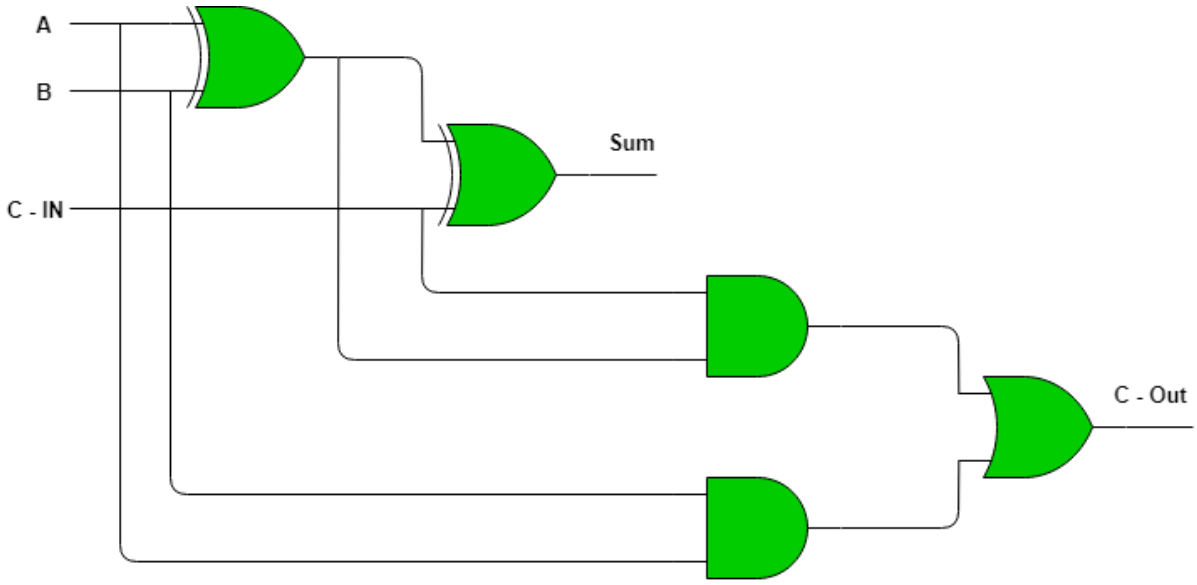


Fig 4.5: Circuit Diagram of "Full Adder"

Implementation of Full Adder using Half Adders:

2 Half Adders and an OR gate is required to implement a Full Adder.

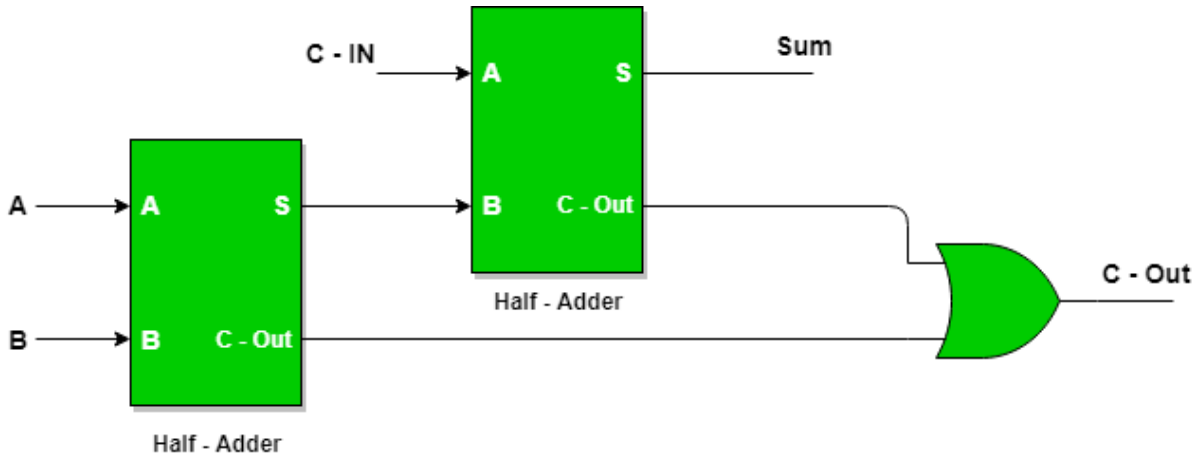


Fig 4.6: Circuit Diagram of “Full Adder” using two “Half-Adder”.

With this logic circuit, two bits can be added together, taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude.

Department of Computer Science and Engineering
Pandit Deendayal Energy University
Digital Electronics and Computer Organization Lab – **20CP203P**

Lab 5: Design a Half Subtractor, Full Subtractor using Half Subtractors

Question 1: Write a Verilog code to design a Half subtractor and test it using the Waveform.

Question 2: Write a Verilog code to design a Full subtractor using Half subtractors and test it using the Waveform.

Question 3: Write the Verilog code (either structural or behavioral) for a 4-bit ripple carry adder. The block diagram of a 4-bit ripple carry adder has been given in Figure 1 for your reference.

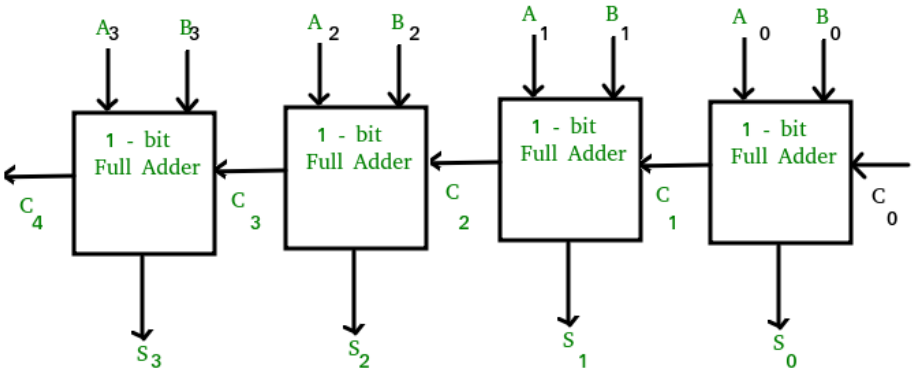


Figure 1: The Block Diagram of four bit ripple carry adder.

Question 4: Write the Verilog code to construct a 4-bit adder-subtractor using. The logic diagram of a 4-bit adder-subtractor using a ripple carry adder has been given in Figure 2 for your reference.

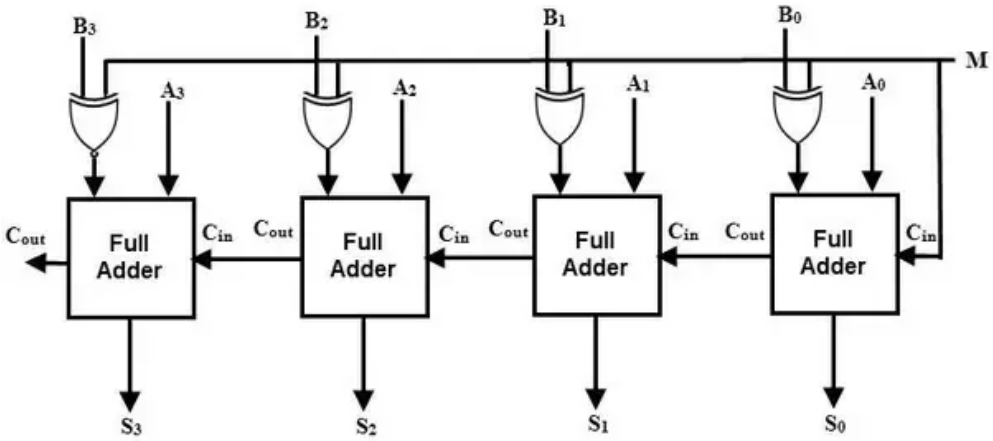


Figure 2: The Block Diagram of four bit adder-cum-subtractor bit.

Question 5: Write the Verilog code for a 4-bit carry look-ahead adder (CLA). The block diagram of the same has been given in Figure 3.

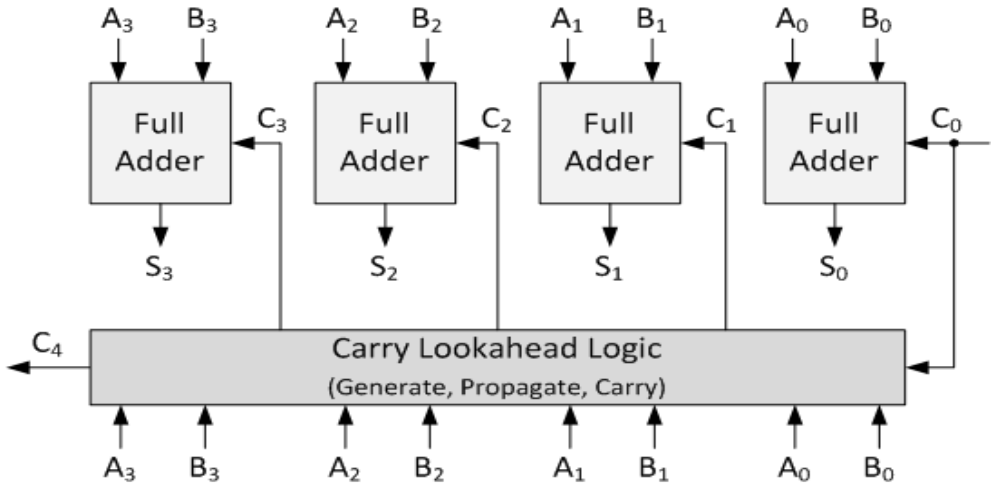


Figure 3: The Block Diagram of four bit carry look-ahead adder (CLA).

Submission Instructions:

- Prepare the submission file according to the following process:
    1. Copy the Verilog code, the Test Bench Code in a Word File.
    2. Take the ScreenShot of Waveform and paste into the same word file.
    3. Repeat Step 1 and 2 for all the programs.
    4. Copy and Paste all the Verilog code, Testbench Code and Waveform into a single word file as 1_verilog, 1_TestBench, 1_Waveform, 2_verilog, 2_TestBench, 2_Waveform… etc.

Convert it into pdf file, Print it and prepare a file for the verification in the next lab.

Department of Computer Science and Engineering
Pandit Deendayal Energy University
Digital Electronics and Computer Organization Lab – **20CP203P**

Lab 6: Designing of Multiplexer, Demultiplexer, Decoder, Encoder

Question 1: Write a Verilog code to design a 4:1 Multiplexer and verify the same.

Question 2: Write a Verilog code to design an 8:1 Multiplexer and verify the same.

Question 3: Write a Verilog code to design a 1:4 Demultiplexer and verify the same.

Question 4: Write a Verilog code to design a 1:8 Demultiplexer and verify the same.

Question 5: Write a Verilog code to design a 3:8 Decoder and verify the same.

Question 6: Write a Verilog code to design a 8:3 Encoder and verify the same.

- Multiplexer is a data selector which takes several inputs and gives a single output. In multiplexer we have 2^n input lines and 1 output lines where n is the number of selection lines.

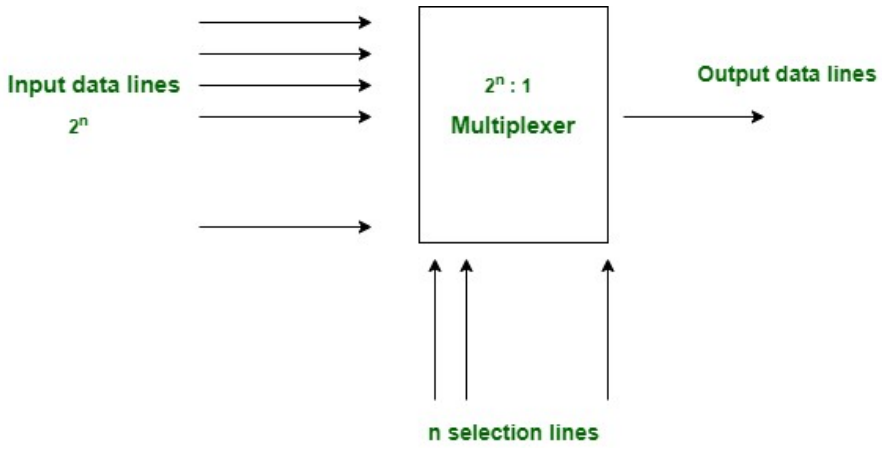


Figure 1: The Block Diagram of Multiplexer.

- Demultiplexer is a data distributor which takes a single input and gives several outputs. In demultiplexer we have 1 input and 2^n output lines where n is the selection line.

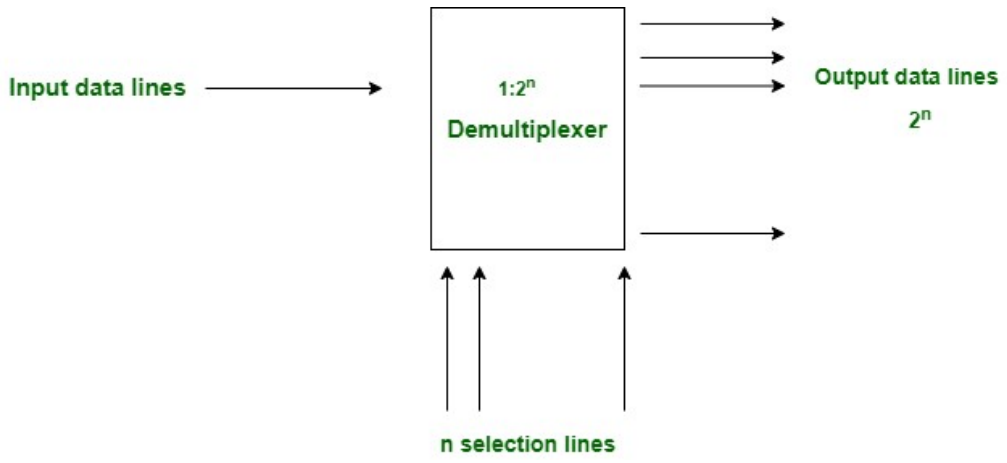


Figure 2: The Block Diagram of Multiplexer.

- A decoder does the opposite job of an encoder. It is a combinational circuit that converts n lines of input into 2^n lines of output.

X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Figure 3: Truth Table of Decoder.

- An encoder is a combinational circuit that converts binary information in the form of a 2N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time.

D7	D6	D5	D4	D3	D2	D1	D0	X	Y	Z
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Submission Instructions:

- Prepare the submission file according to the following process:
    1. Copy the Verilog code, the Test Bench Code in a Word File.
    2. Take the ScreenShot of Waveform and paste into the same word file.
    3. Repeat Step 1 and 2 for all the programs.
    4. Copy and Paste all the Verilog code, Testbench Code and Waveform into a single word file as 1_verilog, 1_TestBench, 1_Waveform, 2_verilog, 2_TestBench, 2_Waveform… etc.

Convert it into pdf file, Print it and prepare a file for the verification in the next lab.

Department of CSE
Pandit Deendayal Energy University (PDEU)
Digital Electronics and Computer Organization Lab (DECO)
20CP203P

Lab 7: Application of Decoder, Recap of Logisim

In this Lab, we shall check about some of the application of Decoder. The design using Logisim will also be checked.

1. Write a Verilog code to implement BCD to seven segment display Decoder. Prepare the Truth Table and circuits and verify the same using Test Bench.
2. Design a 3:8 Decoder using Logisim.
3. Design a 3:8 Decoder using two 2:4 Decoder using Logisim.
4. Design an 8:3 Priority Encoder using Logisim.
5. Design a 4:1 Multiplexer using Logisim.
6. Design a 1:8 Demultiplexer using Logisim.

Submission Instructions:
Prepare the submission file according to the following steps.

1. Copy (Snip/snapshot) the Design circuit and past into the word file.
2. Prepare a corresponding Truth table.
3. Repeat step 1 and 2 for all the problems.
4. For each problem the corresponding circuit design figure must be number accordingly. For example (Fig 1.1 The Circuit Design of XOR using NAND Only).
5. Convert it into pdf file, name it as **RollNo_Assignment# (Example: 20BCP001_ Assignment1.pdf)**.

- Print the corresponding PDF file and prepare a file (Hard copy).
- **The assignment of the previous lab will be verified in the very next lab. Therefore, it is mandatory to bring the file in each lab.**
- Late submission will lead to penalty.
- Any form of plagiarism/copying from peer or internet sources will lead penalty.
- Following of all instructions at submission time is mandatory. Missing of any instructions at submission time will lead to penalty of marks.

Department of Computer Science and Engineering
Pandit Deendayal Energy University
Digital Electronics and Computer Organization Lab – **20CP203P**

Lab Assignment – 8: Sequential Circuits: Latch and flip flop (SR)

In this lab we will learn about the sequential logic circuits. These circuits can act as a memory unit.

Q 1. Design an SR flip-flop using Logisim.

Q 2. An SR latch (Set/Reset) is an asynchronous device: it works independently of control signals and relies only on the state of the S and R inputs. The symbol, the circuit using NOR gates, and the truth table are shown below.

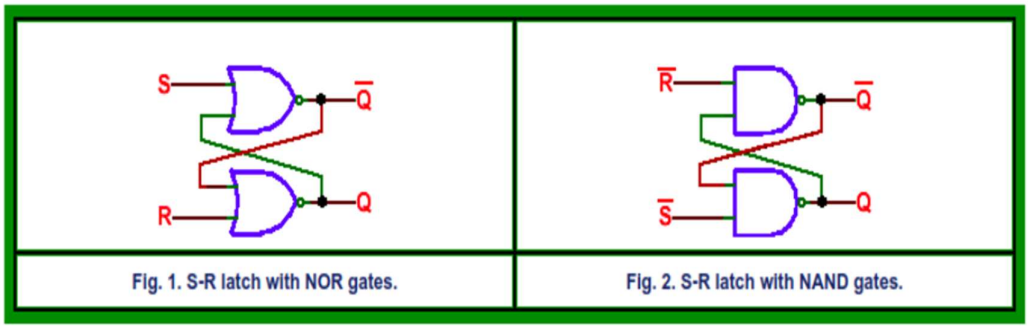


Fig. 1. S-R latch with NOR gates.

Fig. 2. S-R latch with NAND gates.

S	R	Q	Q'	
0	0	NC	NC	No change. Latch remained in present state.
1	0	1	0	Latch SET.
0	1	0	1	Latch RESET.
1	1	0	0	Invalid condition.

Table 1: S-R latch truth table

Based on the above figure and truth table, complete following tasks.

1) Generate the Boolean expression for the S-R latch from the truth table given in Table-1.

2) Write a module for NOR gate and develop a structural Verilog code for the S-R latch using the NOR gate module.

3) Validate the code via a suitable Testbench code.

Q 3. In S-R latch we do not use a clock. Now if we add an additional clock at input so that the S and R inputs are active only when the clock is high. When the clock goes low, the state of flip-flop is latched and cannot change until the clock goes high again. This clocked addition in S-R latch is also called the S-R flip flop. Use the below given figure and truth table for S-R flip flop.

We can add a clock in put in NAND latch in Figure 2, and can convert in S-R flip flop using NAND gates as: (similar results can be achieved via NOR Latch)

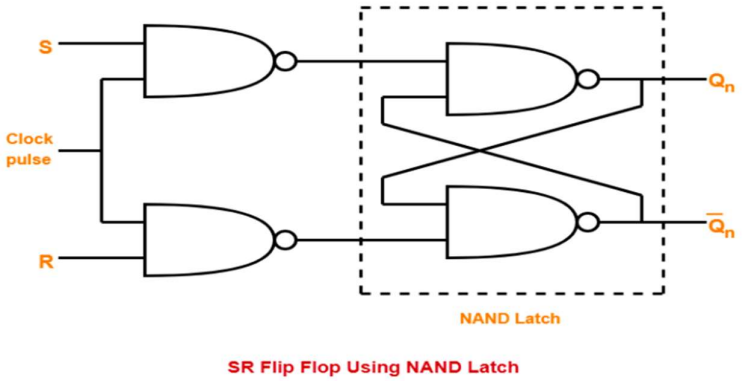


Figure 3: S-R Flip flop using NAND latch

CLOCK EDGE	S	R	Q	Q'	State
Positive/Negative	0	0	Q	Q'	No change
Positive/Negative	0	1	0	1	Reset State
Positive/Negative	1	0	1	0	Set State
Positive/Negative	1	1	X	X	Invalid

Table 2: S-R flip flop truth table

1) Generate the Boolean expression for the S-R flipflop using K-map and truth table given in Table 2.

2) Write a verilog module for NAND gate and utlize it to develop a structural verilog module for S-R flip flop as per figure 3.

3) Validate it using suitable Test bench.

Q 4. We can also develop a behavioral modeling based verilog code for the S-R latch. Here we can use the if-else logic to assign values for output based on the input conditions.

For example, the condition S = 1 or H, R = 0 or L then Q = 1 and Q' = 0 can be expressed under a begin block using if condition as:

```
If (S == 1 & R == 0)
begin
        Q <= 1;
        Q_bar <= 0;
end
```

Similar code can be developed for rest of the conditions.

1) Develop a behavioral verilog code using if-else slatemets for S-R latch.

2) Varify it with a suitable testbench code.

Q 5. Develop a similar behavioral code and test bench for S-R flip flop using if-else condition as per question 3.

Submission Instructions:

- Prepare the submission file according to the following process:
    1. Copy the Verilog code, the Test Bench Code in a Word File.
    2. Take the ScreenShot of Waveform and paste into the same word file.
    3. Repeat Step 1 and 2 for all the programs.
    4. Copy and Paste all the Verilog code, Testbench Code and Waveform into a single word file as 1_verilog, 1_TestBench, 1_Waveform, 2_verilog, 2_TestBench, 2_Waveform… etc.

Convert it into pdf file, print it and prepare a file for the verification in the next lab.

Department of Computer Science and Engineering
Pandit Deendayal Energy University
Digital Electronics and Computer Organization Lab – **20CP203P**

Lab Assignment – 9: Flip flops (JK, D, T)

Q 1. Write a Verilog code to implement J-K flip flop and validate the code via a suitable Test bench code.

Q 2. Design a J-K flip flop in Logisim and validate the circuit.

Q 3. Write a Verilog code to implement D flip flop and validate the code via a suitable Test bench code.

Q 4. Design a D flip flop in Logisim and validate the circuit.

Q 5. Write a Verilog code to implement T flip flop and validate the code via a suitable Test bench code.

Q 6. Design a T flip flop in Logisim and validate the circuit.

Submission Instructions:

- Prepare the submission file according to the following process:
    1. Copy the Verilog code, the Test Bench Code in a Word File.
    2. Take the ScreenShot of Waveform and paste into the same word file.
    3. Repeat Step 1 and 2 for all the programs.
    4. Copy and Paste all the Verilog code, Testbench Code and Waveform into a single word file as 1_verilog, 1_TestBench, 1_Waveform, 2_verilog, 2_TestBench, 2_Waveform… etc.

Convert it into pdf file, print it and prepare a file for the verification in the next lab.

Department of Computer Science and Engineering
Pandit Deendayal Energy University
Digital Electronics and Computer Organization Lab – **20CP203P**

Lab Assignment – 10 : Flip flops Conversion

In the previous lab we discussed about Sequential circuits. We developed Verilog modules for S-R latch and S-R flip-flops. In today's assignment, we will learn about the conversion of one flip-flop into other.

Please use the table 1, describing the input and output states of various flip-flops for your reference.

Present State	Next State	SR flip-flop inputs		D flip-flop input	JK flip-flop inputs		T flip-flop input
Q t	Q $t+1$	S	R	D	J	K	T
0	0	0	x	0	0	x	0
0	1	1	0	1	1	x	1
1	0	0	1	0	x	1	1
1	1	x	0	1	x	0	0

Table 1: Input and output states of various flip-flops

Q1> The **JK flip flop** is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level “1”.

1) Use the table 1 to develop the conversion table for S-R to J-K flip flop and generate the conversion expression between JK and SR via K map.

2) Use this conversion expression to Modify the S-R flip flop created in last lab into JK flip flop.

3) Verify the functionality of J-K flip flop using suitable Testbench.

Q2> Now lets convert the J-K flip flop created in question 1 into D flip flip. One can use the table 1 to perform similar steps as per previous solution for this conversion.

1) Develop the behavioral verilog module of D flip-flop, converting it from a J-K flip flop. You may utilize **if-else statements** to develop your verilog code.

2) Verify the functionality via a suitable test bench code.

Q3> Develop a characteristic table of T flip-flop along with the excitation inputs of JK flip flop from table 1.

1) Use K map to develop the conversion relation between T and JK flip flop.

2) Develop a behavioral verilog module for JK flip flop using **case** statements. Modify it to act like a T flip flop.

3) Validate the modification via a suitable test bench.

Submission Instructions:

- Prepare the submission file according to the following process:
    1. Copy the Verilog code, the Test Bench Code in a Word File.
    2. Take the ScreenShot of Waveform and paste into the same word file.
    3. Repeat Step 1 and 2 for all the programs.
    4. Copy and Paste all the Verilog code, Testbench Code and Waveform into a single word file as 1_verilog, 1_TestBench, 1_Waveform, 2_verilog, 2_TestBench, 2_Waveform… etc.

Convert it into a pdf file, print it and prepare a file for the verification in the next lab.