EXPERIMENT - 1

Aim: Design and Simulation of boolean functions using Verilog HDL.

Hardware implementation of a Boolean function in sum of products and product of sums expressions using universal gates.

Component/Software Used:

Component/Software	Specification
ICs	7400, 7402
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	As per requirement
Software(s) Used	Vivado 2016.1

Theory:

A binary variable may appear either in its normal form (A) or in its complement form ($\bar{A}$). Binary variables combined with an AND operation is called **minterms**, or a standard product. Similarly binary variables forming an OR terms called **maxterms**, or standard sums. For **"n"** variables with each variable being primed or unprimed, provide **"2^n"** possible minterms and maxterms.

The eight (2^3) different minterms and maxterms are shown in Table 1.1 for three (3) variables.

A Boolean function can be expressed algebraically from a given truth table by forming a **minterm** for each combination of the variables that produces a **"1**" in the function and then taking the **OR** of all those terms. From a given truth table, a Boolean function can alternatively be written algebraically by constructing a **maxterm** for each combination of variables that yields a "**0**" in the function and then taking the **AND** of all those terms. Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.The two canonical forms are basic forms of expressing Boolean function which is obtained from reading a given function from the truth table.

			Minterms		Maxterms	
A	B	C	Term	Designation	Term	Designation
0	0	0	$\bar{A}\bar{B}\bar{C}$	0	$A + B + C$	0
0	0	1	$\bar{A}\bar{B}C$	1	$A + B + \bar{C}$	1
0	1	0	$\bar{A}B\bar{C}$	2	$A + \bar{B} + C$	2
0	1	1	$\bar{A}BC$	3	$A + \bar{B} + \bar{C}$	3
1	0	0	$A\bar{B}\bar{C}$	4	$\bar{A} + B + C$	4
1	0	1	$A\bar{B}C$	5	$\bar{A} + B + \bar{C}$	5
1	1	0	$AB\bar{C}$	6	$\bar{A} + \bar{B} + C$	6
1	1	1	ABC	7	$\bar{A} + \bar{B} + \bar{C}$	7

Table 1.1: Minterms and Maxterms for 3 binary variables

Another way to express Boolean functions is in standard form. In digital logic, the "sum of products" (SOP) and "product of sums" (POS) are two distinct standard forms to describe a Boolean expression. These representations are used to simplify and analyze logic circuits.

Sum of Products (SOP): SOP is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms.

Product of Sums (POS): POS is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms.

Both SOP and POS representations can be useful in different scenarios, depending on the complexity of the logic circuit and the requirements of the design. In some cases, SOP might be more suitable, while in others, POS might provide a simpler and more concise expression. The choice of representation can affect the number of logic gates required and the overall complexity of the circuit.

Consider a Boolean function $(, ,) = \bar{} + \bar{} +$ for which the minterms and maxterms can be derived from the truth table shown in Table 1.2. This function can be implemented with NAND gates as shown in Figure 1.1. by considering the SOP expression. Also the given function can be implemented with NOR gates as shown in Figure 1.2. by considering the POS expression.

NAND or NOR gates are called Universal gates. Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates. Universal gates are easier to fabricate with electronic components.

=

Inputs			Output
A	**B**	**C**	**F**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

SOP: $F(A, B, C) = m_1 + m_3 + m_6 + m_7$

Table 1.2: Truth table of the Boolean function

$(,,) = \bar{} + \bar{} +$

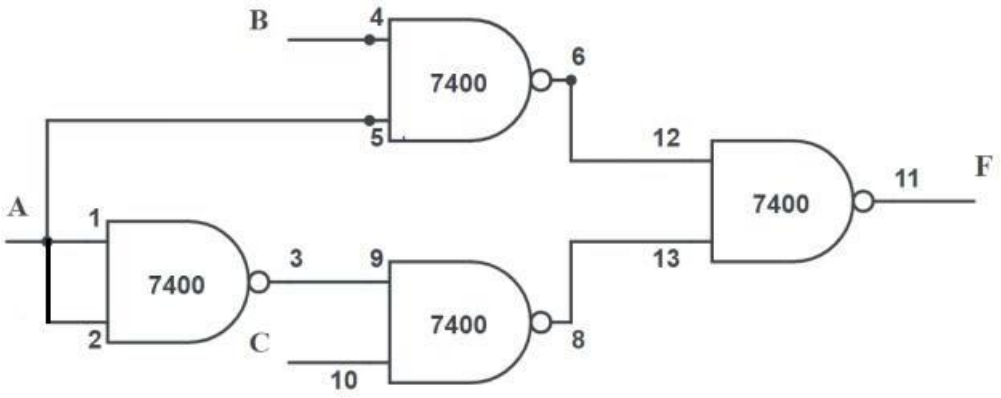


Figure 1.1: SOP implementation of Boolean function using NAND gates.

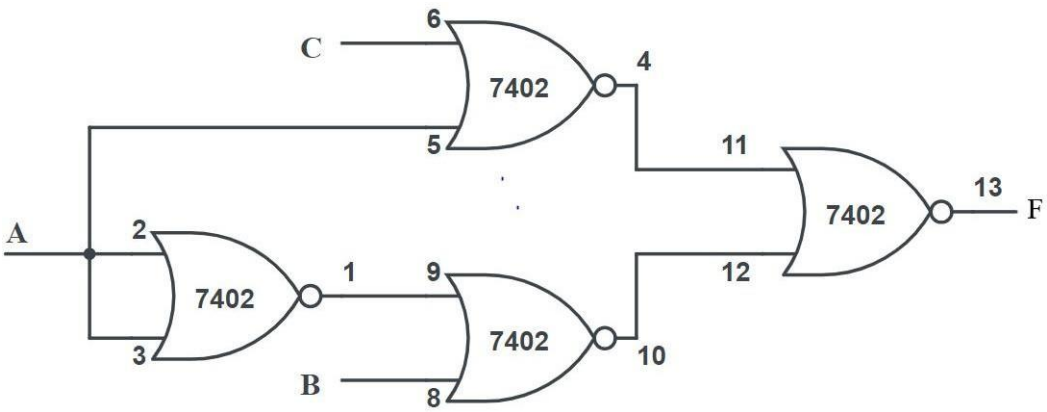


Figure 1.2: POS implementation of Boolean function using NOR gates.

Observation:

To be written by students

Design Problem:

Design a warning indicator using universal gates when the following conditions apply:

I. Switches A,B,C are on.

II. Switches A and B are on but switch C is off.

III. Switches A and C are on but switch B is off.

IV. Switches B and C are on but switch A is off.

Solution:

First derive the truth table for the given design as shown in the Table 1.3. Then obtain the minimized Boolean expression in SOP and POS form. After getting the Boolean expression draw the logic diagram using NAND and NOR gates.

Inputs			Output
A	**B**	**C**	**F**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$= \sum (3,5,6,7) = AB + BC + AC$

$= \overline{\overline{AB}\,\overline{BC}\,\overline{AC}}$)

( )

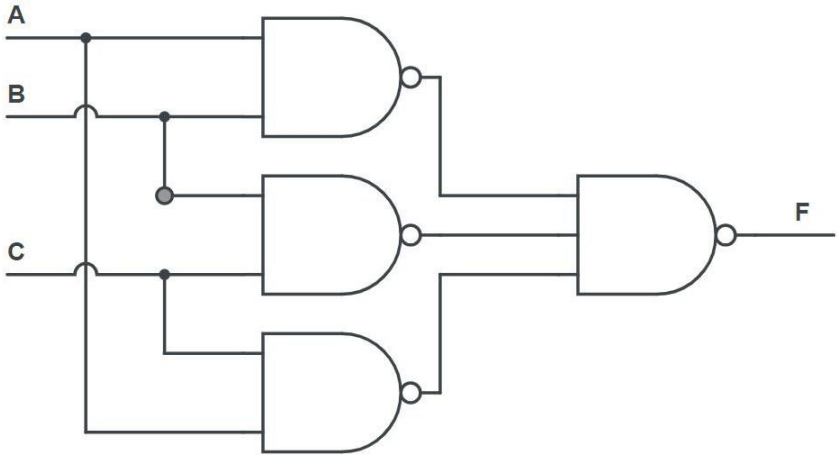


Figure 1.3: Logic diagram of the warning indicator using NAND gates.

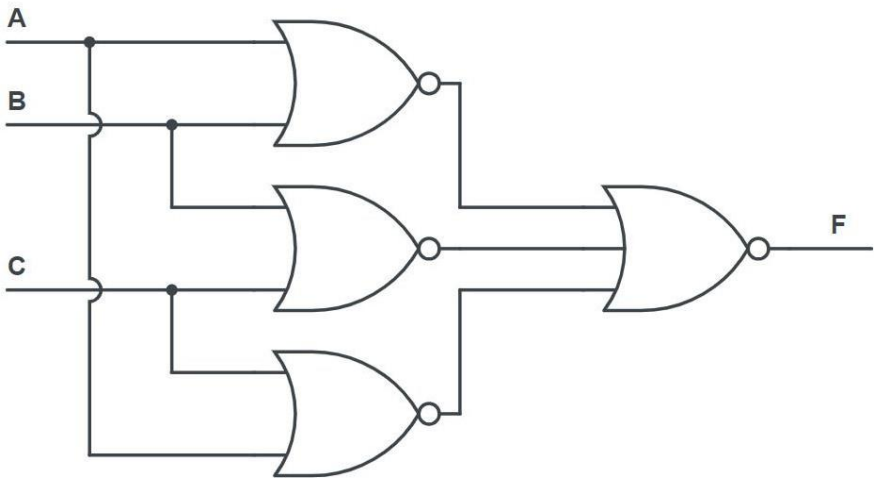


Figure 1.4: Logic diagram of the warning indicator using NOR gates.

EXPERIMENT - 2

Design and Simulation of Full Adder circuit using Verilog HDL.

Hardware implementation of Full Adder circuit using logic gates.

Aim:

Component/Software Used:

Component/Software	Specification
ICs	7408, 7432, 7486
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	As per requirement
Software(s) Used	Vivado 2016.1

Theory:

Full adder:

A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. Two of the input variables, denoted by **A** and **B**, represent the two significant bits to be added. The third input $\mathbf{C_{in}}$ represents the carry from the previous lower significant position. Two outputs are required because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits. The two outputs are designated by **Sum** and $\mathbf{C_{out}}$ for output carry.

When all input bits are 0, the output is 0. The Sum output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The $\mathbf{C_{out}}$ output has a carry of 1 if two or three inputs are equal to 1.

The truth table of the full adder is listed in Table 2.1. The eight rows under the input variables designate all possible combinations of the three variables. The Block diagram of the full adder is shown in the Figure 2.1. Then find out the Boolean expression for **Sum** and $\mathbf{C_{out}}$ using Boolean Algebra. In Figure 2.2 the Logic diagram of the full adder is shown.

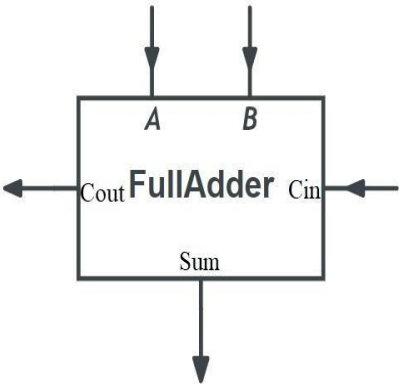


Figure 2.1: Block diagram of the full adder

Inputs			Outputs	
A	**B**	**C_{in}**	**C_{out}**	**Sum**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 2.1: Truth table of the full adder

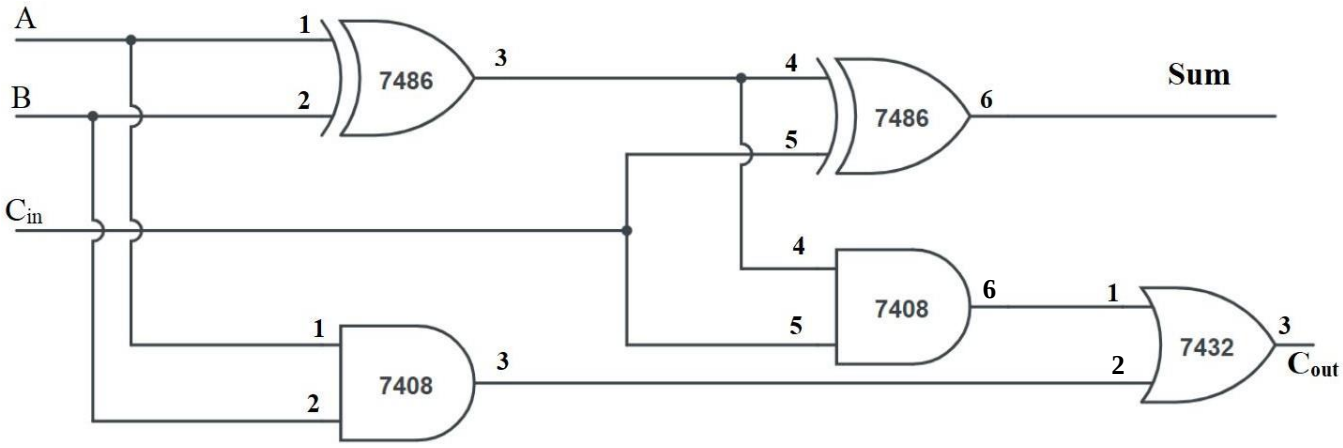


Figure 2.2: Logic diagram of the full adder

Design Problem:

Design and Simulation of Full-Subtractor circuit using Verilog HDL.
Hardware implementation of Full-Subtractor circuit.

Solution:

Full subtractor is a combinational circuit that performs a subtraction of two bits, taking into account borrowing form the lower significant stage. This circuit has three inputs named minuend (X), subtrahend (Y), and previous borrow (Z) and two outputs, called difference (D) and a borrow (B).

Then find out the boolean expression for difference (D) and a borrow (B) using Boolean Algebra. In Figure 2.3 the Logic diagram of the full subtractor is shown.

Inputs			Outputs	
X	**Y**	**Z**	**D**	**B**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 2.2: Truth table of the full subtractor

$$D(X,Y,Z) = \sum m(1,2,4,7) = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$$

$$= (X \oplus Y) \oplus Z$$

$$B(X,Y,Z) = \sum m(1,2,3,7) = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + \bar{X}YZ + XYZ$$

$$= \bar{X}Y(\bar{Z} + Z) + Z(\bar{X}\bar{Y} + XY)$$

$$= \bar{X}Y + Z(\overline{X \oplus Y})$$

Boolean expressions Difference and Borrow of Full Subtractor

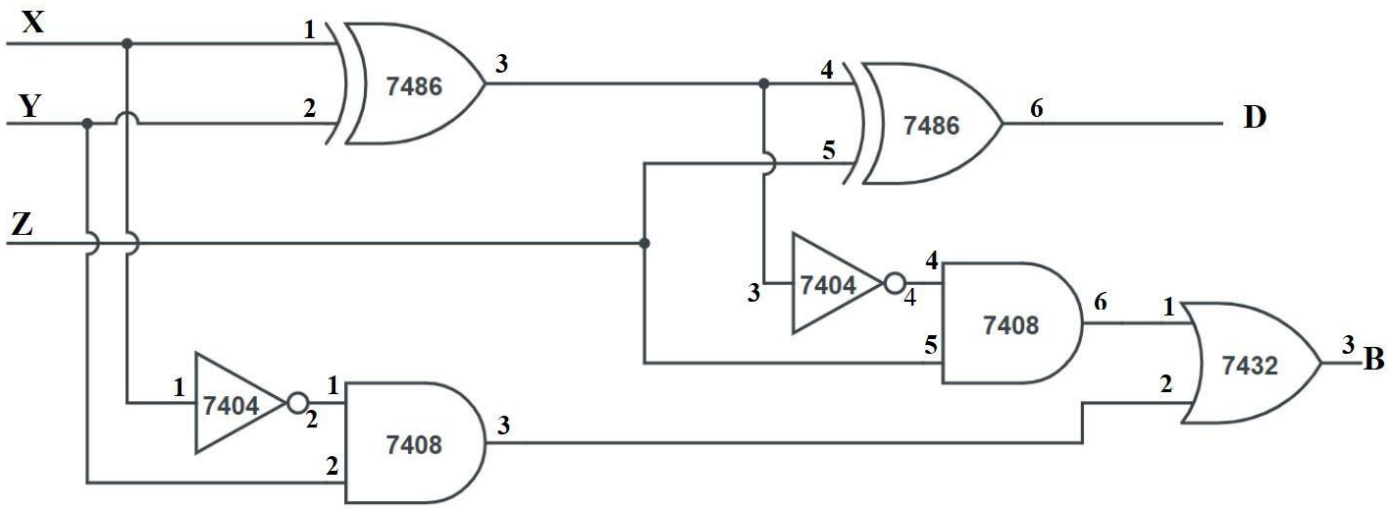


Figure 2.3 : Logic diagram of the Full Subtractor

EXPERIMENT - 3

Design and Simulation of 3 line to 8 line active high decoder using Verilog HDL.

Realization of 3 variable Boolean function using active low decoder.

Aim:

Component/Software Used:

Component/Software	Specification
ICs	7413,7408
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	As per requirement
Software(s) Used	Vivado 2016.1

Theory:

Decoder:

Decoder is a combinational circuit that has 'N' input lines and maximum of $(2)^N$ unique output lines. The name "Decoder" means to translate or decode coded information from one format into another, so a digital decoder transforms a set of digital input signals into an equivalent decimal code at its output. Active High Decoder - The output line that is active will be HIGH(1) and rest all the outputs will be LOW(0). Active Low Decoder - The output line that is active will be LOW(0) and rest all the outputs will be HIGH(1)

This 3-to-8 line binary decoder consists of an array of eight AND gates. The three(3) binary inputs labeled I_0 , I_1 and I_2 are decoded into one of 8 outputs, hence the description of 3 -to - 8 binary decoder. Each output represents one of the minterms of the 3 input variables, (each output is equals to a minterm).

The binary inputs I_0 , I_1 and I_2 determine which output line from D_0 to D_7 is "HIGH" at logic level "1" while the remaining outputs are held "LOW" at logic "0" so only one output can be active (HIGH) at any one time. Therefore, whichever output line is "HIGH" identifies the binary code present at the input, in other words it "de-codes" the binary input. Truth table of 3:8 active high Decoder is shown in Table 3.1 . In case of active low decoder output line from D_0 to D_7 is "LOW" while the remaining outputs are held "HIGH" so only one output can be active (LOW) at any one time as shown in Table 3.2.

Inputs			Outputs							
I_2	I_1	I_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
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

Table 3.1: Truth table of 3:8 active high Decoder

$$D_0(I_2, I_1, I_0) = m_0 = \overline{I_2}\,\overline{I_1}\,\overline{I_0}$$
$$D_1(I_2, I_1, I_0) = m_1 = \overline{I_2}\,\overline{I_1}\,I_0$$
$$D_2(I_2, I_1, I_0) = m_2 = \overline{I_2}\,I_1\,\overline{I_0}$$
$$D_3(I_2, I_1, I_0) = m_3 = \overline{I_2}\,I_1 I_0$$
$$D_4(I_2, I_1, I_0) = m_4 = I_2\,\overline{I_1}\,\overline{I_0}$$
$$D_5(I_2, I_1, I_0) = m_5 = I_2\,\overline{I_1}\,I_0$$
$$D_6(I_2, I_1, I_0) = m_6 = I_2 I_1\,\overline{I_0}$$
$$D_7(I_2, I_1, I_0) = m_7 = I_2 I_1 I_0$$

Boolean Expression of Outputs of 3:8 Decoder

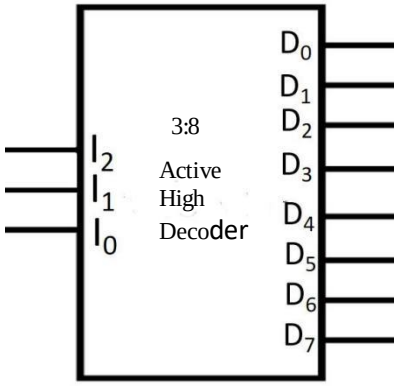


Figure 3.1: Block diagram of 3:8 Decoder

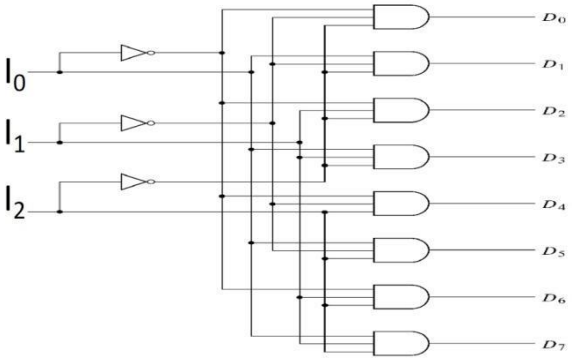


Figure 3.2: Logic diagram of 3:8 Decoder

Inputs			Outputs							
I_2	I_1	I_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

Table 3.2: Truth table of 3:8 active low Decoder

$$D_0(I_2, I_1, I_0) = \overline{m_0} = \overline{\overline{I_2}\,\overline{I_1}\,\overline{I_0}}$$
$$D_1(I_2, I_1, I_0) = \overline{m_1} = \overline{\overline{I_2}\,\overline{I_1}\,I_0}$$
$$D_2(I_2, I_1, I_0) = \overline{m_2} = \overline{\overline{I_2}\,I_1\,\overline{I_0}}$$
$$D_3(I_2, I_1, I_0) = \overline{m_3} = \overline{\overline{I_2}\,I_1 I_0}$$
$$D_4(I_2, I_1, I_0) = \overline{m_4} = \overline{I_2\,\overline{I_1}\,\overline{I_0}}$$
$$D_5(I_2, I_1, I_0) = \overline{m_5} = \overline{I_2\,\overline{I_1}\,I_0}$$
$$D_6(I_2, I_1, I_0) = \overline{m_6} = \overline{I_2 I_1\,\overline{I_0}}$$
$$D_7(I_2, I_1, I_0) = \overline{m_7} = \overline{I_2 I_1 I_0}$$

Boolean Expression of Outputs of 3:8 active low Decoder

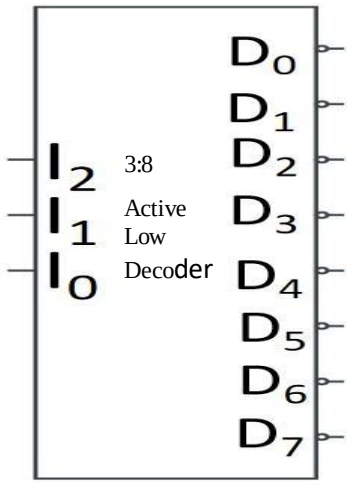


Figure 3.3: Block diagram of 3:8 Decoder

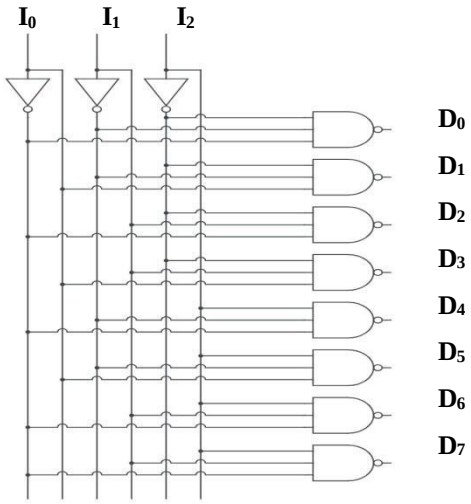


Figure 3.4: Logic diagram of 3:8 Decoder

The IC-74138 is 3:8 active low decoder its pin diagram is shown in Figure 3.5. The IC-74138 has 3 binary inputs A, B, and C which are equivalent to I_0 , I_1 and I_2 respectively. It has 8 outputs Y_0 to Y_7 those are equivalent to D_0 to D_7 respectively. If the device is enabled, these inputs determine which one of the eight normally HIGH outputs will go LOW. The IC has two active LOW and one active HIGH enables (G1, G2A and G2B) which will be helpful in cascading of decoders. The enable pins must be active for normal operation of decoder. The functioning of the IC74138 is illustrated in the Table 3.3, where "X" denotes the Don't care in the table.

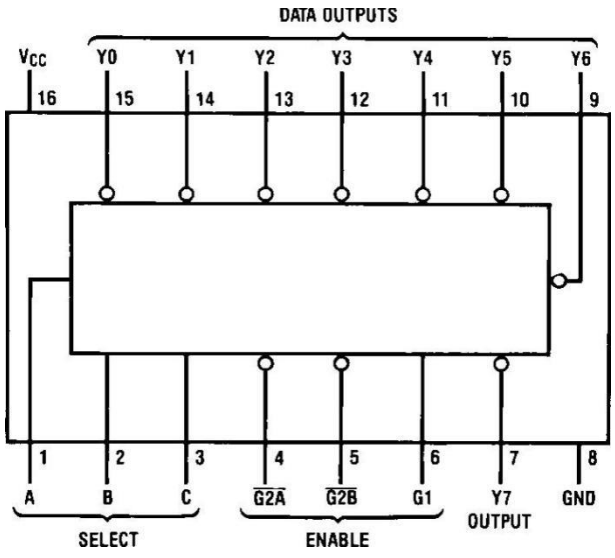


Figure 3.5: Pin diagram of IC 74138

Inputs						Outputs							
Enable Inputs			Select										
G2B	G2A	G1	C	B	A	0	1	2	3	4	5	6	7
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0
x	x	0	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
1	x	x	x	x	x	1	1	1	1	1	1	1	1

Table 3.3: Function table of IC 74138

Boolean function implementation:

Any Boolean function may be represented in sum-of-minterms or product-of-maxterms form, a hardware implementation of the function is provided by a decoder that creates the function's minterms and external gates that construct their logical sum.

The process for creating a Boolean function with a decoder and external gates requires that the Boolean function be defined as a sum-of-minterms or product-of-maxterms. Then, a decoder is chosen to create all the minterms or maxterms of the input variables. The inputs to each gate are chosen from the decoder outputs based on the list of minterms or maxterms for function. This approach is illustrated by implementing a Boolean function given below.

$$F(X, Y, Z) = \sum (0,3,5,6) = m_0 + m_3 + m_5 + m_6$$

$$= \prod (1,2,4,7) = \overline{m_1} \cdot \overline{m_2} \cdot \overline{m_4} \cdot \overline{m_7}$$

To implement the above Boolean function a 3:8 active low decoder is chosen and AND gated are required. The implementation of the Boolean function using the block diagram of the decoder IC 74138 is shown below in Figure 3.6

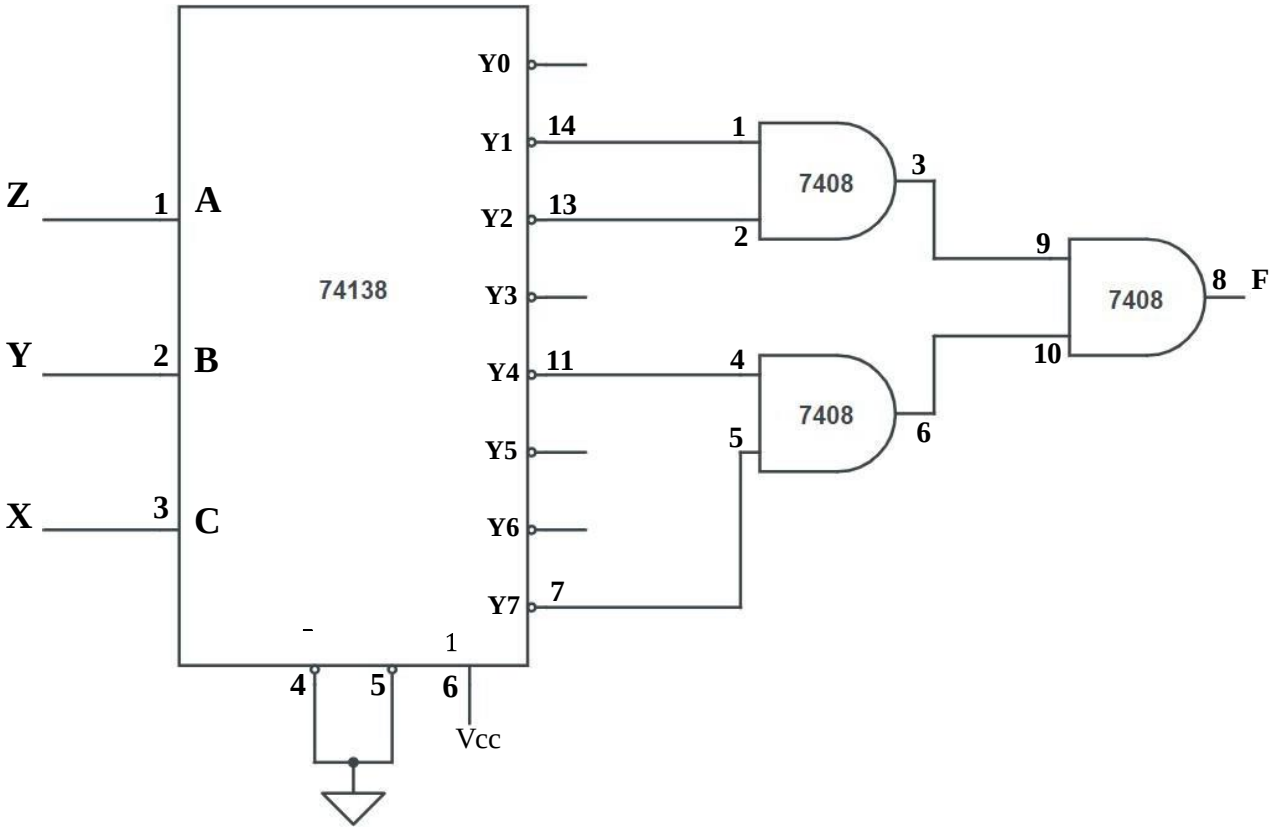


Figure 3.6: Block diagram of Boolean function using 3:8 Decoder

Observation:

To be written by students

Design Problem:

Design and Simulation of 3bit binary to Gray code converter using decoder.

Solution:

Binary to Gray code converter is a logical circuit that is used to convert the binary code into its equivalent Gray code. By putting the MSB of 1 below the axis and the MSB of 1 above the axis and reflecting the (n-1) bit code about an axis after 2^{n-1} rows, we can obtain the n-bit gray code.

In Table 3.4 the truth table of 3-bit binary to Gray code converter is shown. The implementation of the Gray code converter using the block diagram of the decoder IC 74138 is shown below in Figure 3.7.

Inputs			Outputs		
B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

$$_2(2, 1, 0) = \sum (4,5,6,7)$$

$$_1(2, 1, 0) = \sum (2,3,4,5)$$

$$_1(2, 1, 0) = \sum (1,2,5,6)$$

Table 3.4: Truth table of 3-bit binary to Gray code converter and its minterms

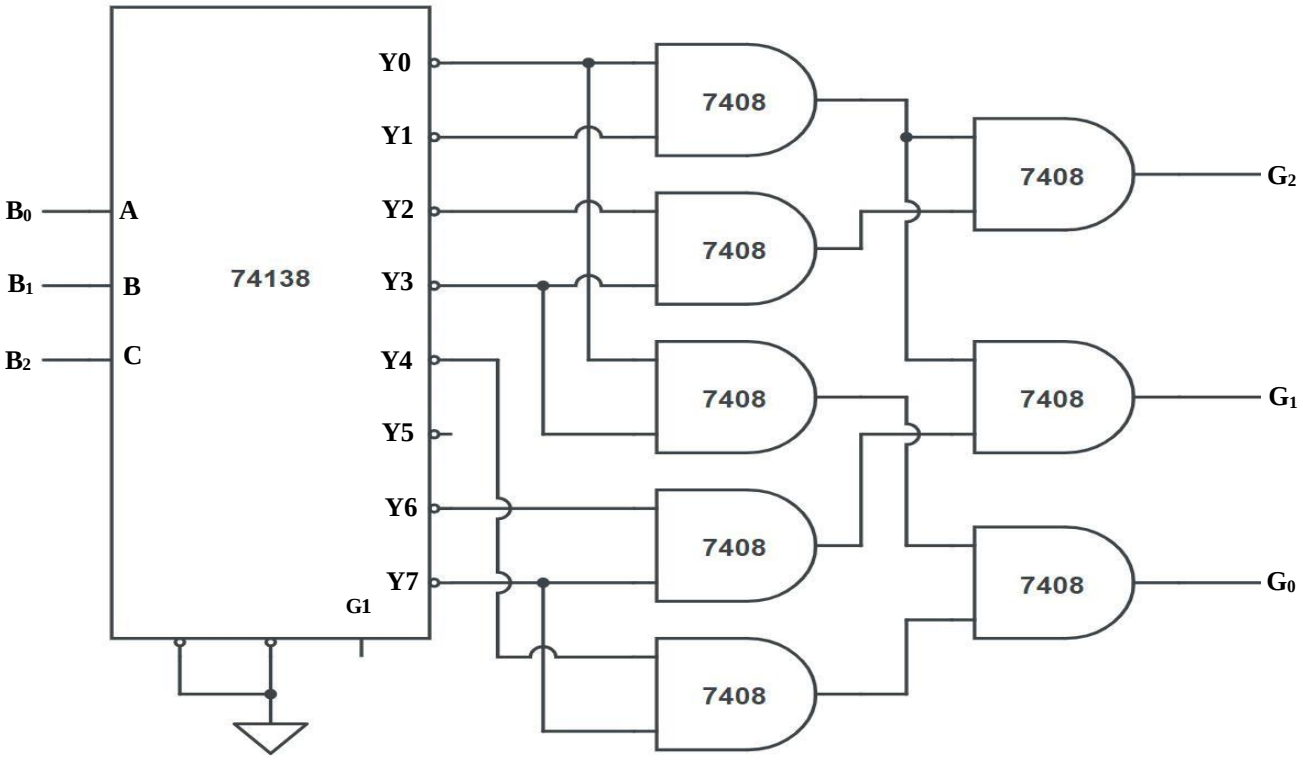


Figure 3.7: Block diagram of 3-bit binary to Gray code converter using decoder

Conclusion:

EXPERIMENT - 4

Design and Simulation of 8-to-1-line Multiplexer using Verilog HDL.
Generation of 4 variable logic function using 8-to-1-line Multiplexer.

Aim:

Component/Software Used:

Component/Software	Specification
ICs	74151, 7404
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	As per requirement
Software(s) Used	Vivado 2016.1

Theory:

Multiplexer:

Multiplexer (MUX): Multiplexer is a combinational circuit that has maximum of $(2)^n$ data inputs, '**n**'select input lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines. Since there are '**n**'selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. The multiplexer acts like a digitally controlled multi-position switch. A multiplexer is also called a **data selector** , since it selects one of many inputs and steers the binary information to the output line.

In 8-to-1-line **MUX (8X1 MUX)** there are eight**(8)** input lines and three**(3)** select lines whose bit combinations determine which input is selected. The block diagram and the function table is shown in Figure 4.1 and Table 4.1 respectively. In the Figure 4.1, $\mathbf{I_0}$ to $\mathbf{I_7}$ are eight inputs, $\mathbf{S_0}$ to $\mathbf{S_2}$ are the select input and **F** is the output of the **8X1 MUX**. The logic diagram is shown in Figure 4.2.

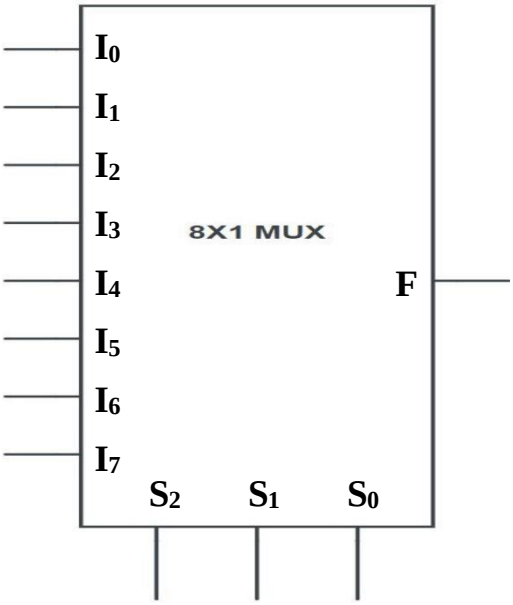


Figure 4.1: Block diagram of 8X1 MUX

Select Inputs			Output
S_2	S_1	S_0	F
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Table 4.1: Function table of 8X1 MUX

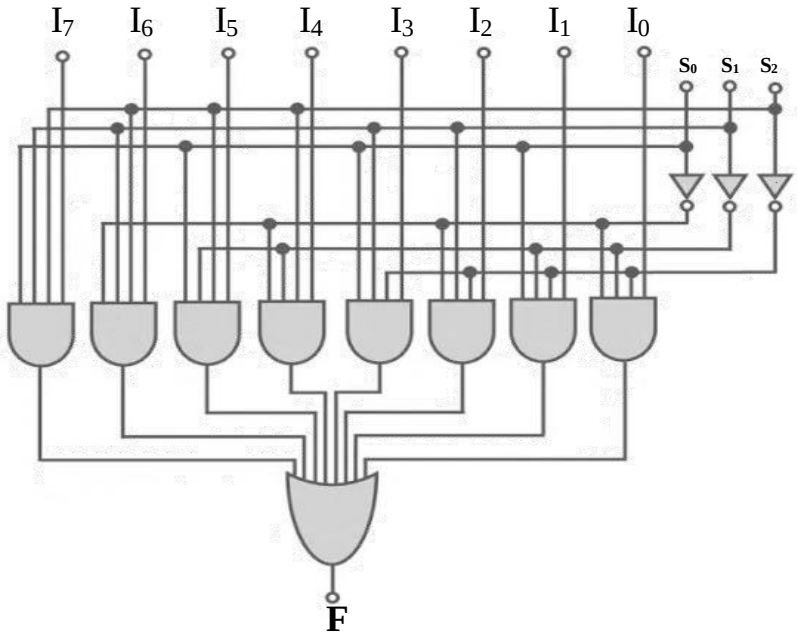


Figure 4.2: Logic diagram of 8X1 MUX

Digital multiplexer is available in IC 74151. IC 74151 is an 8-to-1 line (8X1 MUX) that has active high 8-data inputs and 3-selection input lines and two outputs, one active low and other active high output. The enable input ($\bar{E}$) is active low in the MUX, must be assigned logic 0 (LOW) for its normal operation.Figure 4.3 shows the pin diagram IC-74151.

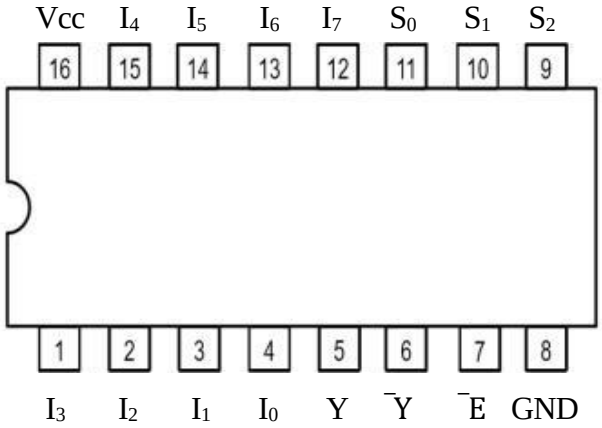


Figure 4.3: Pin diagram IC-74151

PIN NO	SYMBOL	FUNCTION
4, 3, 2, 1, 15, 14, 13, 12	I_0 to I_7	multiplexer inputs
5	Y	multiplexer output
6	Ɏ	complementary multiplexer output
7	¯E	enable input (active LOW)
8	GND	ground (0 V)
11, 10, 9	S0, S1, S2	select inputs
16	Vcc	positive supply voltage

Table 4.2: Pin Description of IC-74151

Inputs												Outputs	
Enable	Select			Data									
¯E	S_2	S_1	S_0	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Z	¯
0	0	0	0	0	x	x	x	x	x	x	x	0	1
0	0	0	0	1	x	x	x	x	x	x	x	1	0
0	0	0	1	x	0	x	x	x	x	x	x	0	1
0	0	0	1	x	1	x	x	x	x	x	x	1	0
0	0	1	0	x	x	0	x	x	x	x	x	0	1
0	0	1	0	x	x	1	x	x	x	x	x	1	0
0	0	1	1	x	x	x	0	x	x	x	x	0	1
0	0	1	1	x	x	x	1	x	x	x	x	1	0
0	1	0	0	x	x	x	x	0	x	x	x	0	1
0	1	0	0	x	x	x	x	1	x	x	x	1	0
0	1	0	1	x	x	x	x	x	0	x	x	0	1
0	1	0	1	x	x	x	x	x	1	x	x	1	0
0	1	1	0	x	x	x	x	x	x	0	x	0	1
0	1	1	0	x	x	x	x	x	x	1	x	1	0
0	1	1	1	x	x	x	x	x	x	x	0	0	1
0	1	1	1	x	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	x	x	x	x	0	1

Table 4.3: Function table of IC-74151

Boolean function implementation:

As the name indicates, MUX is many into one, which means that this circuit is associated with many inputs and a single output. Here, the minterms of a function are generated by the circuit associated with the selection inputs. MUX has n-selection inputs and 2^n- data inputs, one for each minterm.

For implementing a Boolean function of n-variables with a MUX that has (n - 1)selection inputs. The first (n-1) variables of the function are connected to the selection inputs and the remaining single variable are used for the data inputs. For implementing any Boolean function of n-variables with a MUX with (n-1) selection inputs and (2^{n-1}) data inputs, the following steps are used:

I. Firstly, the Boolean function is listed in a truth table.

II. The first (n-1) variables in the table are applied to the selection inputs of the MUX.

III. For each combination of selection variables, evaluate the output as a function of the last variable.This function can be 0, 1, the variable, or the complement of the variable.

IV. These values are then applied to the data inputs in the proper order.

Boolean function $F(A, B, C, D) = \sum (1, 3, 4, 11, 12, 13, 14, 15)$ implementation is shown in Figure 4.4 using IC 74151. The first variable A is connected to selection input S_2 so that A , B, and C correspond to selection inputs S_2, S_1, and S_0,respectively.

The values for the data inputs are determined from the truth table listed in the Table 4.4.

	I_0 –	I_1 –	I_2 –	I_3 –	I_4 $\bar{A}$	I_5 $\bar{A}$	I_6 –	I_7
–	0	2	4	6	8	10	12	14
D	1	3	5	7	9	11	13	15
8x1 MUX data input	D	D	–	0	0	D	1	1

Table 4.4: Truth table for implementing Boolean function with MUX

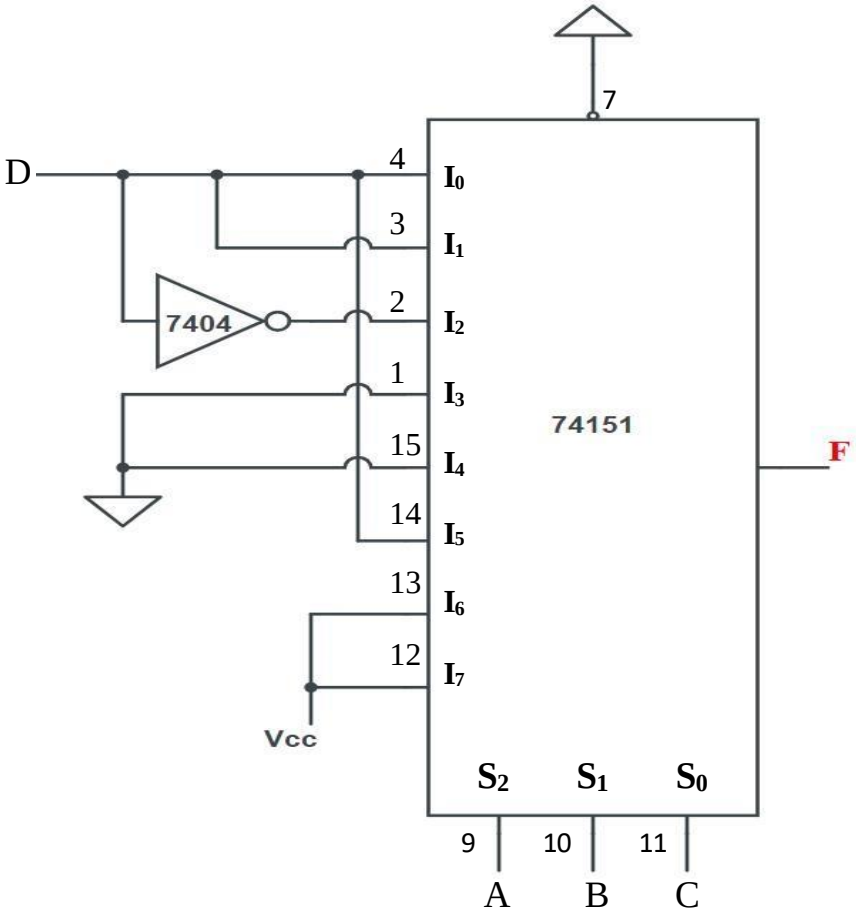


Figure 4.4: Implemention of Boolean function with a multiplexer

Design Problem :

Design and Simulation of 4X1 multiplexer by 2X1 multiplexers using Verilog HDL. Hardware implementation of 4X1 multiplexer using 2X1 multiplexers only.

Solution:

4x1 Multiplexer has four data inputs **I_3, I_2, I_1 & I_0**, two selection lines **S_1 & S_0** and one output F. The block diagram of 4x1 Multiplexer is shown in the Figure 4.5 below.

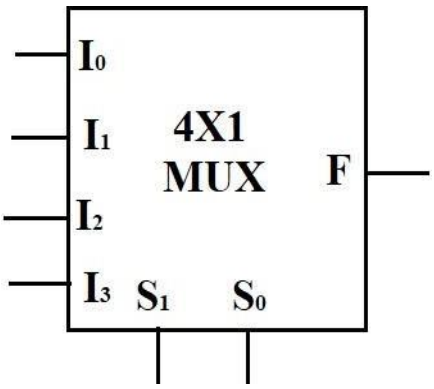


Figure 4.5 : Block diagram of 4 X 1 MUX

Select inputs		Output
S_1	S_0	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 4.5: Function table of 4 X 1 MUX

$$= \overline{(1} \qquad) + \overline{(1\,1)} + \overline{(1)} + (1)$$

Boolean Expression of 4X1 MUX

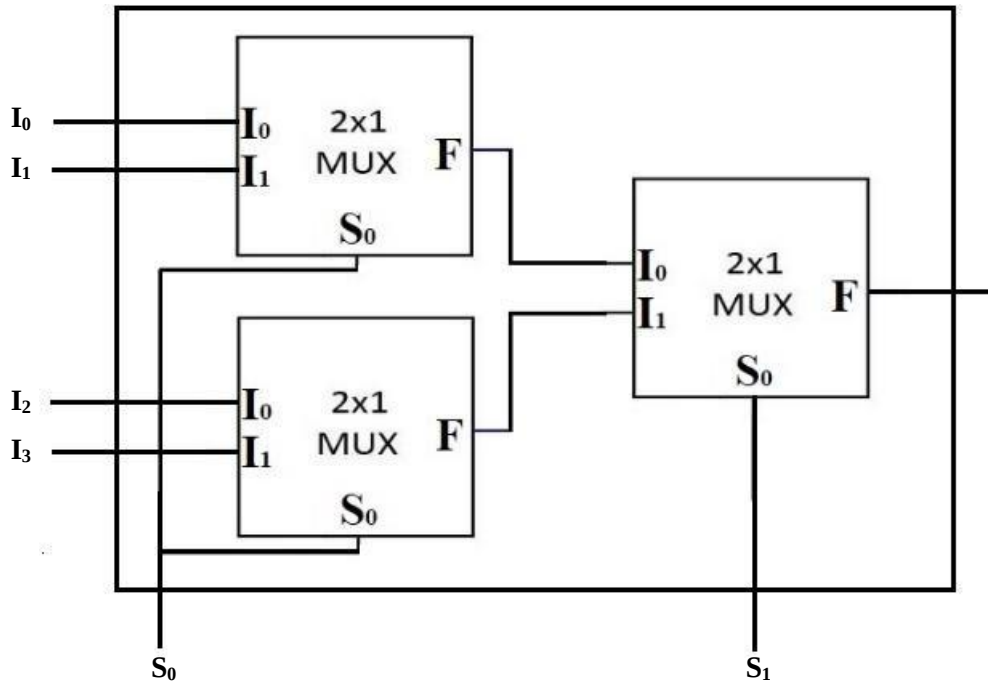


Figure 4.6: Logic diagram of 4 X 1 MUX using 2 X 1 MUXs

The hardware implementation of the 4 x1 MUX using 2x1 MUXs is done with the Quad 2x1 MUX IC 74157.

Quad 2x1 MUX IC 74157:

It has four similar multiplexers inside it, and hence it is called Quad Package 2-Input Multiplexer. Each has two input pins (for the first MUX: input 1A and 1B) and one output pin (for the first MUX: output 1Y), which forms a 2X1 Multiplexer. It selects four bits of data from two sources under the control of common data select input when the enable input is active low.

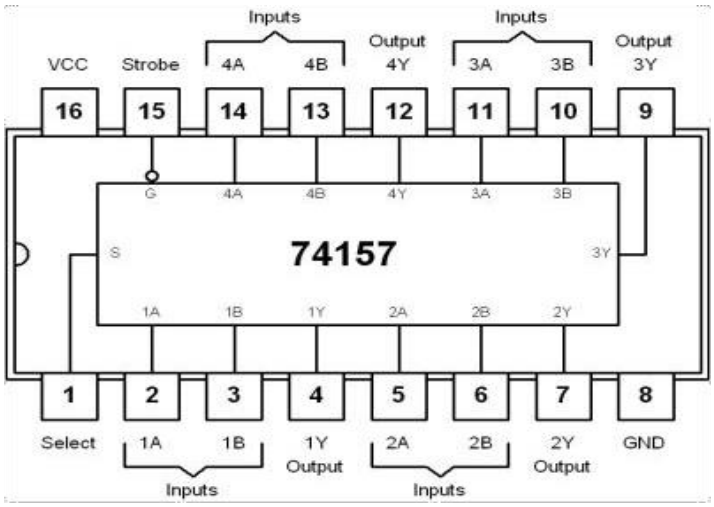


Figure 4.7: Pin diagram of quad 2X1 MUX IC 74157

EXPERIMENT - 5

Design and simulate JK flip-flop and D flip-flop using Verilog behavioral modeling. Designing of JK flip-flop using D flip-flop and 2X1 Multiplexer.

Aim:

Component/Software Used:

Component/Software	Specification
ICs	7476, 7474,74157,7404
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	As per requirement
Software(s) Used	Vivado 2016.1

Theory:

Flip-flop:

Flip - flop is the basic one bit digital memory circuit . It can store either 0 or 1.Flip-flops are the basic building blocks of most sequential circuits. Flip-flops are the fundamental components of shift registers and counters. A Flip-flop has two outputs, **Q** and $\bar{}$, which are always in opposite states. If Q is 1, then $\bar{Q}$is 0, and the flip-flop is said to be **set** or **on**. If Q is 0, then $\bar{Q}$is 1 and the flip-flop is said to be **reset**, **off**, or **cleared**. It can maintain a binary state indefinitely (as long as power is delivered to the circuit) until directed by an input signal to switch states. There are several types of flip flop, and the inputs vary with each type.

Flip-flop triggers only during a signal transition (from 0 to 1 or from 1 to 0), and is disabled during the rest of the clock cycle pulse duration. Positive edge means positive transition i.e. signal transition from 0 to 1. Negative edge means negative transition, i.e. signal transition from 1 to 0.

Flip-flops have **asynchronous** inputs that are used to force the flip-flop into a particular state independent of the clock. The input that sets the flip-flop to **1** is called a **Preset**. The input that clears the flip-flop to **0** is called **Clear**. When power is turned on in a digital system, the state of the flip-flop is unknown. The direct inputs are useful for bringing all the flip-flops in the system to a know starting state prior to the clocked operation.

JK flip-flop:

The J-K flip-flop is very versatile and also the most widely used. The outputs are complement to each other. A clock input is included in edge-triggered flip-flops.

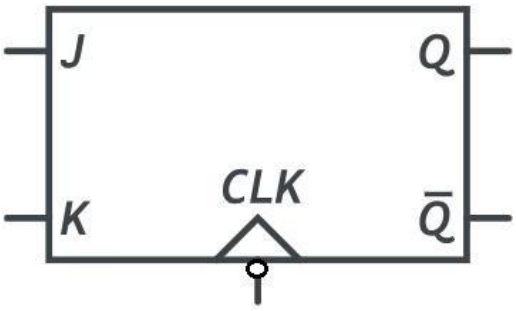


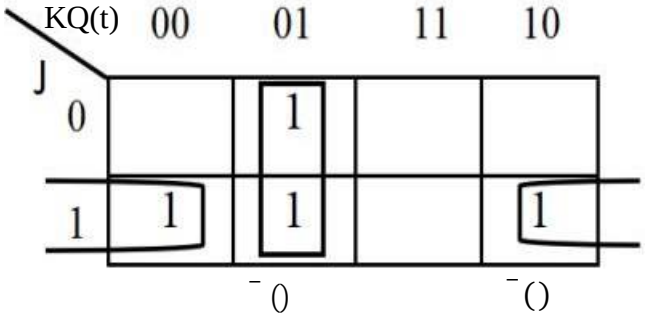
Figure 5.1 JK FF symbol(Negative edge)

Inputs			Output	Comments
CLK	J	K	Q	
1	X	X	Q(t)	**No change**
0	0	0	Q(t)	**No change**
0	0	1	1	**Set**
0	1	0	0	**Reset**
0	1	1	$\bar{Q}(t)$	**Toggle**

Table 5.1: JK Flip-Flop function table

Inputs		P.S	N.S
J	K	()	(+ 1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Table 5.2: Characteristic table of J-K FF



$$(+1) = \bar{}() + ()$$

Characteristic Equation of JK FF

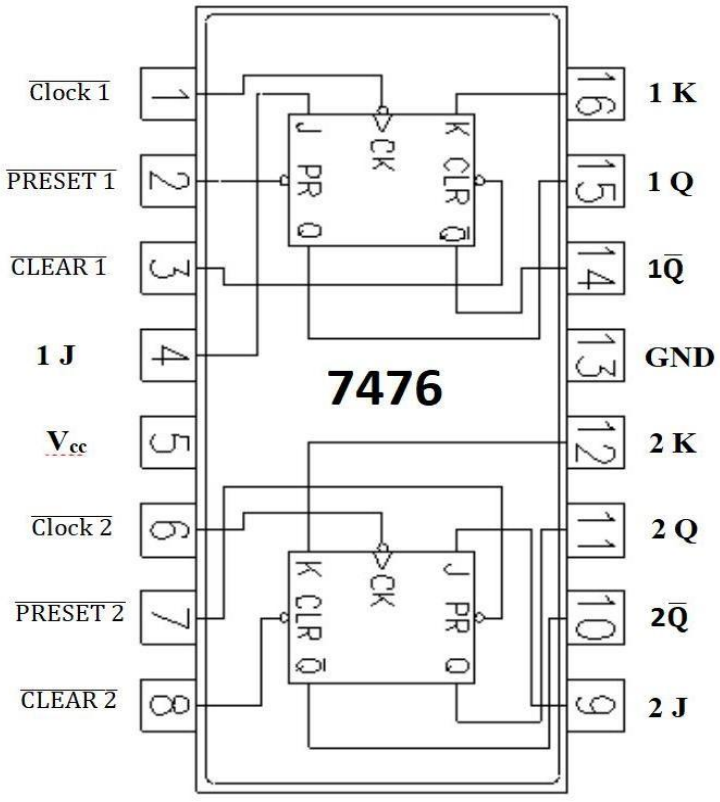


Figure 5.2: Pin diagram of IC 7476

Pin Number	Description
1	Clock 1 Input
2	Preset 1 Input
3	Clear 1 Input
4	J1 Input
5	Vcc - Positive Power Supply
6	Clock 2 Input
7	Preset 2 Input
8	Clear 2 Input
9	J2 Input
10	Complement Q2 Output
11	Q2 Output
12	K2 Input
13	Ground
14	Complement Q1 Output
15	Q1 Output
16	K1 Input

Table 5.3: Pin Description of 7476 IC

D flip-flop:

In D flip-flop, the output is same as input, i.e. when D= 0, the flip-flop is reset and when D = 1, the flip-flop is set.

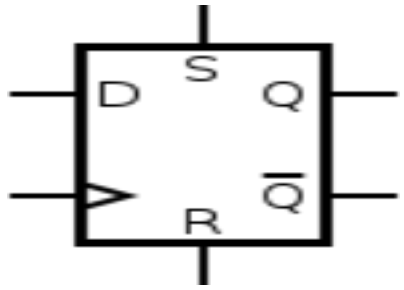


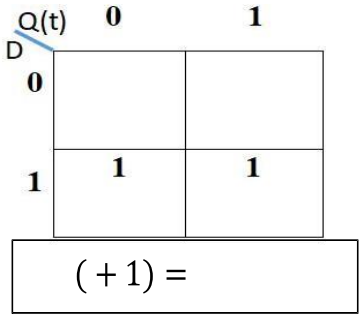
Figure 5.3 : D FF Symbol (Postive edge)

Inputs		Output	Comments
CLK	**D**	**Q**	
0	X	Q(t)	**No change**
1	0	0	**Reset**
1	1	1	**Set**

Table 5.4: D FF function table

P.S	Input	N.S
()	**D**	(+ 1)
0	0	0
0	1	1
1	0	0
1	1	1

Table 5.5: Characteristic table of D FF



Characteristic Equation of D FF

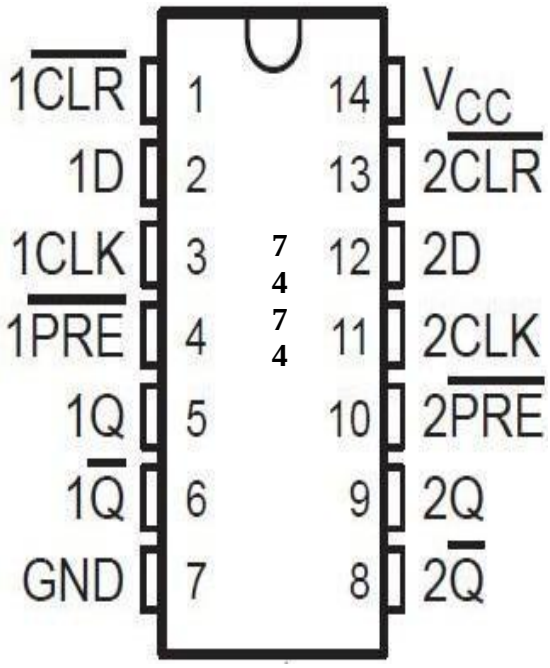


Figure 5.4 :Pin diagram of IC 7474

Pin Number	Description
1	Clear 1 Input
2	D1 Input
3	Clock 1 Input
4	Preset 1 Input
5	Q1 Output
6	Complement Q1 Output
7	Ground
8	Complement Q2 Output
9	Q2 Output
10	Preset 2 Input
11	Clock 2 Input
12	D2 Input
13	Clear 2 Input
14	Vcc - Positive Power Supply

Table 5.6: Pin Description of 7474 IC

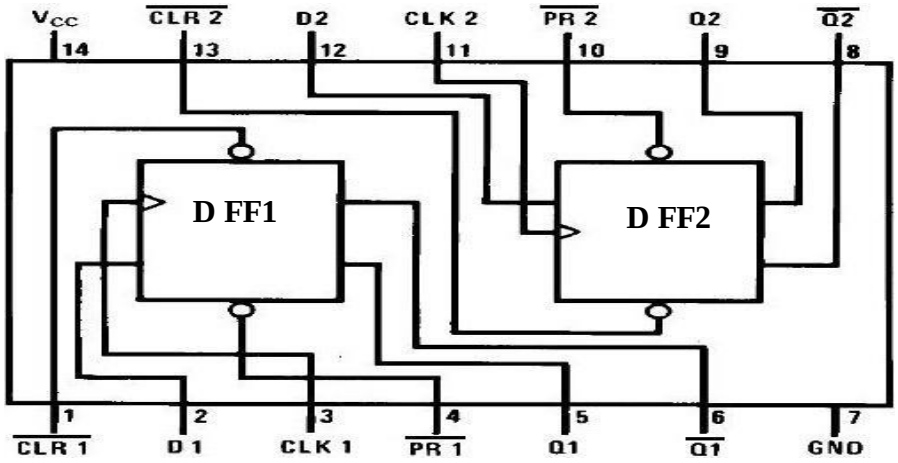


Figure 5.5: IC 7474 internal connection

Conversion of D flip-flop to JK flip-flop:

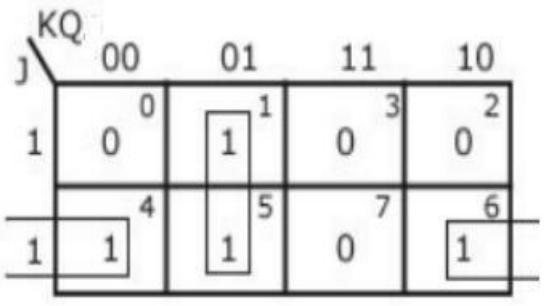
To convert one type of flip-flop into another type, a combination circuit is designed using function table, next state equation and the excitation table such that if the inputsof the required flip-flop are fed as inputs of the combinational circuit and the outputof the combinational circuit is connected to the inputs of the actual flip-flop, then the output of the actual flip-flop is the output of the required flip-flop.

P.S	N.S	Flip-flop inputs		
Q(t)	Q(t+1)	J	K	D
0	0	0	X	0
0	1	1	X	1
1	0	X	1	0
1	1	X	0	1

Table 5.7: Excitation table

Flip flop I/Ps		P.S	N.S	Required I/P's
J	K	Q(t)	Q(t+1)	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Table 5.8:Conversion table D flip-flop to JK FF



$$D(J,K,Q) = \sum m(1,4,5,6)$$

$$= JQ + \bar{K}Q$$

Boolean expressions for D

From the above conversion table and the k-map simplification of the Boolean expressions for D input is: $= \bar{} + \bar{}$. The D flip-flop can be converted into a JK flip-flop by giving the value of D in the D flip-flop. Figure 5.6 shows the logic diagram for D flip-flop converted to form the JK flip-flop using required logic gates to implement the combinational logic. Boolean function can be implemented using multiplexer thus the combinational logic is replaced by 2X1 Mux where select input is assigned the output of D flip-flop $S_0 =$ Q, and the inputs are assigned $I_0 =$ J , $I_1 = \bar{K}$

Figure 5.7 shows the logic diagram for D flip-flop converted to form the JK flip-flop using multiplexer and required logic gate.

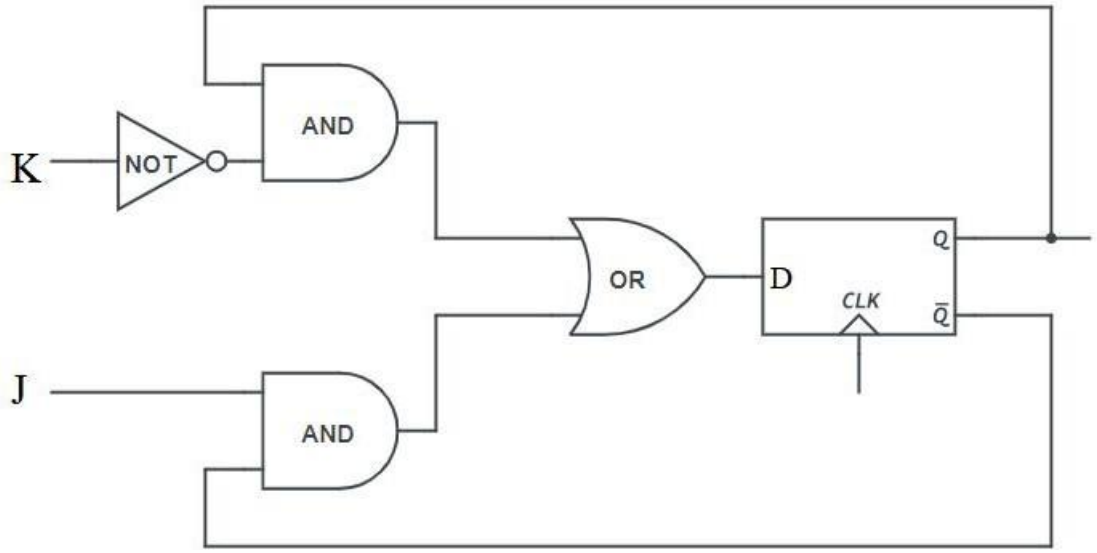


Figure 5.6: Logic diagram for JK FF using D FF and logic gates

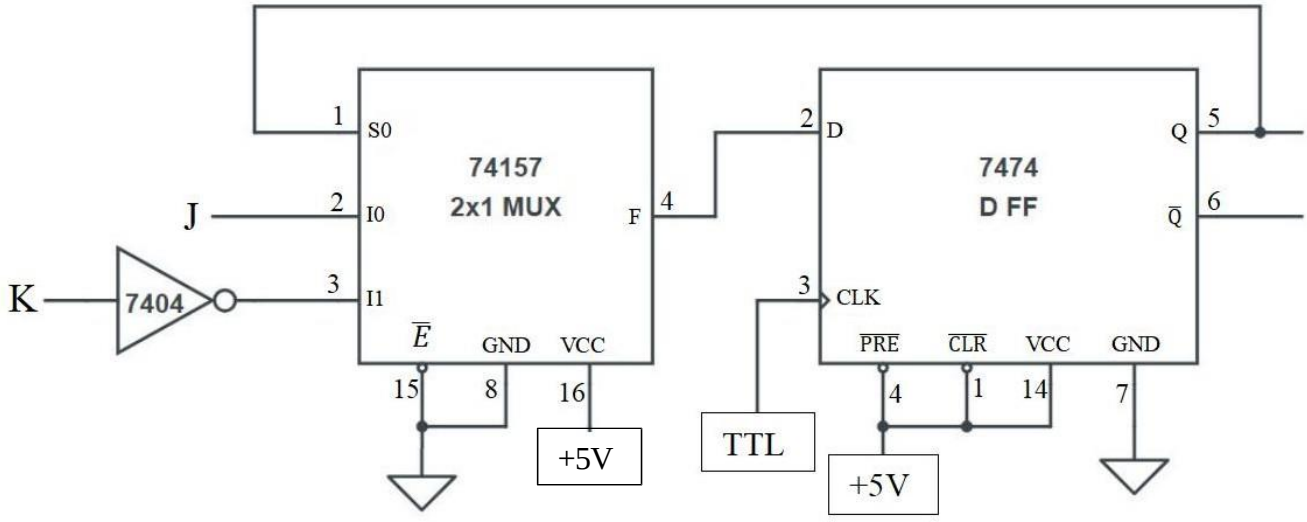


Figure 5.7: Logic diagram for JK FF using D FF (7474 IC) and 2X1 Mux (74157 IC)

Design Problem:

Design and Simulation of D Flip-Flop using JK Flip-Flop in Verilog HDL.
Hardware implementation of D Flip-Flop using JK Flip-Flop.

Solution:

Flip flop I/P	P.S	N.S	Required I/P's	
D	**Q(t)**	**Q(t+1)**	**J**	**K**
0	0	0	0	X
1	0	1	1	X
0	1	0	X	1
1	1	1	X	0

Table 5.9: Conversion table of JK flip-flop to D flip-flop

From the above conversion table, k-map simplification of the Boolean expressions for J and K inputs are: $= , = \bar{}$ .The JK flip-flop can be converted into a D flip- flop by giving the value of J and K in the JK flip-flop. Figure 5.8: shows the logic diagram for D flip-flop converted from JK flip-flop.

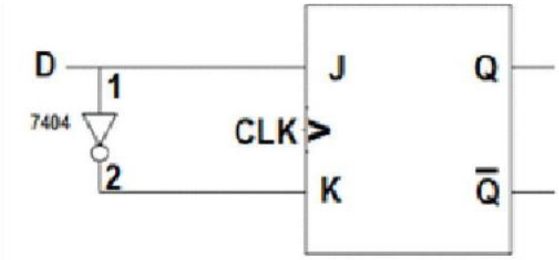


Figure 5.8: The logic diagram for D flip-flop converted from JK flip-flop.

EXPERIMENT - 6

Design and simulation of modulo counter using Verilog behavioral modeling.
Design modulo counter using JK flip-flops.

Aim

Component/Software Used:

Component/Software	Specification
ICs	7476, 7408, 7490, 7447
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	As per requirement
Software(s) Used	Vivado 2016.1

Theory:

Flip-flop:

A counter is one of the most beneficial and adaptable components of a digital system . The number of clock cycles can be counted using a counter that is powered by a clock. The function of a counter is to count the number of clock pulses. Since the clock pulses occur at known intervals, the counter can be used as an instrument for measuring time and, therefore, period or frequency. It can be used to keep track of the number of events that have occurred, such as the number of times a button has been pressed or the number of pulses received from a sensor. Flip-flops are the basic building blocks of a counter. There are basically two different types of counters synchronous and asynchronous.

An asynchronous binary counter is also called a serial or ripple counter. The ripple counter is simple and straightforward in operation and its construction requires a minimum amount of hardware. It does, however, have a speed limit. Each flip-flop is triggered by the previous flip-flop, and thus the counter has a cumulative settling time.

A synchronous binary counter is also called a parallel counter. An increase in the speed of operation can be achieved by use of a parallel or synchronous counter. Here, every flip-flop is triggered by the clock (synchronism), and thus the settling time is simply equal to the delay time of a single flip-flop. Usually, the additional hardware is required to get the speed improvement.

Counters can also be designed to count up or down, and the number of bits used to represent the count will determine the maximum number of counts that can be recorded. For example, a 4-bit counter can count from 0 to 15, while an 8-bit counter can count from 0 to 255.

A digital counter is designed using a set of flip-flops (FFs) whose states change in response to pulses applied at the input to the counter. A counter can be used as a frequency divider to obtain waveform with frequencies that are specific fractions of the clock frequency. They are also used to perform the timing function as in digital watches, to create time delays, to produce non-sequential binary counts, to generate pulse trains, and to act as frequency counters, etc.

The number of states or counting sequences through which a particular counter advances before returning once again back to its original first state is called the modulus (MOD). Modulus Counters, or simply MOD counters, are defined based on the number of states that the counter will sequence through before returning back to its original value. For example, a counter that counts from 00_2 to 11_2 in binary, that is 0 to 3 in decimal, has a modulus value of 4 ($00 \rightarrow 1 \rightarrow 10 \rightarrow 11$, and return back to 00) so would therefore be called a modulo-4, or mod-4, counter. Note also that it has taken four clock pulses to get from 00 to 11.

Design Procedure of a synchronous Counter:

1. Find the number of FFs (flip-flop) required.
The number of Flip-flops required can be determined by using the equation:
$M \leq 2^N$ where, M is the MOD number and N is the number of required flip-flops.
2. Draw a state diagram for a given sequence.
3. Draw the FF transition table (excitation table).
4. Use K-map to derive the logic expressions of inputs of FFs.
5. Implemention using FFs and required combinational logic(gates).

Design of Mod-6 Counter using JK flip-flops:

To design the Mod-6 synchronous counter, contain six counter states (i.e from 0 to 5). No of FFs required = 3 given by Q_A,Q_B and Q_C where Q_C is the MSB and Q_A is LSB to design this counter.

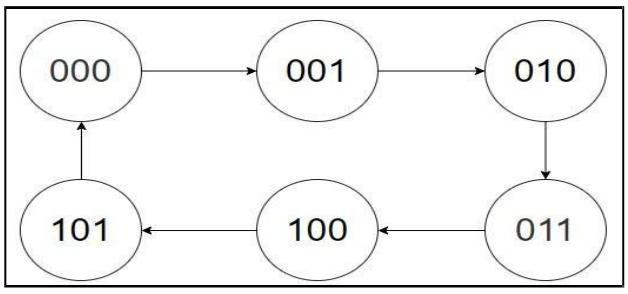


Figure 6.1: State Diagram of MOD-6 synchronous counter

Sl. No.	Clock pulse	Q_C	Q_B	Q_A
1	0	0	0	0
2	1	0	0	1
3	2	0	1	0
4	3	0	1	1
5	4	1	0	0
6	5	1	0	1

Table 6.1: Count table of MOD-6 counter

Present state			Next State			Inputs of the FFs					
Q_C	Q_B	Q_A	Q_C	Q_B	Q_A	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1

Table 6.2: Excitation table of MOD-6 counter

K-map and logic expressions of inputs of FFs.

Note: The counts "110" and "111" need to be consider as Don't care (X) in all K-maps.

Q_C \ $Q_B Q_A$	00	01	11	10
0	1	X	X	1
1	1	X	X	X

$$J_A(Q_C, Q_B, Q_A) = \sum(0,2,4) + d(1,3,5,6,7) = 1$$

Q_C \ $Q_B Q_A$	00	01	11	10
0	X	1	1	X
1	X	1	X	X

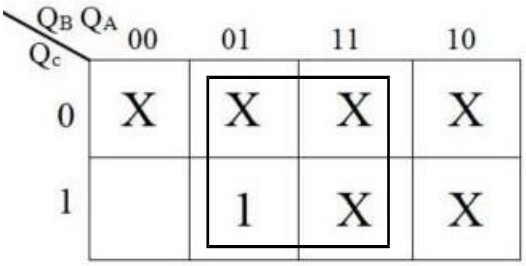
$$K_A(Q_C, Q_B, Q_A) = \sum(0,2,4) + d(1,3,5,6,7) = 1$$

Q_C \ $Q_B Q_A$	00	01	11	10
0		1	X	X
1			X	X

$$J_B(Q_C, Q_B, Q_A) = \sum(1) + d(2,3,6,7) = \bar{Q}_C Q_A$$

Q_C \ $Q_B Q_A$	00	01	11	10
0	X	X	1	
1	X	X	X	X

$$K_B(Q_C, Q_B, Q_A) = \sum(3) + d(0,1,4,5,6,7) = Q_A$$



$$J_C\,(Q_C, Q_B, Q_A) = \sum (3) + d(4,5,6,7)$$

$$= Q_B Q_A$$

$$K_C\;Q_C, Q_B, Q_A = \sum 5 + d(0,1,2,3,6,7)$$

$$= Q_A$$

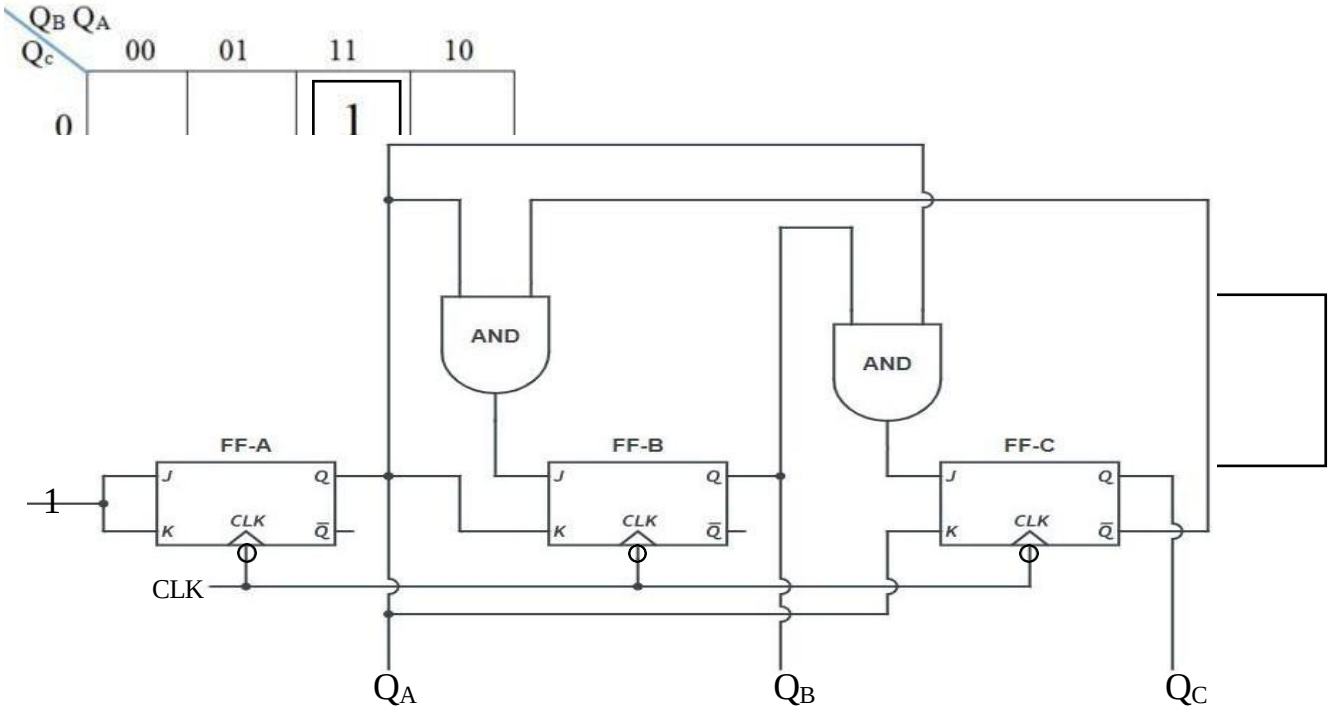


Figure 6.2 Logic Diagram of synchronous binary MOD-6 counter

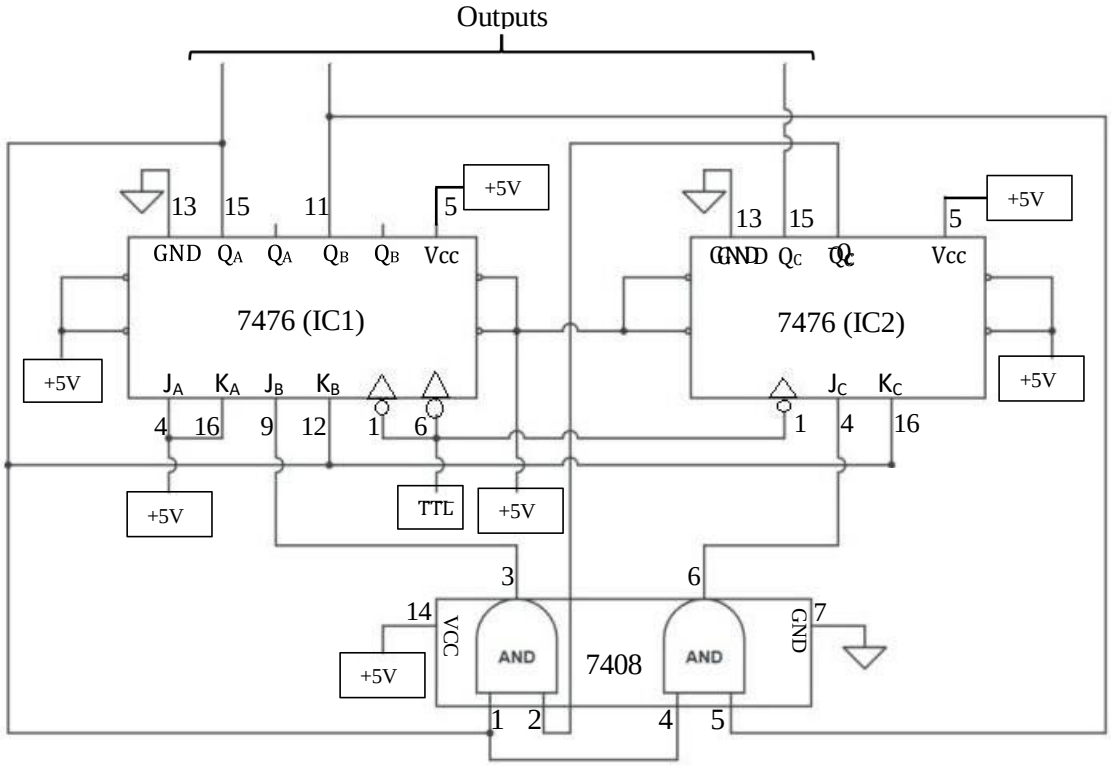


Figure 6.3: Logic Diagram of synchronous binary MOD-6 counter using IC 7476 and IC 7408

Design Problem

Design and Simulation of decade counter using Verilog HDL.
Hardware implementation of a decade counter to display the digits 0 to 9 using IC 7490 (decade counter), IC 7447 (BCD to decimal decoder) and seven segment display.

Solution:

IC 7490 : The IC 7490 is 4-bit ripple type Decade Counter. It consists of four master/slave flip-flops which are internally connected to provide a divide-by-two section and a divide-by-five section. Each section has a separate clock input which initiates state changes of the counter on the HIGH-to-LOW clock transition. State changes of the Q outputs do not occur simultaneously because of internal ripple delays. Therefore, decoded output signals are subject to decoding spikes and should not be used for clocks or strobes.

A gated NAND asynchronous Master Reset (MR_1, MR_2) is provided which overrides clocks and resets (clears) all the flip-flops. A gated NAND asynchronous Master Set (MS_1, MS_2) .

Since the output from the divide-by-two section is not internally connected to the succeeding stages, the devices may be operated in various counting modes.

BCD Decade (8421) Counter : The $\overline{CLK}_B$ input must be externally connected to the Q_A output. The $\overline{CLK}_A$ input receives the incoming count producing a BCD count sequence.

Symmetrical Biquinary Divide-By-Ten Counter: The Q_C output must be externally connected to the $\overline{CLK}_A$ input. The input count is then applied to the $\overline{CLK}_B$ input and a divide-by ten square wave is obtained at output Q0.

Divide-By-Two and Divide-By-Five Counter: No external interconnections are required. The first flip-flop is used as a binary element for the divide-by-two function ($\overline{CLK}_A$ as the input and Q_A as the output). The $\overline{CLK}_B$ input is used to obtain binary divide-by-five operation at the Q_C output.

The pin diagram and the logic symbol are given in Figure 6.4 and Figure 6.5 respectively. The count table is shown in Table 6.3. In Figure 6.5, Q_D (MSB) , Q_C,$Q_B$andQ_A (LSB) are the outputs of the Decade Counter.

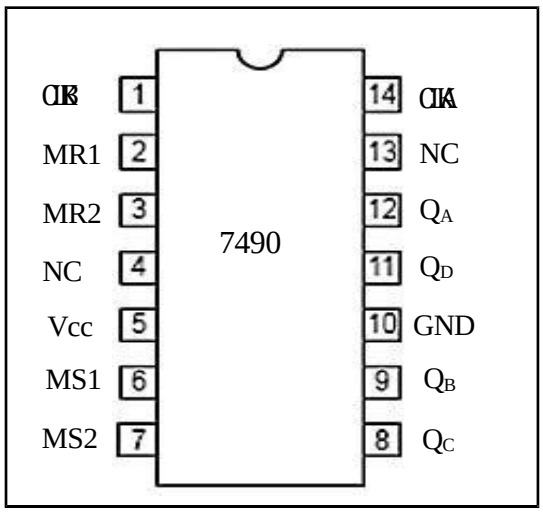


Figure 6.4: Pin diagram of IC 7490

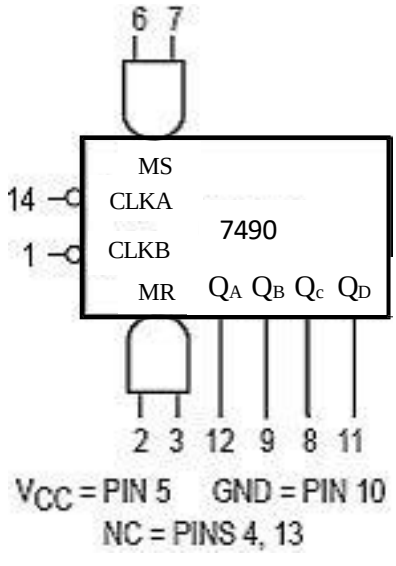


Figure 6.5: Logic symbol of IC 7490

Reset/Set inputs				Outputs			
MR1	MR2	MS1	MS2	Q_D	Q_C	Q_B	Q_A
1	1	0	X	0	0	0	0
1	1	X	0	0	0	0	0
X	X	1	1	1	0	0	1
X	0	X	1	Count			
0	X	0	X	Count			
0	X	X	0	Count			
X	0	0	X	Count			

	Outputs			
Count	QD	QC	QB	QA
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

Table 6.3: Mode selection and Count table of 7490 (Decade counter)

IC 7447 :

The IC 7447 BCD-to-Seven segment decoder. The IC is stand-alone and requires no external components other than the LED current limiting resistors. The output of the IC has complete ripple blanking and requires no external driver transistors. It incorporates automatic leading/or trailing-edge, zero blanking control ($\overline{RBI}$ and $\overline{BI/RBO}$). A Lamp test of these devices may be performed at any time when the $\overline{BI/RBO}$ mode is at a high logic level.

The 7447 decodes the input data in the pattern indicated in the Table 6.4 and the segment identification illustration. If the input data is decimal zero, a LOW signal applied to the $\overline{RBI}$ blanks the display and causes a multi digit display. For example, by grounding the $\overline{RBI}$ of the highest order decoder and connecting its $\overline{BI/RBO}$ to $\overline{RBI}$ of the next lowest order decoder, etc., leading zeros will be suppressed. Similarly, by grounding RBI of the lowest order decoder and connecting its $\overline{BI/RBO}$ to $\overline{RBI}$ of the next highest order decoder, etc., trailing zeros will be suppressed. Leading and trailing zeros can be suppressed simultaneously by using external gates, i.e.: by driving $\overline{RBI}$ of a intermediate decoder from an OR gate whose inputs are $\overline{BI/RBO}$ of the next highest and lowest order decoders. $\overline{BI/RBO}$ also serves as an unconditional blanking input. The internal NAND gate that generates the $\overline{BI/RBO}$ signal has a resistive pull-up, as opposed to a totem pole, and thus $\overline{BI/RBO}$ can be forced LOW by external means, using wired collector logic. A LOW signal thus applied to $\overline{BI/RBO}$ turns off all segment outputs. This blanking feature can be used to control display intensity by varying the duty cycle of the blanking signal. A LOW signal applied to $\overline{LT}$ turns on all segment outputs, provided that $\overline{BI/RBO}$ is not forced LOW

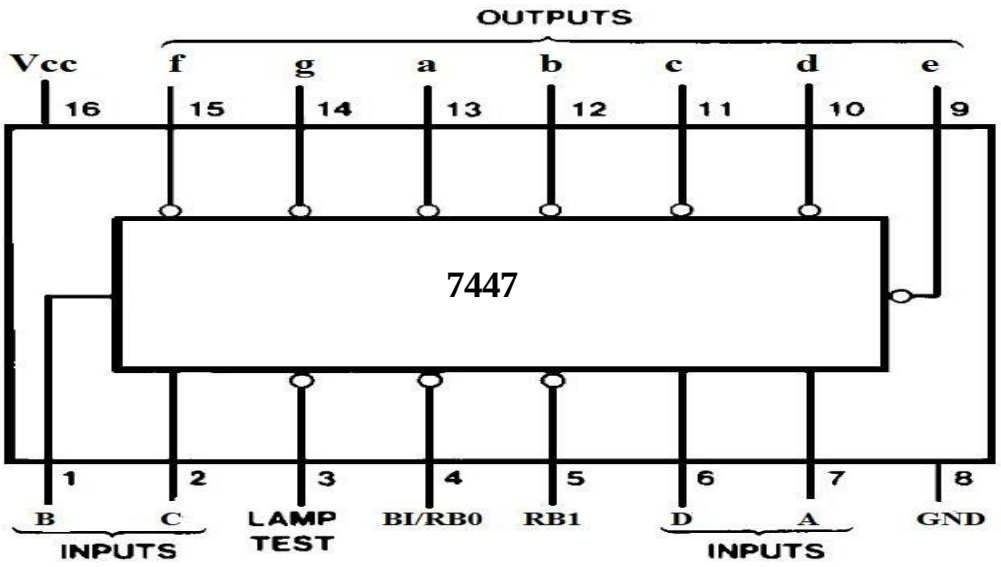


Figure 6.4: Pin diagram of IC 7447

Decimal or function	Inputs						BI / RBO	Outputs							Note
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g	
0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	
1	1	X	0	0	0	1	1	1	0	0	1	1	1	1	
2	1	X	0	0	1	0	1	0	0	1	0	0	1	0	
3	1	X	0	0	1	1	1	0	0	0	0	1	1	0	
4	1	X	0	1	0	0	1	1	0	0	1	1	0	0	
5	1	X	0	1	0	1	1	0	1	0	0	1	0	0	
6	1	X	0	1	1	0	1	1	1	0	0	0	0	0	
7	1	X	0	1	1	1	1	0	0	0	1	1	1	1	
8	1	X	1	0	0	0	1	0	0	0	0	0	0	0	
9	1	X	1	0	0	1	1	0	0	0	1	1	0	0	(1)
10	1	X	1	0	1	0	1	1	1	1	0	0	1	0	
11	1	X	1	0	1	1	1	1	1	0	0	1	1	0	
12	1	X	1	1	0	0	1	1	0	1	1	1	0	0	
13	1	X	1	1	0	1	1	0	1	1	0	1	0	0	
14	1	X	1	1	1	0	1	1	1	1	0	0	0	0	
15	1	X	1	1	1	1	1	1	1	1	1	1	1	1	
BI	X	X	X	X	X	X	0	1	1	1	1	1	1	1	(2)
RBI	1	0	0	0	0	0	0	1	1	1	1	1	1	1	(3)
LT	0	X	X	X	X	X	1	0	0	0	1	0	0	0	(4)

Table 6.4: Function table for IC 7447.

Note (1): The blanking input ($\overline{BI}$) must be open or held at a high logic level when output functions 0 through 15 are desired. The ripple-blanking input ($\overline{RBI}$) must be open or high if blanking of a decimal zero is not desired.

Note (2): When a low logic level is applied directly to the blanking input ($\overline{BI}$), all segment outputs are high regardless of the level of any other input.

Note (3): When ripple-blanking input ($\overline{RBI}$) and inputs A, B, C, and D are at a low level with the lamp test input high, all segment outputs go high and the ripple- blanking output ($\overline{RBO}$) goes to a low level (response condition).

Note (4): When the blanking-output ($\overline{BI}/\overline{RBO}$) is open or held high and a low is applied to the lamp-test input, all segment outputs are low. Input ($\overline{BI}/\overline{RBO}$) is awire-AND logic serving as a blanking input ($\overline{BI}$) and /or ripple blanking output ($\overline{RBO}$).

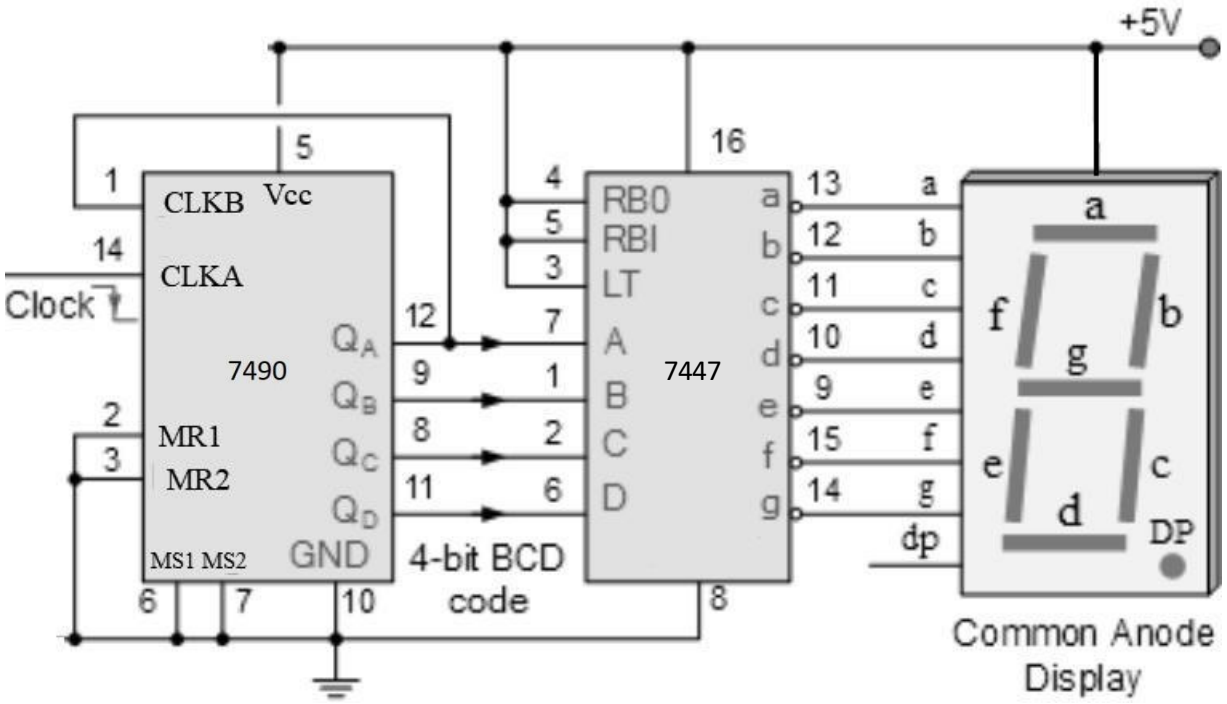


Figure 6.5: Logic diagram of Decade counter using IC 7490 and IC 7447

EXPERIMENT - 7

Design and simulation of a pseudo random sequence generator in Verilog.

Implementation of pseudo random sequence generator using shift register.

Aim:

Component/Software Used:

Component/Software	Specification
ICs	7495,7486
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	As per requirement
Software(s) Used	Vivado 2016.1

Theory:

A register is a device capable of storing a number of bits. We know that a flip-flop has a memory element. Thus, flip-flop can be connected together to form a register. A register capable of shifting the binary information held in each cell to its neighboring cell, in a selected direction is called a shift register.

Shift registers are synchronous sequential circuits consists of a group of flip-flops connected so that each flip-flop transfers its bit of data to the next flip-flop of the register when a clock pulse arrives. Data may have to be shifted left or shifted right. We have shifted left and shifted right registers.

An n - bit register consists of a group of n flip-flops capable of storing n bits of binary information. In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks. The flip-flops hold the binary information. and the gates determine how the information is transferred into the register.

Serial in - Serial out:

The data is loaded into and read from the shift register serially. Fig.7.1 shows the circuit of a 4-bit serial in-serial out (SISO) shifts right register using four D flip-flops. The Q outputs of one state (FF) are connected to the D input of the next state(FF). Thus, the inputs to second, third and fourth flip-flop are conditioned by their preceding flip-flops. The data in each flip-flop is shifted to the next flip-flop on the arrival of a positive edge of clock pulse. Since it is a 4 bit register, 4 clock pulses are required to shift the data through this register.

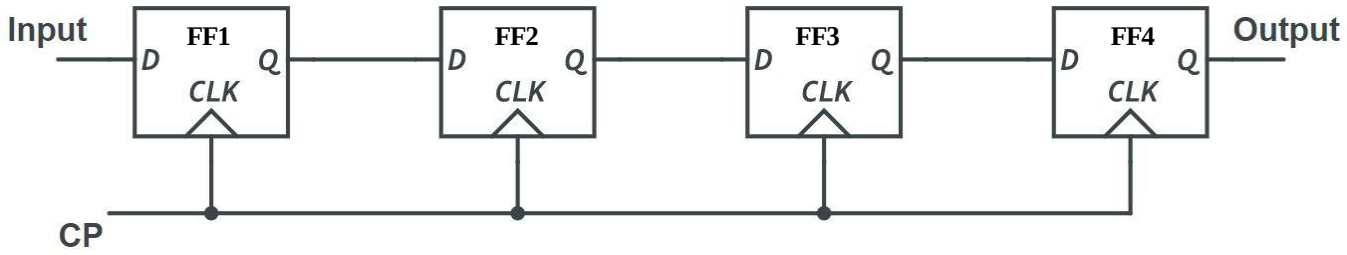


Figure 7.1: Logic diagram of 4 bit serial-in serial-out shift register

The data shifting in the 4-bit SISO is summarized below in the Table.7.1 with an example of 4 bit data "1110". Load the shift register with a 4-bit data '$B_3B_2B_1B_0$'one by one serially. In the 1st clock pulse, B_0 is shifted to Q_1 (FF1). In the 2nd clock pulse,B_0 is shifted to Q_2 (FF2) and B_1 is shifted to Q_1(FF1). In the 3rd clock pulse,B_0 is shifted to Q_3(FF3), B_1 is shifted to Q_2(FF2) and B_3 is shifted to Q_1(FF1). At the end of 4th clock pulse, $B_0$is shifted to Q_4(FF4), B_1 is shifted to Q_3(FF3), B_2 is shifted to Q_2 (FF2) and B_3 is shifted to Q_1 (FF1). In the next clock pulse, the second data 'B_1'will appear at Q_4 (FF4). In the next clock pulse, the third data 'B_3' will appear at Q_4 (FF4).Application of next clock pulse will enable the 4th data 'B_3' to appear at Q_4(FF4).Thus the data applied serially at the input comes out serially at Q_4(FF4).

Clock	Serial I/P	Q_1	Q_2	Q_3	Q_4
1	$B_0 = 0$	0	X	X	X
2	$B_1 = 1$	1	0	X	X
3	$B_2 = 1$	1	1	0	X
4	$B_3 = 1$	1	1	1	0
5	X	X	1	1	1
6	X	X	X	1	1
7	X	X	X	X	1

Table 7.1: Truth table of 4-bit SISO shift register

Parallel in-Parallel out:

The data is loaded in parallel and read from the register in parallel, i.e., all bits are loaded simultaneously and read simultaneously. Figure 7.2 shows the circuit of a 4-bit parallel in-parallel out (PIPO) shifts right register using four D flip-flops. The data in each flip-flop is shifted to the output the arrival of a positive edge of clock pulse. Output terminals and data are available at all of them together. The parallel data bits are $B_0B_1B_2B_3$ and Q1, Q2, Q3, and Q4 are parallel data outputs. After one clock pulse, all the input data is available in the outputs.

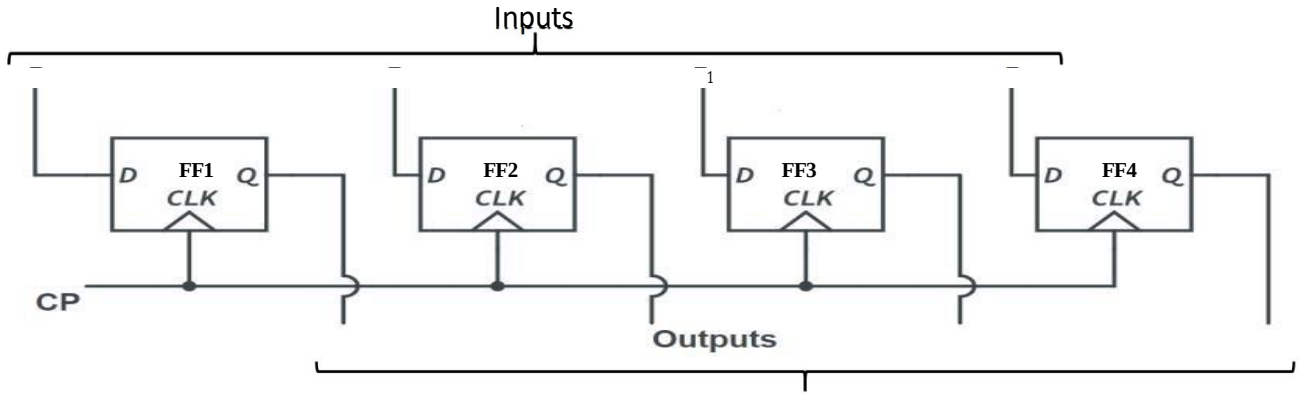


Figure 7.2: Logic diagram of PIPO shift register

Serial in-Parallel out: The data is loaded into the register serially but read in parallel mode, i.e., data is available from all flip-flops simultaneously. Figure 7.3 shows the circuit of a 4-bit serial in-parallel out (SIPO) shifts right register using four D flip-flops. The Q outputs of one state are connected to the D input of the next state. Thus, the inputs to second, to the third and fourth flip-flop are conditioned by their preceding flip-flops. The data in each flip-flop is shifted to the next flip-flop on the arrival of a positive edge of the clock pulse. Output terminals and data are available atall of them together. Since it is a 4 bit register, after 4 clock pulses we see all the datais available in the outputs of this register.

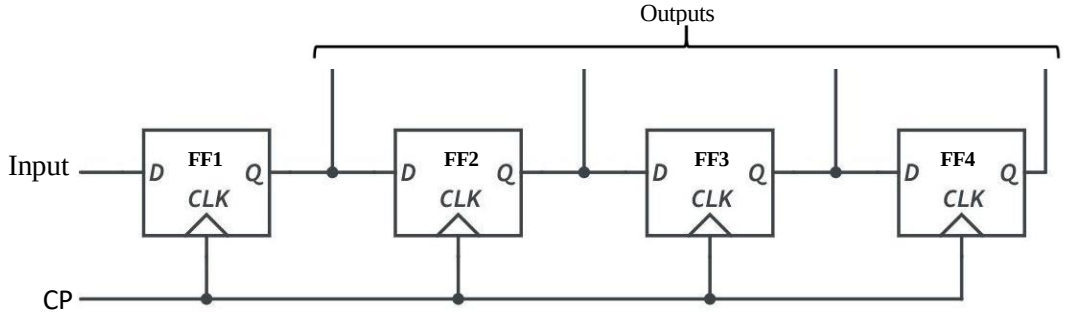


Figure 7.3: Logic diagram of PIPO shift register

The data shifting in the 4-bit SIPO is summarized below in the Table.7.2 with an example of 4 bit data ”1110”. Load the shift register with a 4-bit data ‘$B_3B_2B_1B_0$’one by one serially. In the 1st clock pulse,B_0 is shifted to Q_1(FF1). In the 2nd clock pulse,B_0 is shifted to Q_2 (FF2) and B_1 is shifted to Q_1(FF1). In the 3rd clock pulse,B_0 is shifted to Q_3 (FF3), B_1 is shifted to Q_2 (FF2) and B_3 is shifted to Q_1 (FF1). At the end of 4th clock pulse, $B_0$is shifted to Q_4(FF4), B_1 is shifted to Q_3(FF3), B_2 is shifted to Q_2 (FF2) and B_3 is shifted to Q_1(FF1). Thus after 4th clock pulse all the outputs are available.

Clock	Serial I/P	Q_1	Q_2	Q_3	Q_4
1	$B_0 = 0$	0	X	X	X
2	$B_1 = 1$	1	0	X	X
3	$B_2 = 1$	1	1	0	X
4	$B_3 = 1$	1	1	1	0

Table 7.2: Truth table of 4-bit SIPO shift register

Parallel in-Serial out: The data is loaded in parallel form and read serially. Figure 7.4 shows the circuit of a 4-bit parallel in-serial out (PISO) shifts right with four bits. It uses D flip-flops and four data input lines: B_0 (LSB), B_1, B_2, and B_3 (MSB).

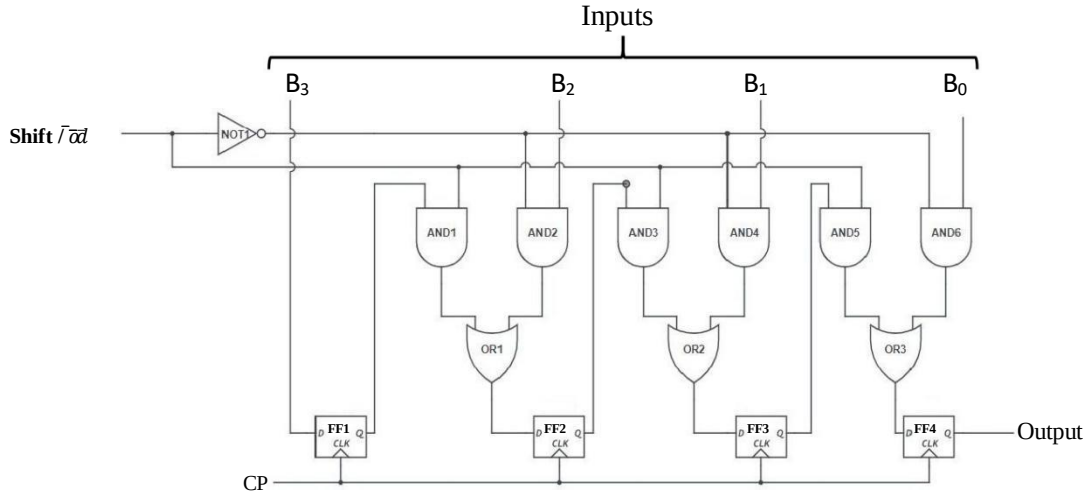


Figure 7.4: Logic diagram of PISO shift register

The data shifting in the 4-bit PISO is summarized below in the Table 7.3 with an example of 4 bit data "1110". Load the shift register with a 4-bit data '$B_3B_2B_1B_0$' simultaneously in the 1st clock pulse. In the 2nd clock pulse,B_3 is shifted to Q_2 (FF2), B_2 is shifted to Q_3 (FF3) and B_1 is shifted to Q_4 (FF4) . In the 3rd clock pulse,B_3 is shifted to Q_3 (FF3) and B_2 is shifted to Q_4 (FF4). At the end of 4th clock pulse B_3 is shifted to Q_4(FF4).

Clock	Parallel I/P	Q_1	Q_2	Q_3	Q_4
1	$B_3B_2B_1B_0$ = 1110	1	1	1	0
2		X	1	1	1
3		X	X	1	1
4		X	X	X	1

Table 7.3: Truth table of 4-bit PISO shift register

IC7495:

The IC-7495 is a 4-Bit Shift Register with serial and parallel synchronous operating modes. The serial shift right and parallel load are activated by separate clock inputs which are selected by a mode control input. The data is transferred from the serial or parallel D inputs to the Q outputs synchronous with the HIGH to LOW transition of the appropriate clock input. It has a Serial (D_S) and four Parallel ($P_0 – P_3$) Data inputs and four Parallel Data outputs ($Q_0 – Q_3$). The serial or parallel mode of operation is controlled by a Mode Control input (S) and two Clock Inputs ($\overline{P}_1$) and ($\overline{P}_2$). The serial (right-shift) or parallel data transfers occur synchronous with the HIGH to LOW transition of the selected clock input.

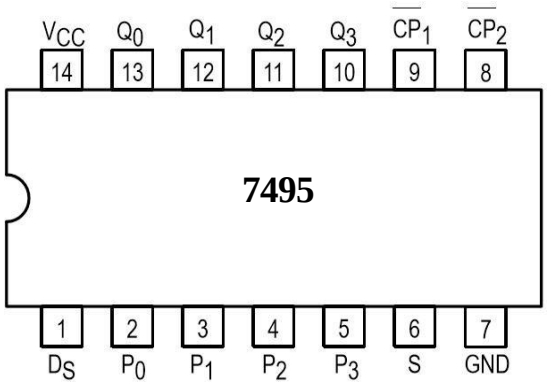


Figure 7.5: Pin diagram of IC 7495

Pin No	Pin Name	Pin Description
1	D_S	Serial Data Input
2 to 5	P_0 to P_3	Parallel Data Inputs 0 to 3
6	S	Mode Control Input
7	GND	Ground Pin
8	$\overline{P}_2$	Negative edge Parallel Clock input
9	$\overline{P}_1$	Negative edge Serial Clock input
10 to 13	Q_3 to Q_0	Parallel Outputs 3 to 0
14	Vcc	Supply Voltage

Table 7.4: Pin description of IC 7495

When the Mode Control input (S = 1) is HIGH, $\overline{P}_2$ is enabled. A HIGH to LOW transition on enabled $\overline{P}_2$ transfers parallel data from the $P_0 – P_3$ inputs to the $Q_0 – Q_3$ outputs.

When the Mode Control input (S = 0) is LOW, $\overline{P}_1$ is enabled. A HIGH to LOW transition on enabled $\overline{P}_1$ transfers the data from Serial input (D_S) to Q_0 and shifts the data in Q_0 to Q_1, Q_1 to Q_2, and Q_2 to Q_3 respectively (right-shift).

A left-shift is accomplished by externally connecting Q_3 to P_2, Q_2 to P_1 & Q_1 to P_0 and operating the IC7495 in the parallel mode (S = 1) HIGH.

For normal operation, S should only change states when both Clock inputs are LOW. However, changing S from LOW to HIGH while $\overline{P}_2$ is HIGH, or changing S from HIGH to LOW while $\overline{P}_1$ is HIGH and CP_2 is LOW will not cause any changes on the register outputs.

Implementation of various shift registers using IC 7495 has been show below.

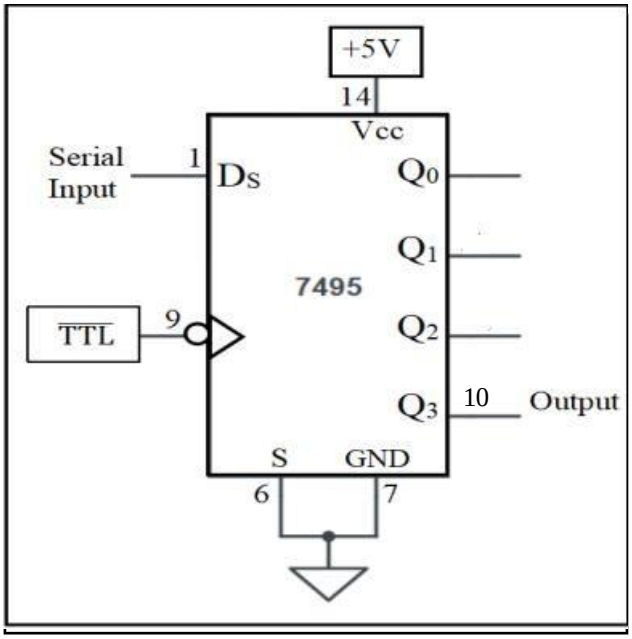


Figure 7.6: Logic diagram of 4 bit SISO shift register using IC 7495

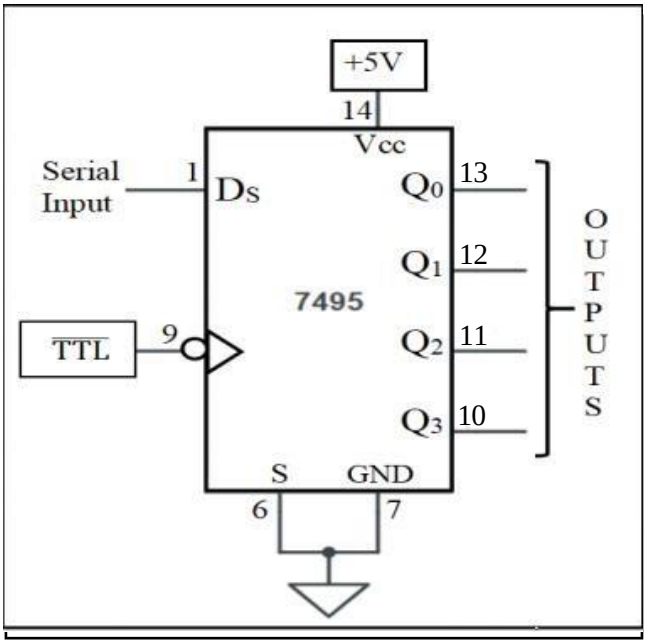


Figure 7.7: Logic diagram of 4 bit SIPO shift register using IC 7495

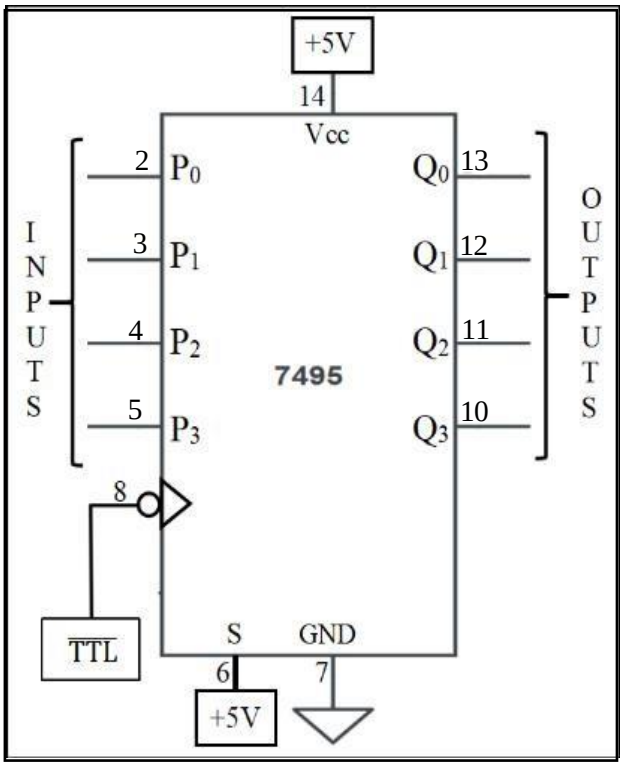


Figure 7.8: Logic diagram of 4 bit PIPO shift register using IC 7495

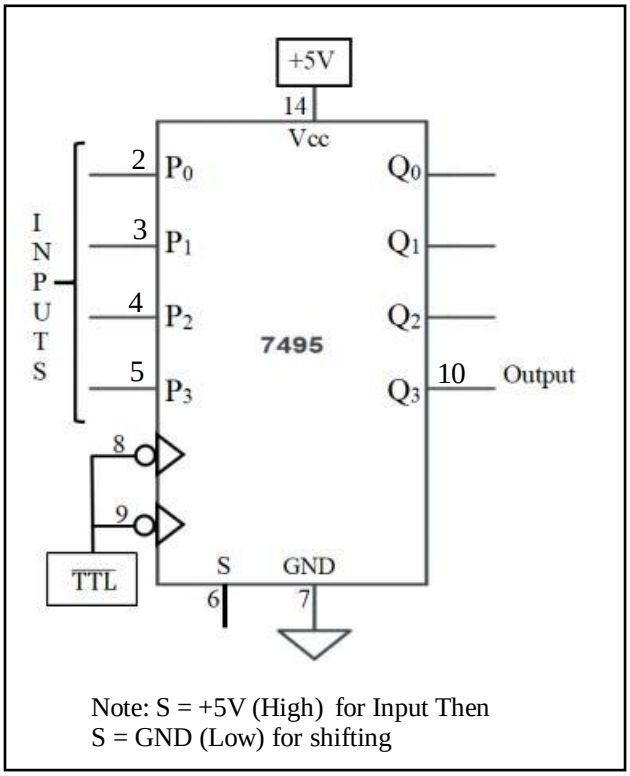


Figure 7.9: Logic diagram of 4 bit PISO shift register using IC 7495

Pseudo Random Sequence generator:

The sequence generator digital circuit which generates a set of outputs. This is a shift register where the input is a combinational logic function of the outputs of the flip-flops of the shift register. The sequence generator generates a sequence of binary bits. Thus, the length of the sequence is related to the number of flip-flops that are required to design a sequence generator. These generators are utilized in a wide variety of applications like coding and control. The length of the sequence is related to the number of flip-flops that are required to produce a sequence generator.

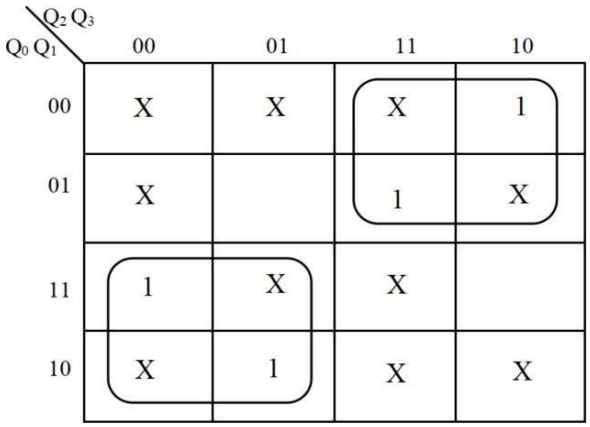
$$\leq -1$$

where **L** is the length of the sequence and **n** is the minimum number of flip-flops required.

Designing of a sequence generator is shown using IC7495 for the sequence "**1101001**" .

Q_0	Q_1	Q_2	Q_3	D_S
1	1	0	0	1
1	1	1	0	0
0	1	1	1	1
1	0	1	1	0
0	1	0	1	0
0	0	1	0	1
1	0	0	1	1

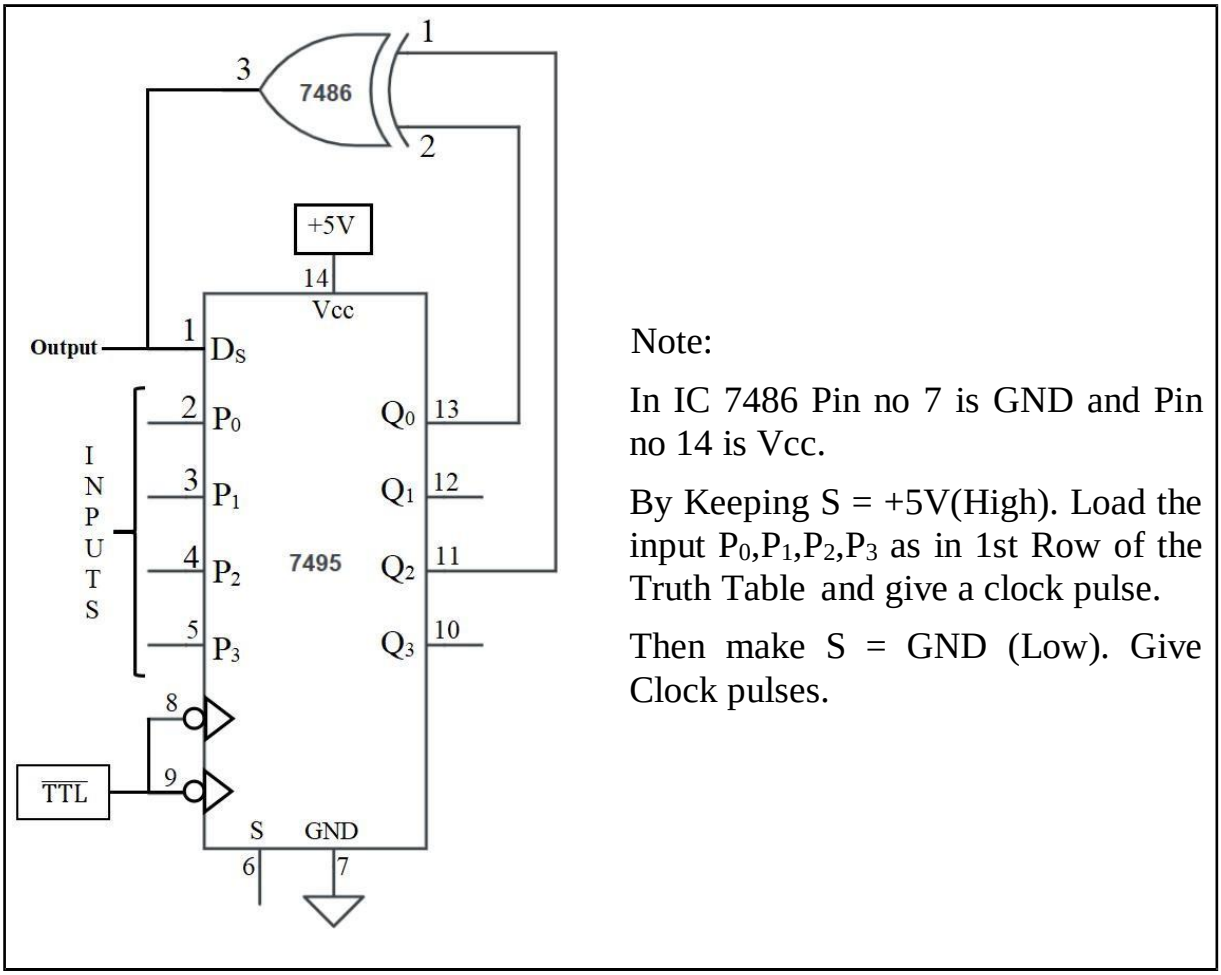
Table 7.5: Truth Table of the Sequence Generator



K-Map for the Input(D_S) of the shift register

$$(_{0,1,2,3}) = \sum(2,7,9,12) + (0,1,3,4,6,8,10,13,15) = _2 \oplus _0$$

Boolean expression for the input D_s



Note:

In IC 7486 Pin no 7 is GND and Pin no 14 is Vcc.

By Keeping S = +5V(High). Load the input P_0,P_1,P_2,P_3 as in 1st Row of the Truth Table and give a clock pulse.

Then make S = GND (Low). Give Clock pulses.

Figure 7.10: Logic diagram of "1101001" sequence generator using IC 7495

Design Problem

Design and Simulation of 4 bit Ring counter and 4 bit Johnson counter using Verilog HDL.

Hardware implementation of 4 bit Ring counter and 4 bit Johnson counter.

Solution:

Ring Counter:

Ring counter is a sequential logic circuit that is constructed using shift register. Same data recirculates in the counter depending on the clock pulse.It consists of a group of flip-flops connected in a circular chain or "ring" formation, where the output of one flip-flop is connected to the input of the next, and the last flip-flop is connected back to the first.

When a clock pulse is applied to the ring counter, the data is shifted from one flip-flop to the next, with each flip-flop in the ring taking turns being in the "high" or "low" state. The output of each flip-flop represents a different bit of the counter's binary value, with the least significant bit (LSB) being the first flip-flop in the ring, and the most significant bit (MSB) being the last flip-flop.

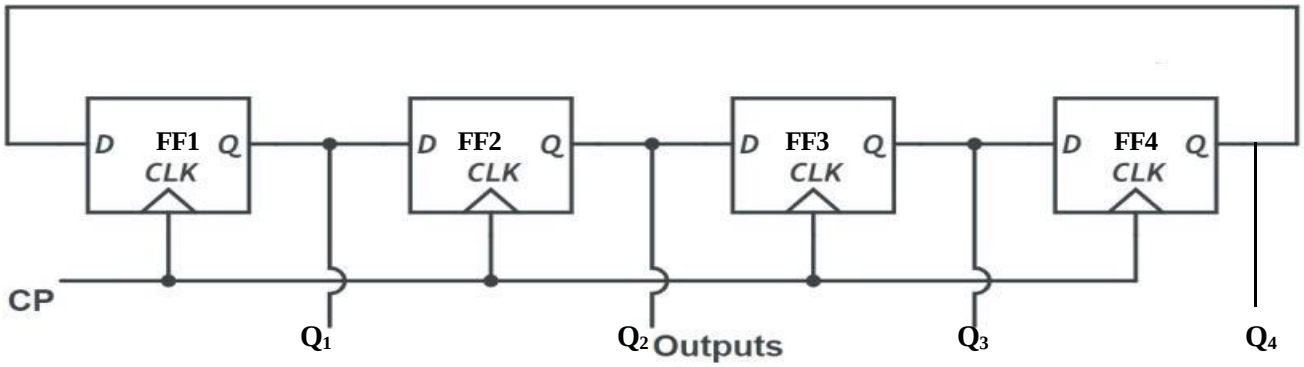


Figure 7.11: Logic diagram of 4 bit Ring Counter

Assume, initial status of the D flip-flops from leftmost to rightmost is Q_1 Q_2 Q_3 Q_4 = "1000" .This status repeats for every four(4) clock pulse. which is shown in below Table 7.6. A "mod-n" ring counter will require "n" number of flip-flops connected together to circulate a single data bit providing "n" different output states.Each state repeats after every "n" clock pulse.

Clock Pulse	Q_1	Q_2	Q_3	Q_4
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Table 7.6: Truth table of 4-bit Ring Counter

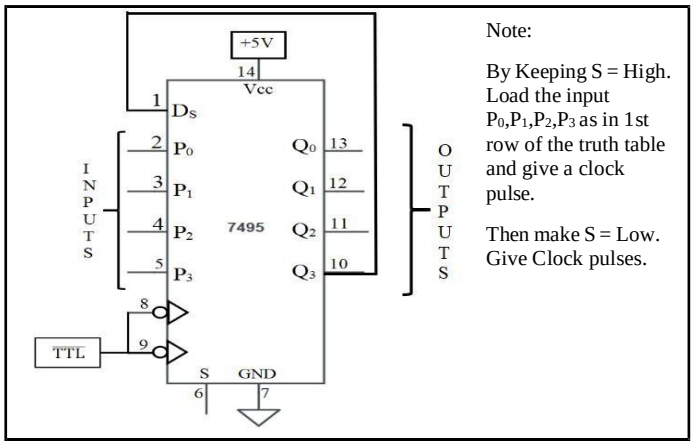


Figure 7.12: Logic diagram of Ring Counter using IC 7495

Johnson Counter:

A Johnson counter is a type of shift register that is similar to a ring counter, but with an additional invert stage. It is sometimes also called a "twisted ring counter" counter". The operation of a Johnson counter is similar to that of a ring counter, with each flip-flop output changing state on each clock pulse. However, in a Johnson counter, the output of the last flip-flop is inverted and fed back into the first flip-flop. This causes the counter to cycle through a sequence of states that includes both ascending and descending binary values.

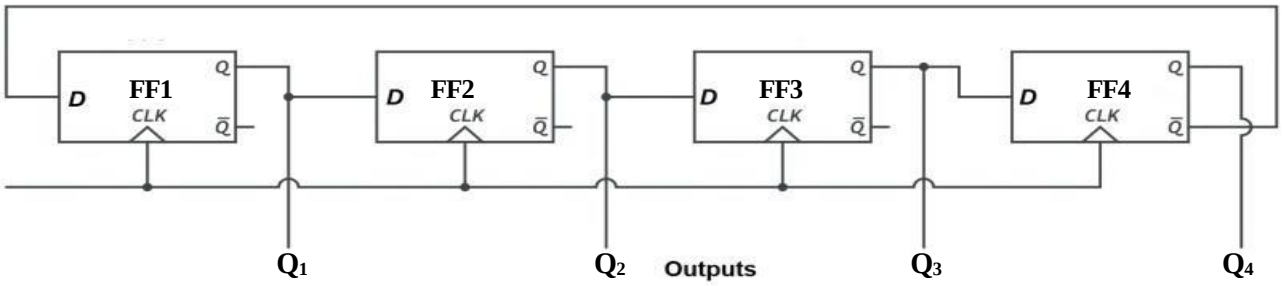


Figure 7.13: Logic diagram of Johnson Counter

Assume, initial status of the D flip-flops from leftmost to rightmost is Q1 Q2 Q3 Q4 = "0000" .This status repeats for every Eight (8) clock pulse which is shown in below Table 7.7.

A "mod-2n" ring counter will require "n" number of flip-flops connected together to circulate a single data bit providing "2n" different output states.Each state repeats after every "2n" clock pulse.

Clock Pulse	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Table 7.7: Truth table of 4-bit Johnson Counter

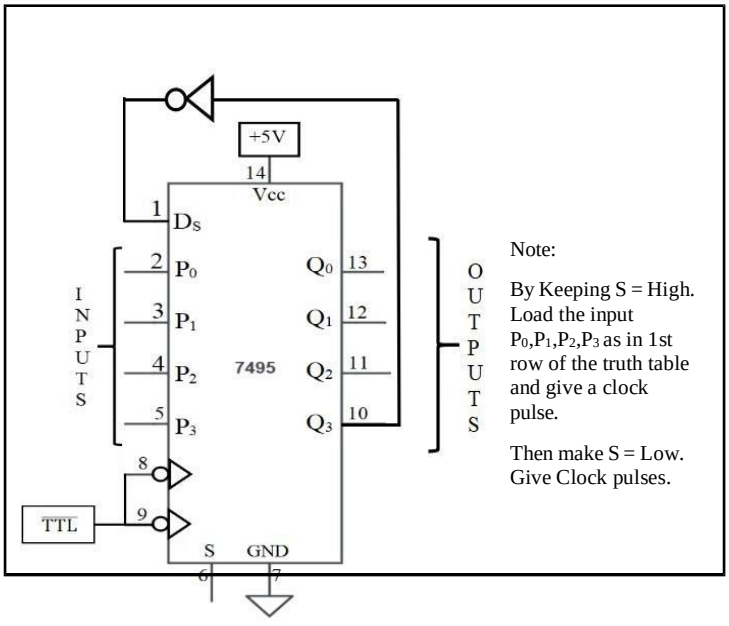


Figure 7.14: Logic diagram of 4-bit Johnson Counter using IC 7495

EXPERIMENT - 8

Design and simulation of a finite state machine in Verilog to detect a given sequence of bits.

Aim:

Component/Software Used:

Component/Software	Specification
ICs	-
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	-
Software(s) Used	Vivado 2016.1

Theory:

A finite state machine is a sequential logic circuit that has a finite number of defined states that can be represented. A finite state machine requires the use of memory to store the state of the machine. Combinational logic is used to combine the values of the present state along with inputs to the system to determine the next state of the system.

An example of a simple state machine could be a counter that counts from from 0 to 1 to 2 to 3 and back to 0. In this case, the state machine does not have any input at all, it uses the past state and increments the value every clock cycle. An example of a complex state machine would be a computer. In this case, the computer can have many different inputs and has many different states. Input data can come from the keyboard, network, mouse, memory, etc., while the state would normally be associated with the address in memory of the program being run. In this text, the state machines will be like the counter just described and certainly nothing as complex as a computer. State machines are used in more than just computers. Any process that can be defined with a given predictable algorithm can often be represented by an electronic state machine.

There are two methods to design finite state machines, first is **Mealy** and second is **Moore**. In the Mealy model, the output is a function of **both the present state and the input**.

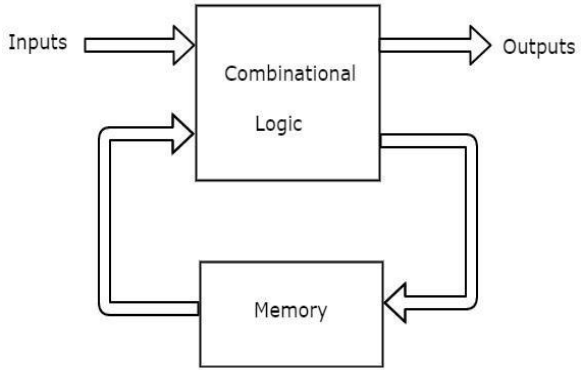


Figure 8.1: Block diagrams of Mealy state machine

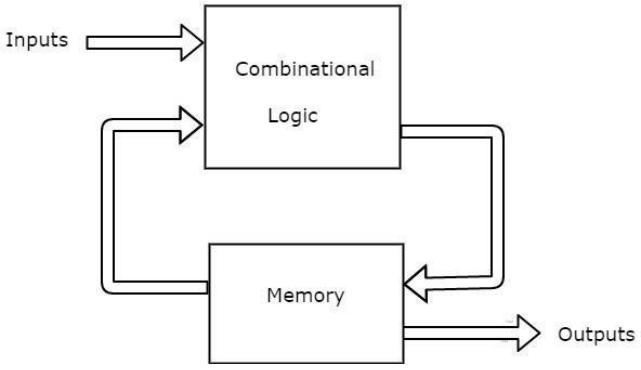


Figure 8.1: Block diagrams of Moore state machine

In a Moore model, the outputs of the sequential circuit are synchronized with the clock, because they depend only on flip-flop outputs that are synchronized with the clock. In a Mealy model, the outputs may change if the inputs change during the clock cycle. Moreover, the outputs may have momentary false values because of the delay encountered from the time that the inputs change and the time that the flip-flop outputs change. In order to synchronize a Mealy-type circuit, the inputs of the sequential circuit must be synchronized with the clock and the outputs must be sampled immediately before the clock edge. The inputs are changed at the inactive edge of the clock to ensure that the inputs to the flip-flops stabilize before the active edge of the clock occurs.

In order to design a state machine one would need to recognize the inputs of the system, the states, and how it transitions from one state to the next. This is graphically represented with a state transition diagram. Then, the transition diagram should be used to create a truth table that has the inputs to the system and current state values as inputs in that table. The output of the truth table is the next state of the system. Combinational logic is used to implement the functions required to obtain the next state values for the state machine. The Boolean logic minimization techniques are used.

Sequence detector is sequential machine that generates an output **1** every time the desired sequence is detected and output **0** at rest all other times. In this experiment a sequence detector for bit sequence '1011' is designed. The input is represented with D_{in} and output is represented with D_{out}. The design of the sequence detector is illustrated below using both the methods.

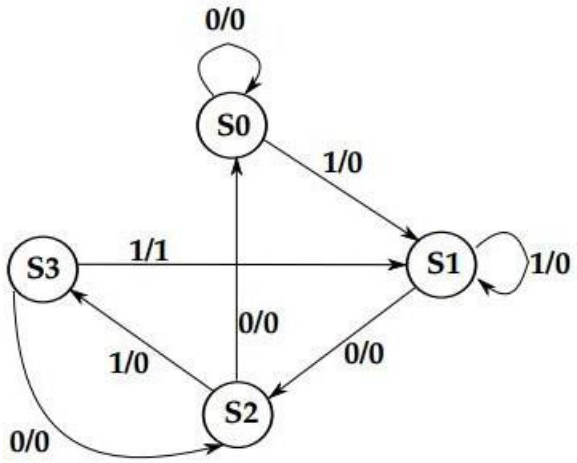


Figure 8.1: **Mealy State Machine for Detecting a Sequence of '1011'**

- When in initial state (S0) the machine gets the input of '1' it jumps to the next state with the output equal to '0'. If the input is '0' it stays in the same state.
- When in 2nd state (S1) the machine gets an input of '0' it jumps to the 3rd state with the output equal to '0'. If it gets an input of '1' it stays in the same state.
- When in the 3rd state (S2) the machine gets an input of '1' it jumps to the 4th state with the output equal to '0'. If the input received is '0' it goes back to the initial state.
- When in the 4th state (S3) the machine gets an input of '1' it jumps back to the 2nd state, with the output equal to '1'. If the input received is '0' it goes back to the 3rd state.

Present State	Next State		Output (D_{out})	
	$D_{in} = 0$	$D_{in} = 1$	$D_{in} = 0$	$D_{in} = 1$
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S0	S3	0	0
S3	S2	S1	0	1

Table 8.1: **Mealy State table for Detecting a Sequence of '1011'**

As there are no redundant states thus the final states are S0, S1, S2 and S3. There four states therefore two state variables are required. Binary values are assigned to the states arbitrarily.

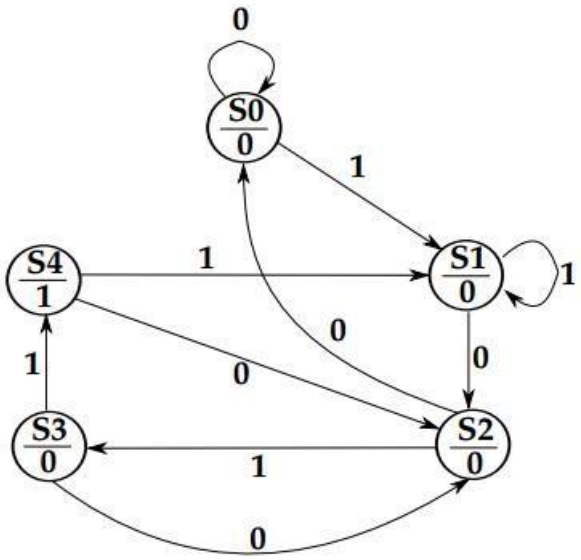


Figure 8.2 : Moore State Machine for Detecting a Sequence of '1011'

- In initial state (S0) the output of the detector is '0'. When machine gets the input of '1' it jumps to the next state. If the input is '0' it stays in the same state.
- In 2nd state (S1) the output of the detector is '0'. When machine gets an input of '0' it jumps to the 3rd state. If it gets an input of '1' it stays in the same state.
- In the 3rd state (S2) the output of the detector is '0'. When machine gets an input of '1' it jumps to the 4th state. If the input received is '0' it goes back to the initial state.
- In the 4th state (S3) the output of the detector is '0'. When machine gets an input of '1' it jumps to the 5th state. If the input received is '0' it goes back to the 3rd state.
- In the 5th state the output of the detector is '1'. When machine gets an input of '0'it jumps to the 3rd state, otherwise it jumps to the 2nd state.

Present State	Next State		Output (D_{out})
	$D_{in} = 0$	$D_{in} = 1$	
S0	S0	S1	0
S1	S2	S1	0
S2	S0	S3	0
S3	S2	S4	0
S4	S2	S1	1

Table 8.2: **Moore State table for Detecting a Sequence of '1011'**

As there are no redundant states thus the final states are S0, S1, S2, S3 and S4. There five states therefore three state variables are required. Binary values are assigned to the states arbitrarily.

After designing the state machines the models have to be transformed into Verilog code describing it.

```
// Design source of Sequence detector of sequence "1011"
module Seq_Det (input clk, input rst, input Din, output Dout);
reg [1:0] state;
reg Dout;
initial
begin
state <= 2'b00;
end
always @ ( posedge clk, rst )
begin
if ( rst )
state <= 2'b00;
else
begin
case( {state,Din} )
3'b000: begin
state <= 2'b00;
end
3'b001: begin
state <= 2'b01;
end
3'b010: begin
state <= 2'b10;
end
3'b011: begin
state <= 2'b01;
end
3'b100: begin
state <= 2'b10;
end
3'b101: begin
state <= 2'b11;
end
3'b110: begin
state <= 2'b10;
end
3'b111: begin
state <= 2'b01;
end
endcase
end
assign Dout = (({state,Din}) == 3'b111) ? 1'b1 : 1'b0 ;
end
endmodule
```