Experiment No. 1 **Date:**

Objectives: Multiplexer in Verilog

Software: Vivado Software

Design Description: -

A multiplexer is a data selector device that selects one input from several input lines, depending upon the enabled, select lines, and yields one single output.

A multiplexer of 2n inputs has n select lines, are used to select which input line to send to the output. There is only one output in the multiplexer, no matter what's its configuration.

These devices are used extensively in the areas where the multiple data can be transferred over a single line like in the communication systems and bus architecture hardware. Visit this post for a crystal clear explanation to multiplexers.

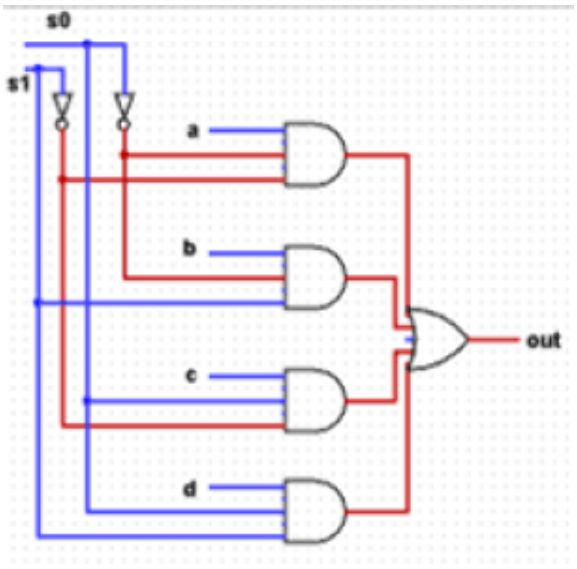


Fig. 1.1 4x1 Multiplexer

Truth Table :-

The truth table of the 4:1 MUX has six input variables, out of which two are select lines, and one is the output signal. The input data lines a, b, c, d are selected depending on the values of the select lines.

Select Lines		Output
s1	s0	out
0	0	a
0	1	b
1	0	c
1	1	d

Truth table of 4×1 Mux

Procedure:

Step 1: Start the Vivado IDE by using the desktop shortcut or by using the Start ➔ Programs ➔ Vivado IDE.

Step 2: Create New Project➔ Select RTL Project ➔ Click on Next ➔ Choose the Board, Vendor and Processor ➔ click on Next ➔ Finish

Step 3: In the Flow Navigator window ➔ Project Manager➔ Add Sources➔ Add or Create Design Sources➔ Create File ➔ VHDL/VERILOG➔ Define IN and OUTs ➔ Finish.

Step 4: Write the Code Editorial window and save it. Add Sources ➔ Add or Create Simulation Sources➔ Finish➔ Run simulation.

Step 5: Check the RTL Schematic of the source code. Flow Navigator window➔ RTL Analysis➔ Open Elaborated Design.

Step 6: Generate Bit Stream ➔ Open Hardware Target ➔ Next ➔ Select Target hardware and hardware device ➔ Next ➔ Finish.

Step 7: Click on Program Device➔ Program. Finished.

Conclusion:

.

Rubrics for Evaluation:

Very good (5 Marks)	**Good (4 Marks)**	**Average (3 Marks)**	**Satisfactory (1 to 2 Marks)**
Criteria-1 :Functionality/Specifications			
The code is completely functional and responds correctly producing the correct outputs and or responses under all test cases	The code is mostly functional and responds correctly producing the correct outputs and or responses under most test cases. There are minor problems with the program implementation.	The code is marginally functional with numerous errors. The code may respond correctly under certain circumstances, but there are significant errors and/or incomplete code sections.	The code is minimally functional with significant portions of the code missing or incomplete. The code is largely nonresponsive to most test cases and/or inputs. Or copied.
Criteria-2 : Design Demonstration			
VERILOG code is executed. Simulated & demonstrated on target device. Explained the code and output fluently	VERILOG code is executed. Simulated & demonstrated on target device Explained the code and output partially	VERILOG code is Simulated & demonstrated on target device with some errors. Partially explained the code and output.	VERILOG code is Simulated but unable to demonstrated on target device & unable to explain the code and output.
Criteria-3 : Documentation			
The Code, Test bench, RTL, Simulation waveform and XDC file is extremely well documented.	The Code, Test bench, RTL, Simulation waveform and XDC file is reasonably well documented. There are minor formatting omissions that would have improved user understanding of code purpose.	The Code, Test bench, RTL, Simulation waveform and XDC file is poorly documented. There are no proper comments /titles given.	one or more documents from Code, Test bench, RTL, Simulation waveform and XDC file are missing. There are no proper comments /titles given. Or copied.

Experiment No. 2 **Date:**

Objectives: 8-bit Arithmetic and Logic Unit in Verilog

Software: Vivado Software

Design Description: -

ALU is the fundamental building block of the processor, which is responsible for carrying out the arithmetic and logic functions. ALU comprises of combinatorial logic that implements arithmetic operations such as Addition, Subtraction and Multiplication, and logic operations such as AND, OR, NOT. The ALU gets operands from the register file or memory. The block diagram of a typical ALU is shown in Figure 1. The ALU reads two input operands In A and In B. The operation to perform on these input operands is selected using the control input Opcode. The ALU performs the selected operation on the input operands In A and In B and produces the output, Out. The ALU also updates different flag signals after performing the selected function. Note that the ALU is purely combinatorial logic and contains no registers or latches.

The arithmetic functions are much more complex to implement than the logic functions. The performance of the ALU depends upon the architecture of each structural components of the ALU. In this example only some basic ALU functions are implemented. The ALU is divided into an arithmetic section and a logical section.

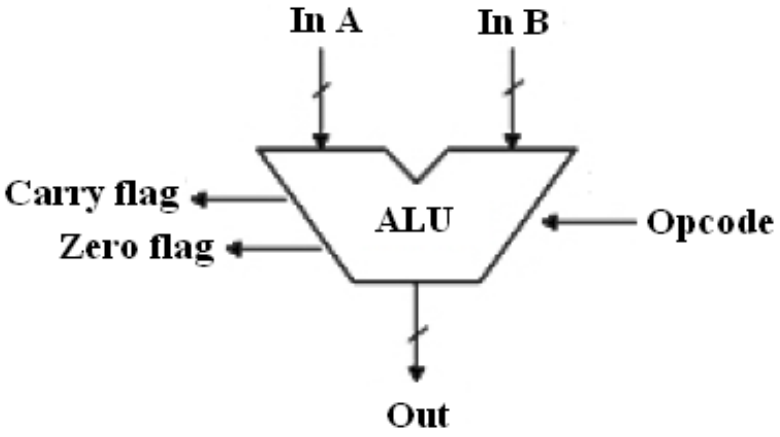


Fig. 2.1Arithmatic Logic Unit

Truth Table :-

Truth table of the 8-bit ALU is given below. The ALU works on 8-bit operands. It supports 16 instructions which are selected by the 4-bit Opcode.

ALU Arithmetic and Logic Operations

Opcode	ALU Operation
0000	ALU_Out = A + B;
0001	ALU_Out = A - B;
0010	ALU_Out = A * B;
0011	ALU_Out = A / B;
0100	ALU_Out = A << 1;
0101	ALU_Out = A >> 1;
0110	ALU_Out = A rotated left by 1;
0111	ALU_Out = A rotated right by 1;
1000	ALU_Out = A and B;
1001	ALU_Out = A or B;
1010	ALU_Out = A xor B;
1011	ALU_Out = A nor B;
1100	ALU_Out = A nand B;
1101	ALU_Out = A xnor B;
1110	ALU_Out = 1 if A>B else 0;
1111	ALU_Out = 1 if A=B else 0;

Procedure:

Step 1: Start the Vivado IDE by using the desktop shortcut or by using the Start ➔ Programs ➔ Vivado IDE.

Step 2: Create New Project➔ Select RTL Project ➔ Click on Next ➔ Choose the Board, - Vendor and Processor ➔ click on Next ➔ Finish

Step 3: In the Flow Navigator window ➔ Project Manager➔ Add Sources➔ Add or Create Design Sources➔ Create File ➔ VHDL /VERILOG➔ Define IN and OUTs ➔ Finish.

Step 4: Write the Code Editorial window and save it. Add Sources ➔ Add or Create Simulation Sources➔ Finish➔ Run simulation.

Step 5: Check the RTL Schematic of the source code. Flow Navigator window➔ RTL Analysis➔ Open Elaborated Design.

Step 6: Generate Bit Stream ➔ Open Hardware Target ➔ Next ➔ Select Target hardware and hardware device ➔ Next ➔ Finish.

Step 7: Click on Program Device➔ Program. Finished

Conclusion:

.

Rubrics for Evaluation:

Very good (5 Marks)	**Good (4 Marks)**	**Average (3 Marks)**	**Satisfactory (1 to 2 Marks)**
Criteria-1 :Functionality/Specifications			
The code is completely functional and responds correctly producing the correct outputs and or responses under all test cases	The code is mostly functional and responds correctly producing the correct outputs and or responses under most test cases. There are minor problems with the program implementation.	The code is marginally functional with numerous errors. The code may respond correctly under certain circumstances, but there are significant errors and/or incomplete code sections.	The code is minimally functional with significant portions of the code missing or incomplete. The code is largely nonresponsive to most test cases and/or inputs. Or copied.
Criteria-2 : Design Demonstration			
VERILOG code is executed. Simulated & demonstrated on target device. Explained the code and output fluently	VERILOG code is executed. Simulated & demonstrated on target device Explained the code and output partially	VERILOG code is Simulated & demonstrated on target device with some errors. Partially explained the code and output.	VERILOG code is Simulated but unable to demonstrated on target device & unable to explain the code and output.
Criteria-3 : Documentation			
The Code, Test bench, RTL, Simulation waveform and XDC file is extremely well documented.	The Code, Test bench, RTL, Simulation waveform and XDC file is reasonably well documented. There are minor formatting omissions that would have improved user understanding of code purpose.	The Code, Test bench, RTL, Simulation waveform and XDC file is poorly documented. There are no proper comments /titles given.	one or more documents from Code, Test bench, RTL, Simulation waveform and XDC file are missing. There are no proper comments /titles given. Or copied.

Experiment No. 3 **Date:**

Objectives: To build and investigate the operation of an D, T, SR, JK flip-flop, clocked.

Software: Vivado Software

Design Description: -

Basics and Overview of Flip Flops :

A flip flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Both are used as data storage elements. It is the basic storage element in sequential logic. But first, let's clarify the difference between a latch and a flip-flop.

Flip flop v/s Latch

The basic difference between a latch and a flip-flop is a gating or clocking mechanism.

In Simple words. Flip Flop is edge-triggered and a latch is level triggered.

For example, let us talk about SR latch and SR flip-flops. In this circuit when you Set S as active the output Q would be high and Q' will be Low. This is irrespective of anything else. (This is an active-low circuit so active here means low, but for an active high circuit active would mean high)

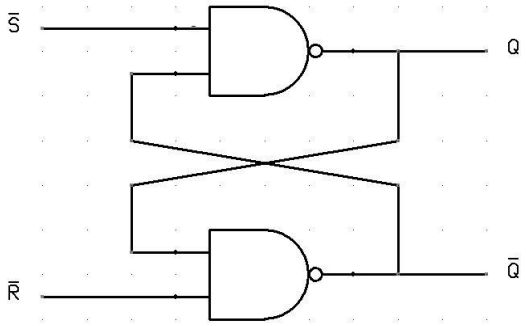


Fig. 3.1 SR Latch

A flip-flop, on the other hand, is synchronous and is also known as a gated or clocked SR latch.

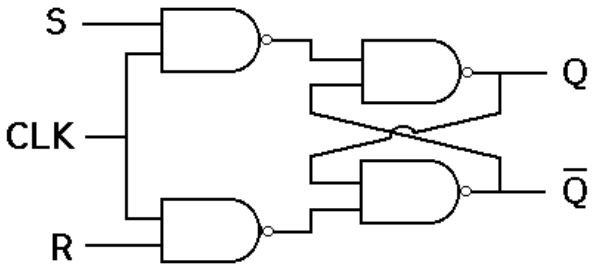


Fig. 3.2 SR Flip-Flop

In this circuit diagram, the output is changed (i.e. the stored data is changed) only when you give an active clock signal. Otherwise, even if the S or R is active the data will not change. Let's look at the types of flip-flops to understand better.

SR Flip Flop

There are majorly 4 types of flip-flops, with the most common one being SR flip-flop. This simple flip-flop circuit has a set input (S) and a reset input (R). In this system, when you Set "S" as active the output "Q" would be high and "Q'" will be low. Once the outputs are established, the wiring of the circuit is maintained until "S" or "R" go high, or power is turned off. As shown above, it is the simplest and easiest to understand. The two outputs, as shown above, are the inverse of each other. The truth table of SR Flip-Flop is highlighted below.

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

JK Flip-flop

Due to the undefined state in the SR flip-flop, another flip-flop is required in electronics. The JK flip-flop is an improvement on the SR flip-flop where S=R=1 is not a problem.

The input condition of J=K=1, gives an output inverting the output state. However, the outputs are the same when one tests the circuit practically.

In simple words, If J and K data input are different (i.e. high and low) then the output Q takes the value of J at the next clock edge. If J and K are both low then no change occurs. If J and K are both high at the clock edge then the output will toggle from one state to the other. JK Flip-Flops can function as Set or Reset Flip-flops

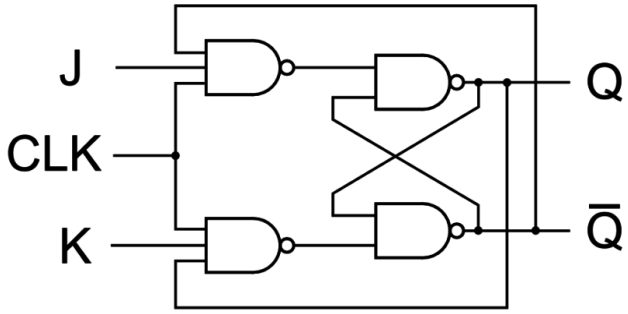


Fig. 3.3 JK Flip-Flop

J	K	Q	Q’
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	0

D Flip Flop

D flip-flop is a better alternative that is very popular with digital electronics. They are commonly used for counters and shift-registers and input synchronisation.

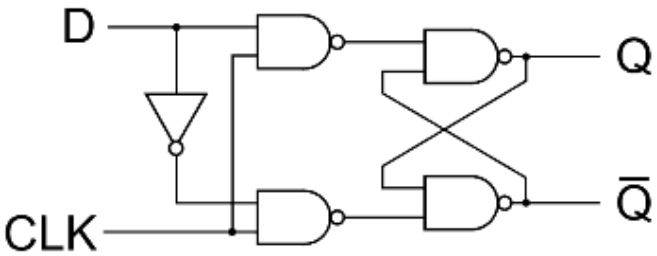


Fig. 3.4 D Flip-Flop

In this, the output can be only changed at the clock edge, and if the input changes at other times, the output will be unaffected.

Clock	D	Q	Q’
↓ » 0	0	0	1
↑ » 1	0	0	1
↓ » 0	1	0	1
↑ » 1	1	1	0

The change of state of the output is dependent on the rising edge of the clock. The output (Q) is same as the input and can only change at the rising edge of the clock.

T Flip Flop

A T flip-flop is like a JK flip-flop. These are basically a single input version of JK flip-flops. This modified form of JK flip-flop is obtained by connecting both inputs J and K together. It has only one input along with the clock input.

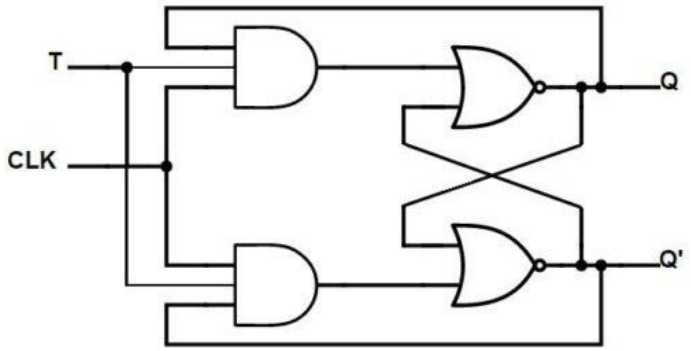


Fig. 3.4 T Flip-Flop

These flip-flops are called T flip-flops because of their ability to complement its state (i.e.) Toggle, hence the name Toggle flip-flop.

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

Applications of Flip-Flops

These are the various types of flip-flops being used in digital electronic circuits and the applications of Flip-flops are as specified below.

- Counters
- Frequency Dividers
- Shift Registers
- Storage Registers

Procedure:

Step 1: Start the Vivado IDE by using the desktop shortcut or by using the Start ➔ Programs ➔ Vivado IDE.

Step 2: Create New Project➔ Select RTL Project ➔ Click on Next ➔ Choose the Board, - Vendor and Processor ➔ click on Next ➔ Finish

Step 3: In the Flow Navigator window ➔ Project Manager➔ Add Sources➔ Add or Create Design Sources➔ Create File ➔ VERILOG/VERILOG➔ Define IN and OUTs ➔ Finish.

Step 4: Write the Code Editorial window and save it. Add Sources ➔ Add or Create Simulation Sources➔ Finish➔ Run simulation.

Step 5: Check the RTL Schematic of the source code. Flow Navigator window➔ RTL Analysis➔ Open Elaborated Design.

Step 6: Generate Bit Stream ➔ Open Hardware Target ➔ Next ➔ Select Target hardware and hardware device ➔ Next ➔ Finish.

Step 7: Click on Program Device➔ Program. Finished

Conclusion:

Rubrics for Evaluation:

Very good (5 Marks)	**Good (4 Marks)**	**Average (3 Marks)**	**Satisfactory (1 to 2 Marks)**
Criteria-1 :Functionality/Specifications			
The code is completely functional and responds correctly producing the correct outputs and or responses under all test cases	The code is mostly functional and responds correctly producing the correct outputs and or responses under most test cases. There are minor problems with the program implementation.	The code is marginally functional with numerous errors. The code may respond correctly under certain circumstances, but there are significant errors and/or incomplete code sections.	The code is minimally functional with significant portions of the code missing or incomplete. The code is largely nonresponsive to most test cases and/or inputs. Or copied.
Criteria-2 : Design Demonstration			
VERILOG code is executed. Simulated & demonstrated on target device. Explained the code and output fluently	VERILOG code is executed. Simulated & demonstrated on target device Explained the code and output partially	VERILOG code is Simulated & demonstrated on target device with some errors. Partially explained the code and output.	VERILOG code is Simulated but unable to demonstrated on target device & unable to explain the code and output.
Criteria-3 : Documentation			
The Code, Test bench, RTL, Simulation waveform and XDC file is extremely well documented.	The Code, Test bench, RTL, Simulation waveform and XDC file is reasonably well documented. There are minor formatting omissions that would have improved user understanding of code purpose.	The Code, Test bench, RTL, Simulation waveform and XDC file is poorly documented. There are no proper comments /titles given.	one or more documents from Code, Test bench, RTL, Simulation waveform and XDC file are missing. There are no proper comments /titles given. Or copied.

Experiment No. 4 **Date:**

Objectives: To design of 4-bit counters in

Verilog/VERILOG. Software: Vivado Software

Design Description

Binary Counter:

Design a 4 bit loadable binary Up-Down counter with asynchronous reset as shown below.

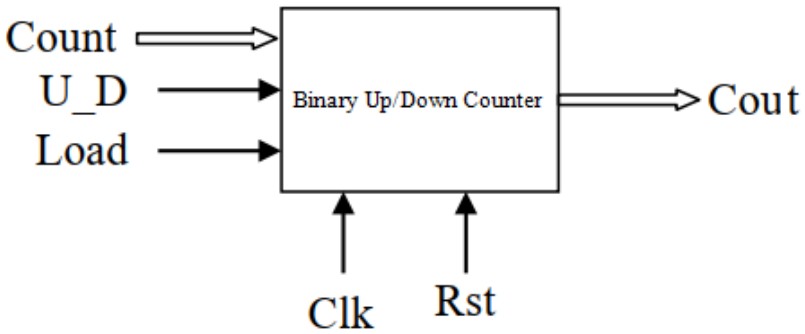


Fig. 4.1 Binary Up/Down Counter

Rst	Clk	Load	U_D	Count(t)	Operation
1	X	X	X	0000	Reset Counter
0	Positive edge	1	X	Din(t - 1)	Load data into the counter
0	Positive edge	0	1	Count(t-1) + 1	Up counting
0	Positive edge	0	0	Count(t-1) - 1	Down counting

Function Table of Binary Counter

A 4-bit binary up/down counter counts sequence from 0000 to 1111 and 1111 to 0000. The state table is given in above Table and the logic diagram in fig 4.1 The circuit operation can be explained as follows:

When the external input UP is equal to 1, no matter what the DOWN input is, the circuit operates as an UP counter and counts sequence from 0000 to 1111.

When the external input DOWN is equal to 1 and UP is equal to 0, the circuit operates as a DOWN counter and counts sequence from 1111 to 0000.

If both the inputs UP and DOWN are equal to 0, then the output of the flip-flop remains unchanged.

Procedure:

Step 1: Start the Vivado IDE by using the desktop shortcut or by using the Start ➔ Programs ➔ Vivado IDE.

Step 2: Create New Project➔ Select RTL Project ➔ Click on Next ➔ Choose the Board, - Vendor and Processor ➔ click on Next ➔ Finish

Step 3: In the Flow Navigator window ➔ Project Manager➔ Add Sources➔ Add or Create Design Sources➔ Create File ➔ VHDL /VERILOG➔ Define IN and OUTs ➔ Finish.

Step 4: Write the Code Editorial window and save it. Add Sources ➔ Add or Create Simulation Sources➔ Finish➔ Run simulation.

Step 5: Check the RTL Schematic of the source code. Flow Navigator window➔ RTL Analysis➔ Open Elaborated Design.

Step 6: Generate Bit Stream ➔ Open Hardware Target ➔ Next ➔ Select Target hardware and hardware device ➔ Next ➔ Finish.

Step 7: Click on Program Device➔ Program. Finished

Conclusion:

Rubrics for Evaluation:

Very good (5 Marks)	**Good (4 Marks)**	**Average (3 Marks)**	**Satisfactory (1 to 2 Marks)**
Criteria-1 :Functionality/Specifications			
The code is completely functional and responds correctly producing the correct outputs and or responses under all test cases	The code is mostly functional and responds correctly producing the correct outputs and or responses under most test cases. There are minor problems with the program implementation.	The code is marginally functional with numerous errors. The code may respond correctly under certain circumstances, but there are significant errors and/or incomplete code sections.	The code is minimally functional with significant portions of the code missing or incomplete. The code is largely nonresponsive to most test cases and/or inputs. Or copied.
Criteria-2 : Design Demonstration			
VERILOG code is executed. Simulated & demonstrated on target device. Explained the code and output fluently	VERILOG code is executed. Simulated & demonstrated on target device Explained the code and output partially	VERILOG code is Simulated & demonstrated on target device with some errors. Partially explained the code and output.	VERILOG code is Simulated but unable to demonstrated on target device & unable to explain the code and output.
Criteria-3 : Documentation			
The Code, Test bench, RTL, Simulation waveform and XDC file is extremely well documented.	The Code, Test bench, RTL, Simulation waveform and XDC file is reasonably well documented. There are minor formatting omissions that would have improved user understanding of code purpose.	The Code, Test bench, RTL, Simulation waveform and XDC file is poorly documented. There are no proper comments /titles given.	one or more documents from Code, Test bench, RTL, Simulation waveform and XDC file are missing. There are no proper comments /titles given. Or copied.

Experiment No. 5 **Date:**

Objectives: Finite State Machine (Sequence Detector implementation)

Software: Vivado Software

Design Description: -

Finite State Machines (FSMs) are a useful abstraction for sequential circuits with centralized "states" of operation at each clock edge, combinational logic computes outputs and next state as a function of inputs and present state a state is a description of the status of a system that is waiting to execute a transition. A transition is a set of actions to be executed when a condition is fulfilled or when an event is received.

There are two types of finite state machines that generate output

- Mealy Machine
- Moore machine

Mealy Machine:-

Output depends both upon the present state and the present input. Generally, it has fewer states than Moore Machine.
The value of the output function is a function of the transitions and the changes, when the input logic on the present state is done. Mealy machines react faster to inputs. They generally react in the same clock cycle.

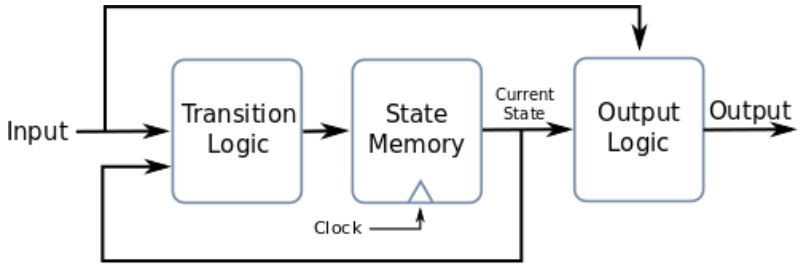


Fig. 5.1 Mealy Machine

Moore Machine:-

Output depends only upon the present state. Generally, it has more states than Mealy Machine. The value of the output function is a function of the current state and the changes at the clock edges, whenever state changes occur. In Moore machines, more logic is required to decode the outputs resulting in more circuit delays. They generally react one clock cycle later.

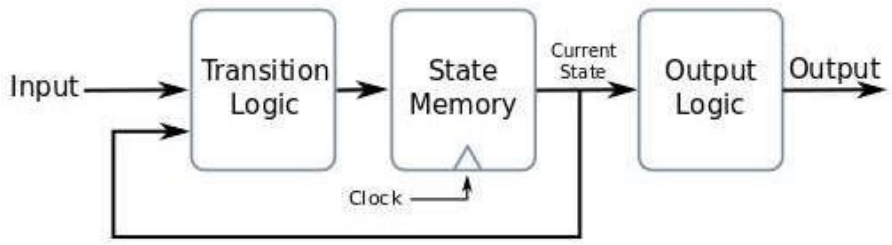


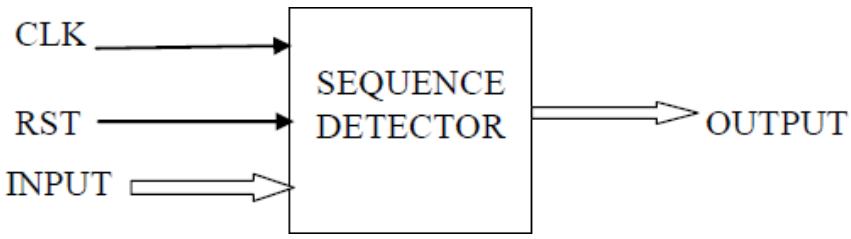
Fig. 5.2 Moore Machine

A sequence detector is a sequential state machine which takes an input string of bits and generates an output 1 whenever the target sequence has been detected. In a Mealy machine, output depends on the present state and the external input (x). Hence in the diagram, the output is written outside the states, along with inputs. Sequence detector is of two types:

1. Overlapping
2. Non-Overlapping

In an overlapping sequence detector the last bit of one sequence becomes the first bit of next sequence. However, in non-overlapping sequence detector the last bit of one sequence does not become the first bit of next sequence.

Block Diagram:



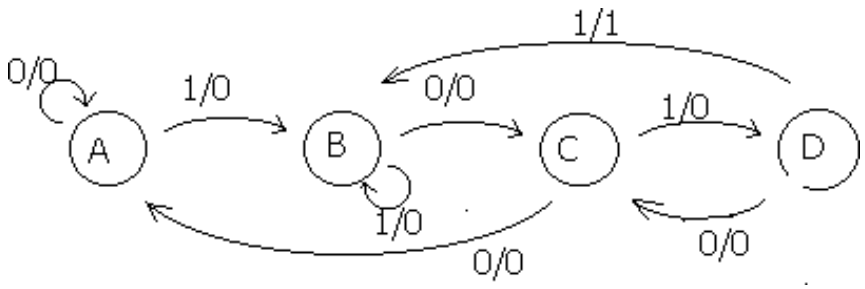


Fig. 5.3 State diagram for "1011" Mealy sequence detector

State Table :

INPUT STATES	OUTPUT STATES/OUTPUT	
	X=0	X=1
A	A/0	B/0
B	C/0	B/0
C	A/0	D/0
D	C/0	B/1

Procedure:

Step 1: Start the Vivado IDE by using the desktop shortcut or by using the Start ➔ Programs ➔ Vivado IDE.

Step 2: Create New Project➔ Select RTL Project ➔ Click on Next ➔ Choose the Board, Vendor and Processor ➔ click on Next ➔ Finish

Step 3: In the Flow Navigator window ➔ Project Manager➔ Add Sources➔ Add or Create Design Sources➔ Create File ➔ VHDL /VERILOG➔ Define IN and OUTs ➔ Finish.

Step 4: Write the Code Editorial window and save it. Add Sources ➔ Add or Create Simulation Sources➔ Finish➔ Run simulation.

Step 5: Check the RTL Schematic of the source code. Flow Navigator window➔ RTL Analysis➔ Open Elaborated Design.

Step 6: Generate Bit Stream ➔ Open Hardware Target ➔ Next ➔ Select Target hardware and hardware device ➔ Next ➔ Finish.

Step 7: Click on Program Device➔ Program. Finished

Conclusion:

Rubrics for Evaluation:

Very good (5 Marks)	**Good (4 Marks)**	**Average (3 Marks)**	**Satisfactory (1 to 2 Marks)**
Criteria-1 :Functionality/Specifications			
The code is completely functional and responds correctly producing the correct outputs and or responses under all test cases	The code is mostly functional and responds correctly producing the correct outputs and or responses under most test cases. There are minor problems with the program implementation.	The code is marginally functional with numerous errors. The code may respond correctly under certain circumstances, but there are significant errors and/or incomplete code sections.	The code is minimally functional with significant portions of the code missing or incomplete. The code is largely nonresponsive to most test cases and/or inputs. Or copied.
Criteria-2 : Design Demonstration			
VERILOG code is executed. Simulated & demonstrated on target device. Explained the code and output fluently	VERILOG code is executed. Simulated & demonstrated on target device Explained the code and output partially	VERILOG code is Simulated & demonstrated on target device with some errors. Partially explained the code and output.	VERILOG code is Simulated but unable to demonstrated on target device & unable to explain the code and output.
Criteria-3 : Documentation			
The Code, Test bench, RTL, Simulation waveform and XDC file is extremely well documented.	The Code, Test bench, RTL, Simulation waveform and XDC file is reasonably well documented. There are minor formatting omissions that would have improved user understanding of code purpose.	The Code, Test bench, RTL, Simulation waveform and XDC file is poorly documented. There are no proper comments /titles given.	one or more documents from Code, Test bench, RTL, Simulation waveform and XDC file are missing. There are no proper comments /titles given. Or copied.