NEW HORIZON
COLLEGE OF ENGINEERING

Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade.

MINI PROJECT

REPORT ON

"IMPLEMENTATION OF MOORE FSM SEQUENCE DETECTOR USING VHDL CODE"

SUBMITTED BY:

MANISH M [1NH18EC066]

SOUVIK MANNA [1NH18EC107]

CHAITANYA REDDY [1NH18EC065]

VAMSI KRISHNA [1NH18EC083]

Under the guidance of

Dr. Mohan Kumar Naik. B

Professor, Dept. of ECE, NHCE, Bengaluru.

NEW HORIZON COLLEGE OF ENGINEERING

(ISO-9001:2000 certified, Accredited by NAAC 'A', Autonomous college permanently affiliated to VTU)Outer Ring Road, Panathur Post, Near Marathalli, Bengaluru-560103

NEW HORIZON COLLEGE OF ENGINEERING

DEPARTMENT OF COMMUNICATION

ELECTRONICS AND ENGINEERING

CERTIFICATE

Certified that the mini project work entitled "**IMPLEMENTATION OF MOORE FSM SEQUENCE DETECTOR USING VHDL CODE**" carried out by **MANISH M** [1NH18EC066],**SOUVIK MANNA** [1NH18EC107],**CHAITANYA REDDY** [1NH18EC065],**VAMSI KRISHNA** [1NH18EC083] bonafide students of Electronics and Communication Department , New Horizon College of Engineering, Bangalore.

The mini project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for the said degree.

Project Guide

HOD ECE

External Viva

Name of Examiner

Signature with Date

1.

2.

ACKNOWLEDGEMENT

The satisfaction that accompany the successful completion of any task would be, but impossible without the mention of the people who made it possible, whose constant guidance and encouragement helped us succeed.

We thank **Dr. Mohan Manghnani**, Chairman of **New Horizon Educational Institution**, for providing necessary infrastructure and creating good environment.

We also record here the constant encouragement and facilities extended to us by **Dr.Manjunatha**, Principal, NHCE and **Dr. Sanjeev Sharma**, head of the department of Electronics and Communication Engineering. We extend sincere gratitude to them.

We sincerely acknowledge the encouragement, timely help and guidance to us by our beloved guide **Dr. Mohan Kumar Naik. B** to complete the project within stipulated time successfully.

Finally, a note of thanks to the teaching and non-teaching staff of electronics and communication department for their co-operation extended to us, who helped us directly or indirectly in this successful completion of mini project.

MANISH M [1NH18EC066]

SOUVIK MANNA [1NH18EC107]

CHAITANYA REDDY [1NH18EC065]

VAMSI KRISHNA [1NH18EC083]

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

With the recent technological advances, modern society is increasingly dependent on automatic machines. To deal with their busy lives. Modern automatic machines coordinate a series of actions depending on the environment and the event.

FSM (Finite State Machine) is used to mathematically represent these sequences of actions or instructions. This article describes two FSM machine types, Moore and Mealy. Various results are shown to show the importance of FSM modeling. The edge detection circuit is designed using both Moore and Mealy machines. This is an FSM design example and can be used to build and demonstrate student concepts. These designs are implemented in VHDL. Comparisons are also made based on both implementations.A state machine is a machine that detects the transition from one state to anotherMany times. Moore FSM for this mini project

The sequence is retrieved with a sequence like "1001". Moore FSM output is

Current state of FSM. Therefore, the current state and the next state are recorded. This Vhdl

The miniproject shows the detection of sequences using Vhdl code.

Keywords-FSM;, VHDL;Xilinx-ISE;

CHAPTER 01

INTRODUCTION

Digital systems have two basic types of circuits. The first type is a combinational logic circuit. In combinatorial logic, the output depends only on the input. Examples of combinatorial logic circuits include adders, encoders, and multiplexers.

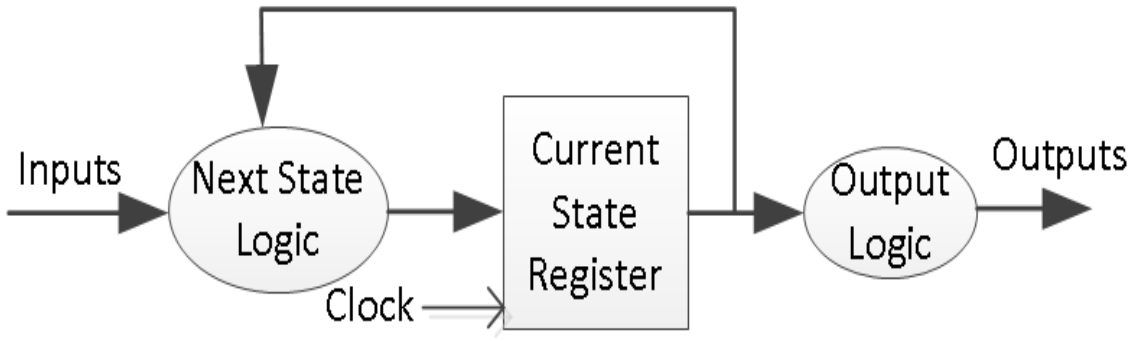
The second type of digital logic circuit is a sequential logic circuit. In sequential logic circuits, the output depends not only on the input but also on the current state of the system

The Sequential Logic System is a Finite State Machine (FSM).

Like FSM, they consist of a set of states, some inputs, some outputs, and a set of rules for moving from one state to another

When designing a digital system, it is quite common to start out by specifying how the system works with the limited-case machine model.

A general model of the Moore Sequential Machine is shown below. It is produced from the state registration block. The next state is determined by the current (current) and current (current) status. Here the case model is also recorded using D flip-flops. Moore machines are usually described using three blocks, one of which must be serialized and the other two models can always be designed with blocks or a set of data flow modeling groups always.



HDL(Hardware Description Language)

In computer engineering, HDL is a specialized computer language used to describe the structure and behavior of electronic circuits, and the most common digital logic circuits.

The device description language provides an accurate and formal description of an electronic circuit that enables automated analysis and simulation of an electronic circuit. It also allows the synthesis of HDL descriptions in the netlist (specifications for electronic electronic components and how to connect them together), You can then set and instruct to generate the set of masks used to create the integrated circuit .

Hardware description languages are very similar to programming languages such as C and ALGOL. A textual description consisting of expressions, data, and control structures . One of the main programming languages and HDL is that HDL explicitly includes the notion of time.
HDL is used to describe a hardware executable specification. Programs that implement the basic semantics of language statements and are designed to simulate the progression of time provide the ability for hardware designers to model hardware before physically creating it. It is this workability that gives the illusion that HDL is a programming language when it is more accurately classified as a specification or modeling language. There are simulators that can support discrete event (digital) and continuous time (analog) modeling, and HDLs for each are available.
The first hardware description languages appeared more like traditional languages in the late 1960s. The first to have a lasting effect was in 1971 c. Described in Gordon Bell and Alan Newell's Text Computer Structures. This text introduced the concept of register transfer levels. First used in the ISP language and described the operation of the Digital Equipment Corporation (DEC) PDP-8.

This language became even more popular with the introduction of DEC's PDP-16 RT Level Module (RTM) and a book explaining its use. At least two implementations of the basic ISP languages (ISPL and ISPS) followed. ISPS is well suited to describe the relationship between

design inputs and p-outputs and has been adopted by the DESC commercial team and many research teams in the US and NATO allies.

Due to the increasing popularity of new technologies, especially Very Large Scale Integration (VLSI), RTM products have never been commercially popular and DEC discontinued their marketing in the mid-1980s.

Most modern digital circuit designs revolve around it as a result of the efficiency gains achieved using HDL. Most designs begin with a set of requirements or a high-level architectural diagram. Control and decision models are often created in flowchart applications or incorporated into state diagram editors. The process of writing an HDL description largely depends on the designer's preference for the nature of the circuit and the coding style.

HDL is simply a "capture language" and often starts with a high level algorithmic description such as a C ++ mathematical model. Designers often use scripting languages like Perl to automatically create redundant circuit structures in HDL.

A special text editor provides features for automatic indentation, syntax-dependent coloring, and macro-based extension of entity / architecture / signal declarations.

The HDL code then undergoes a code review or audit. In preparation for synthesis, the HDL description is subject to a series of automatic checkers.
Checker reports deviations from standard code guidelines, identifies ambiguous code constructs before misleading them, and checks for common logic coding errors such as floating ports or short p outputs. This process helps fix errors before code is synced.

The ability to mimic HDL programs is essential to HDL design.HDL description simulation allows design verification to be presented, an important feature that checks design intent (specifications), in contrast to implementing code in HDL description. Architectural exploration is also allowed.

Engineers can test design options by writing multiple forms of basic design and comparing their behavior with simulation, and simulation is critical to successful HDL design.

To simulate an HDL model, a high-level simulation environment engineer (called a test bench) writes. At a minimum, the test table contains an instant copy of the model (called the device under test or DUT), pin / signal announcements for model I / O, and a waveform around the clock. Event-driven testbench code: The engineer writes HDL statements to implement the reset signal (generated by testbench), to model interface parameters (such as host / bus read / write), and to monitor DUT output.

The HDL emulator - the test stand test program - maintains the simulation time, which is the main reference for all events in the test stand simulation.

Events occur only in situations dictated by testLench HDL (such as an encrypted recovery key in the test table) or in a reaction (depending on the model) to stimulate and trigger events. Modern HDL emulators have complete graphical user interfaces, complete with a set of debugging tools. This allows the user to pause and restart simulations at any time, insert emulator breakpoints (regardless of the HDL code) and monitor or modify any element in the HDL hierarchy.

Modern emulators can also link the HDL environment to user-collected libraries, via a specific PLI / VHPI interface. System-based connection (x86, SPARC, etc. running on Windows / Linux / Solaris), where HDL emulator and user libraries are collected and linked outside HDL environment.

HDL is very similar to programming language, but there are fundamental differences. Most programming languages are procedural (single-threaded), with grammar support and limited competition management indicators. On the other hand, HDLs are similar to concurrent programming languages in their ability to model multiple parallel processes (such as slippers and plug-ins) that work automatically independently of each other. Any changes to the process entry update the emulator process stack automatically.

Any changes to the process entry update the emulator process stack automatically.

Both programming languages and HDL are handled by an interpreter (often called an HDL complex), but with different goals. For HDL, the term “classification” refers to logical code; the process of converting an HDL code list to a netlist is actually achievable. The netlist output can take any of several forms: netlist "simulation" with gate delay information, netlist "handoff" for post-installation and routing on a semiconductor template, or general standard electronic design exchange (EDIF) format (for subsequent conversion to a file in JEDEC format) .

Meanwhile, the compiler translates the source code listing into microprocessor object code for execution on the target microprocessor. Since HDL and programming languages borrow concepts and functions from each other, their boundaries are not clear. However, HDL is not suitable for developing general-purpose application software . [Rationale] Like general-purpose programming languages, it is not desirable for device modeling .

However, as electronic systems become more complex and reconfigurable, the industry demands a single language that can perform both hardware design and software programming tasks. I. System C is one example. Embedded system hardware can be modeled as less detailed architectural blocks. That is, a black box with modeled signal input and output drivers. The target application is written in C or C ++ and is natively compiled for the host development system. Instead of targeting an embedded CPU, you need a host simulation of the embedded or emulated CPU.

The high-level summary of the SystemC model is well-suited for exploring early architecture, where architectural changes can be easily assessed with little attention to signal level implementation problems. However, the correlation model used by System C depends on shared memory, so the parallel execution and low-level models cannot properly handle the language.

CHAPTER 02

LITERATURE SURVEY

Moore State Machine is designed with sequential detection design Sequence "1001". This was implemented using J-K flipflop using VHDL code . Later The same sequence detector was implemented using Mealy FSM Implemented on Xilinx platform whose code was also written in VHDL . An Implement Moore Sequence Detector using all Sequences and Fusion Circles executed.

Finite state machine hardware book

A comprehensive guide to hardware limited case theory and design in hardware, with design examples developed in VHDL and SystemVerilog. Modern and evolving digital systems always include state-of-the-art devices. Correct design of these parts is critical to achieving proper system performance. This book provides detailed and comprehensive theoretical and comprehensive coverage of any class of limited-state devices that are implemented by devices.

It describes the serious design issues that lead to incorrect or suboptimal implementations and gives examples of finite state machines developed in VHDL and the SystemVerilog hardware description language (the successor to Verilog).

Important features include: a comprehensive review of serial circuit design practices; a new division of all case devices into three device-based categories, covering all possible situations, with many practical examples provided in all three categories; display of complete designs, with detailed VHDL and SystemVerilog codes , Comments and simulation results are all tested on FPGAs; exercise sampling, which can be synthesized, simulated, and actually implemented on FPGAs.
Additional resources are available on the book's website. Hardware design is more complicated than software design. Although interest in the material of machines in limited conditions has

grown significantly in recent years, there is no comprehensive cure for this topic. This book provides the most detailed coverage of limited case machines available.

RTL HARDWARE DESIGN USING VHDL BOOK

This book serves as an introduction to VHDL for beginners and a resource for the most advanced designers. I find it particularly useful for beginners because it focuses on the synthesizable subset of VHDL, while many other books are delusional about the "behavioral characteristics" of VHDL, which makes it seem like a bad idea. 'a general programming language rather than what it really is: a pretty good way to describe hardware. What I liked best about this book is its good approach to robust synchronous encoding, which is very important if you want to use VHDL that is synthesized on successful hardware. I didn't like the author's style of implementing FSM, but other than that the code is very well written. The last chapter on synchronization and data transfer between clock domains is amazing and worth the price of the book alone. Of all the VHDL books I've read or flipped through, this is just the second good book. The other is Peter Ashenden's "Guide to the VHDL Designer" (of which we now have two working copies), which the author often mentions as reference text. So if VHDL is important to you, I highly recommend this book.

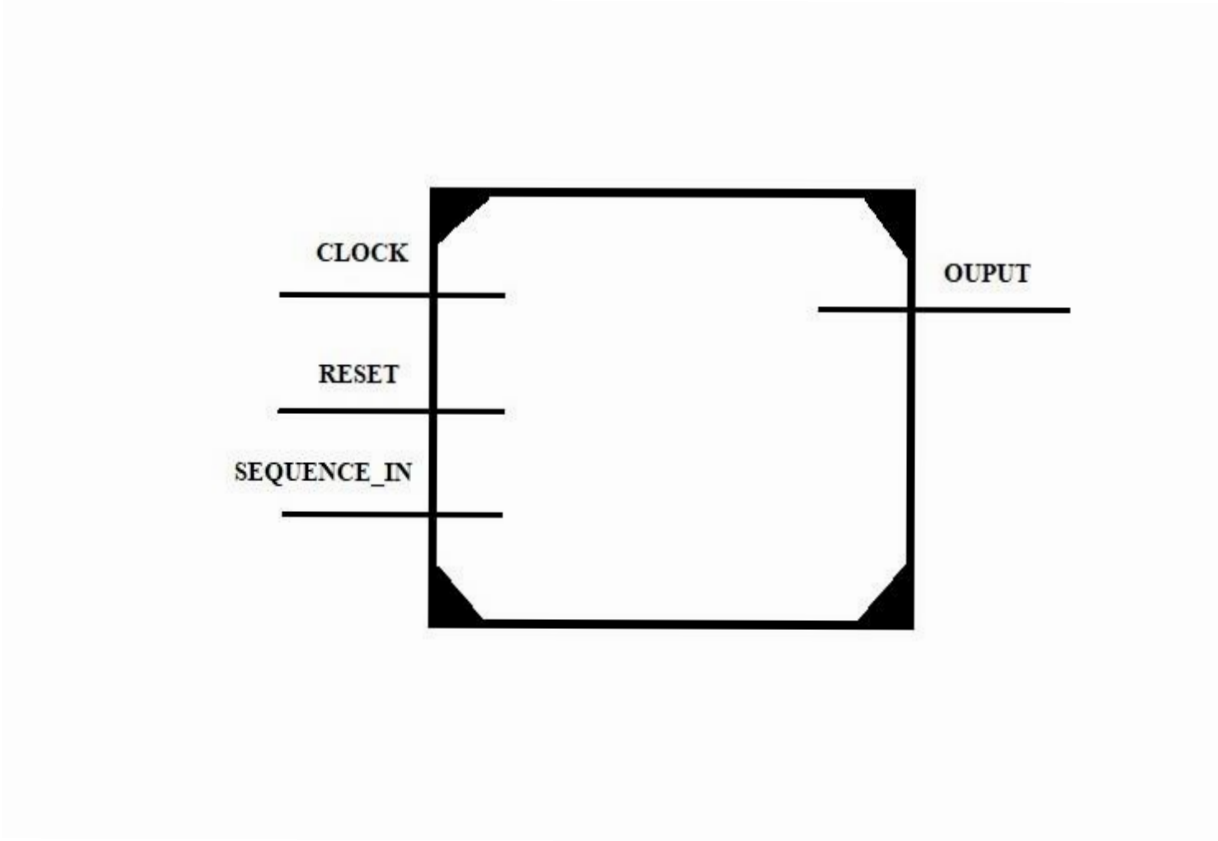
1.1: LITERATURE REVIEW

Title of the paper	Author & Year of Publication	Outcome	Limitation
Finite State Machines : Theory and Design using VHDL	Volnei A. Pedroni Published at 20 December 2013	A comprehensive review of design practices for serial digital circuits	View full layouts, with detailed VHDL and SystemVerilog icons, comments, and simulation results
RTL Hardware Design Using VHDL	Pong Chu &Published at 2005	The hardware-based category contains all possible positions and many practical examples are provided in all three categories.	Examples of FPGA equipment and exercises. All of these can be synthesized, simulated and physically implemented on an FPGA board.

CHAPTER 03

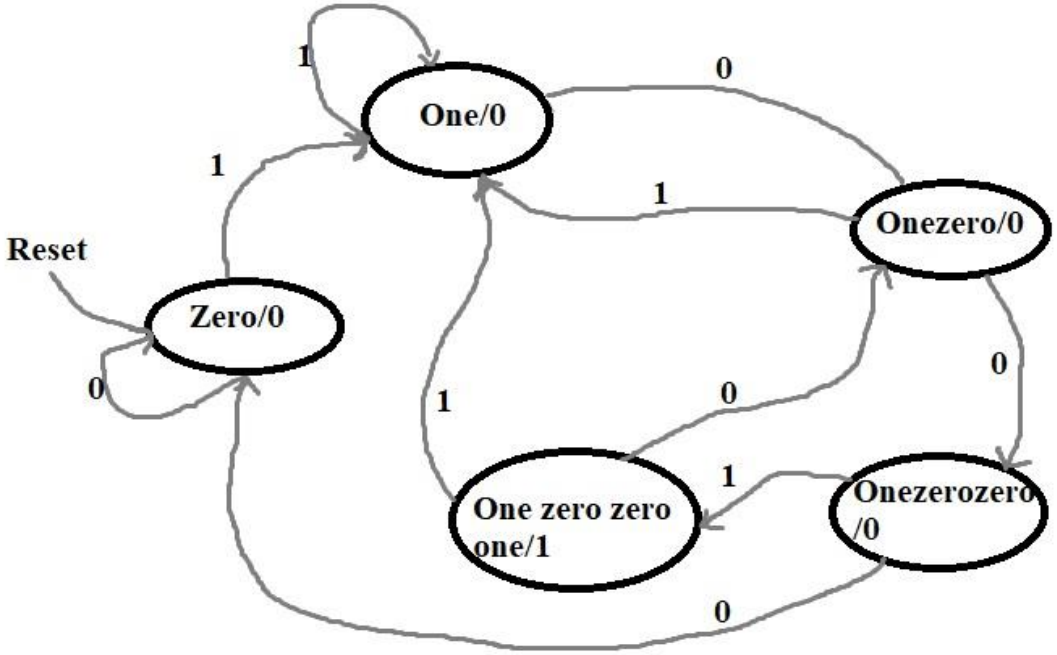
PROPOSED METHODOLOGY

Block Diagram



1.1:Block diagram of Moore Sequence

Flow Chart



1.2:Flow Chart of FSM Moore Sequence

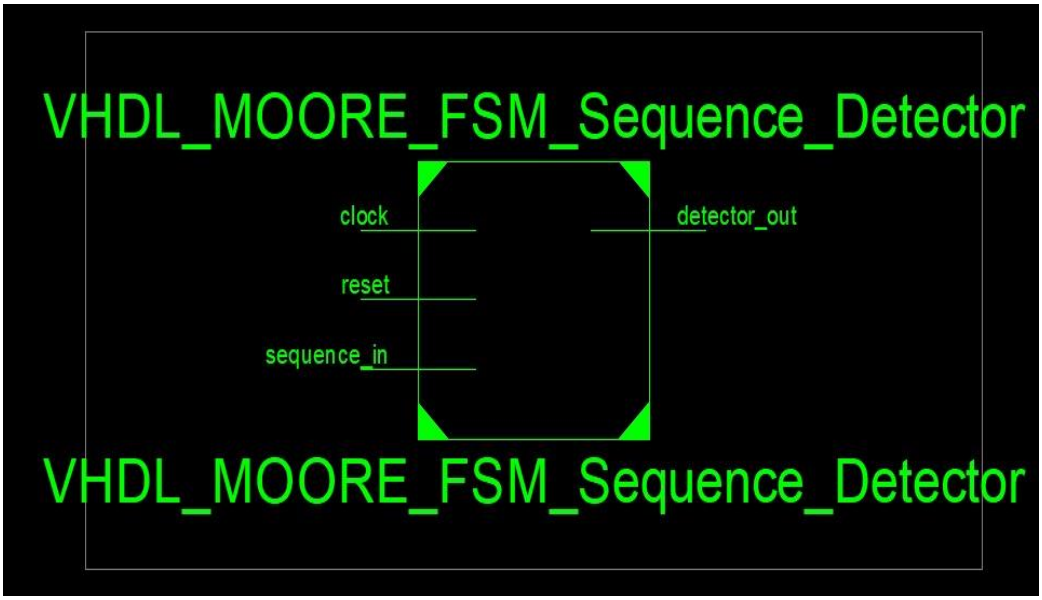
CHAPTER 04

PROJECT DESCRIPTION

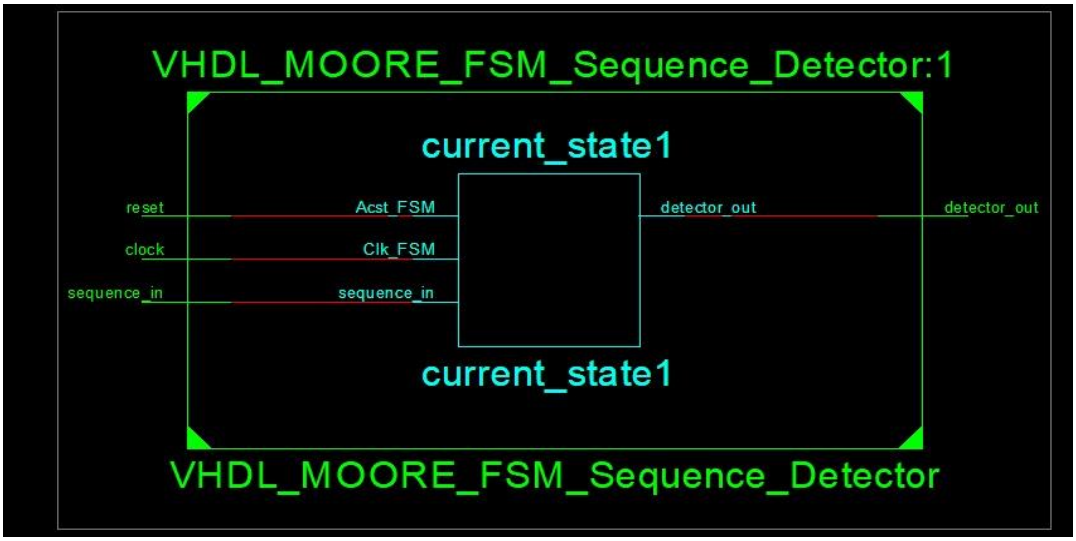
Software description

XIILINX ISE 7.1

- Xilinx ISE (Integrated Control Environment) is a software tool developed by Xilinx.
- Used to compile and analyze HDL design, allow developers to synthesize a design ("aggregate"), perform time analysis, validate RTL graphs, and simulate design response to various catalysts.
- Used to configure the programmer and target device
- Xilinx ISE is the design environment for Xilinx FPGA products.
- It is tightly coupled to structure of these chips and cannot use in FPGA products from other vendors.
- Xilinx ISE is primarily used for tuning and circuit design, and ISIM or ModelSim logic simulators are used for system level testing.
  Embedded Development Kit (EDK), Software Development Kit (SDK) and Chipscope Pro are the other components included in the Cylinks ISE.
- Since 2012, Xilinx ISE has been discontinued in favor of the Vivado Design Suite. It covers the same role as ISE with additional system features in chip development. Xilinx released the latest version of ISE (version 14.7) in October 2013, stating, "ISE is in the sustainability phase of the product lifecycle, no other ISE version is planned.



(1.3)RTL SCHEMATIC



(1.4)TECHNOLOGY SCHEMATIC

Hardware description

The hardware component we use here is a FPGA (**Field Programmable Gate Array)**
The FPGA can be reprogrammed after manufacturing to meet the desired application or functional requirements. This feature distinguishes FPGAs from custom-made application-specific integrated circuits (ASICs) for specific design tasks.

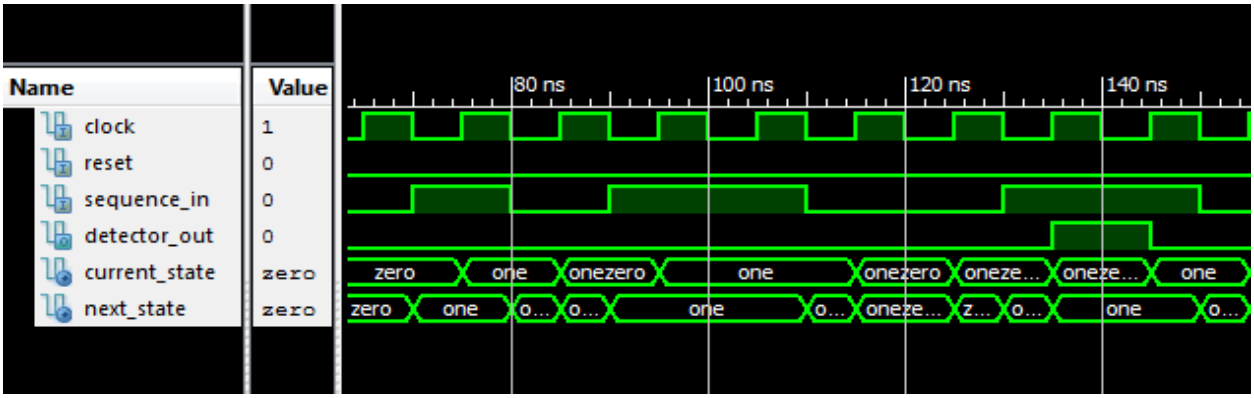
One-time programmable (OTP) FPGAs are available, but the main type is SRAM-based and can be reprogrammed as the design evolves.

FPGA Stands for Field Programmable Gate Array consists of a set of logic gates that store the specified digital logic from the hardware description language. As the name implies, it is a programmable field without the need to change new devices. FPGA is mostly used to model any digital logical circuit before it is manufactured as ASIC. FPGA is made up of SRAM cells, so we need to configure it at run time. For this purpose we can use flash memory with different types of interfaces. FPGA consists of configurable logic blocks (CLB), BRAM, DCM (digital clock manager), multiplexer, processor (optional) and many solid IP address cores. FPGA IC is mostly manufactured by Xilinx, Altera, Lattice and Microsemi.

In the past, FPGAs have chosen designs with speed / complexity / low volume, but the current FPGA easily pushes the 500MHz performance barrier. With an unprecedented increase in logical density and other features such as integrated processors, DSP blocks, recordings and high-speed series, at low price points, FPGA is a great display for any type of design.

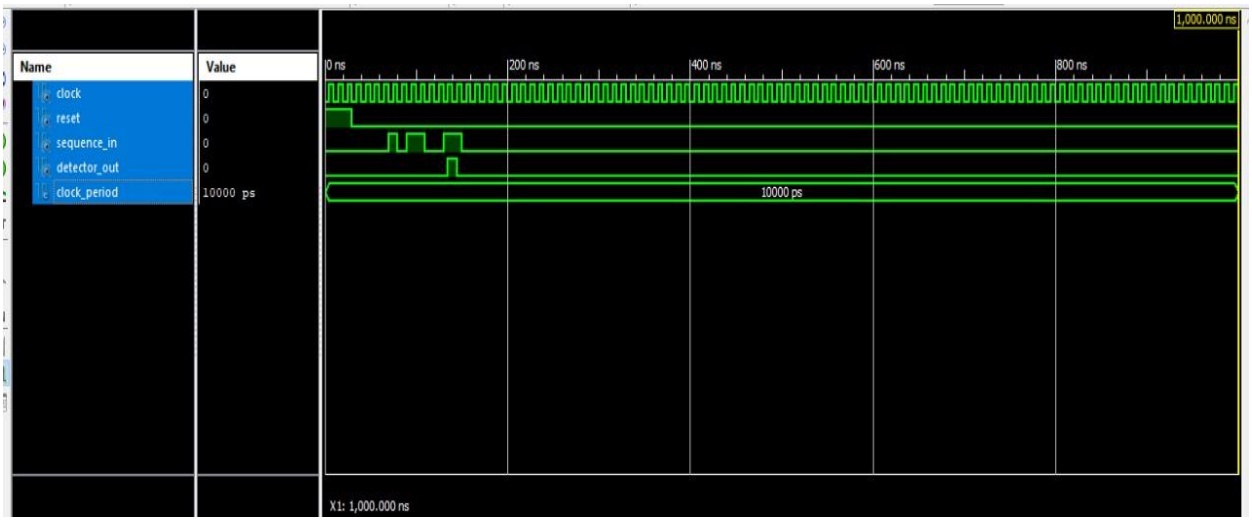
CHAPTER 05

RESULT



(1.5)Output Waveform

It takes an input string of bits and generates an output 1 whenever the target sequence has been detected.



(1.6)Output

In a Moore machine, output depends on the present state and the input.

CHAPTER 06

CONSLUSION

This VHDL project provides a complete VHDL code to discover Moore FSM sequence. VHDL emulation is also available. The sequence to be discovered is "1001"

The required code is been written and executed. The code has been executed and the output has been obtained. The output graphs have been attached in this file.

REFERENCES

[1]D. W. Bliss, P. A. Parker, A. R. Margetts, "Simultaneous transmission and reception for improved wireless network performance", *Statistical Signal Processing 2007 IEEE/SP 14th Workshop on*, pp. 478-482, 2007.

[2] https://www.fpga4student.com/2017/09/vhdl-code-for-moore-fsm-sequence-detector.html?m=1

[3] https://ieeexplore.ieee.org/document/8124583

[4] http://yue-guo.com/2018/11/18/sequence-detector-1001-moore-machine-mealy-machine-overlapping-non-overlapping/

[5] V. Solov'ev and I. Bulatova, "Synthesis of D-Class Finite State Machines on Programmable Logic Devices," in *Proc. of the 4th Int. Conf. on Computer-Aided Design of Discrete Devices (CAD DD'01), Minsk, Belarus, 2001*, Vol. 2, pp. 6–13.

z

APPENDIX

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity VHDL_MOORE_FSM_Sequence_Detector is
port ( clock: in std_logic; --- clock signal
 reset: in std_logic; -- reset input
 sequence_in: in std_logic; -- binary sequence input
 detector_out: out std_logic -- output of the VHDL sequence detector
);
end VHDL_MOORE_FSM_Sequence_Detector;

architecture Behavioral of VHDL_MOORE_FSM_Sequence_Detector is
type MOORE_FSM is (Zero, One, OneZero, OneZeroZero, OneZeroZeroOne);
signal current_state, next_state: MOORE_FSM;

begin
-- Sequential memory of the VHDL MOORE FSM Sequence Detector
process(clock,reset)
begin
 if(reset='1') then
```

```
 current_state <= Zero;
elsif(rising_edge(clock)) then
 current_state <= next_state;
end if;
end process;
-- Next state logic of the VHDL MOORE FSM Sequence Detector
-- Combinational logic
process(current_state,sequence_in)
begin
case(current_state) is
when Zero =>  if(sequence_in='1') then
  -- "1"
  next_state <= One;
 else
  next_state <= Zero;
 end if;
when One =>
 if(sequence_in='0') then
  -- "10"
  next_state <= OneZero;
 else
  next_state <= One;
```

```
 end if;
when OneZero =>
 if(sequence_in='0') then
  -- "100"
  next_state <= OneZeroZero;
 else
  next_state <= One;
 end if;
when OneZeroZero =>
 if(sequence_in='1') then
  -- "1001"
  next_state <= OneZeroZeroOne;
 else
  next_state <= Zero;
 end if;
when OneZeroZeroOne =>
 if(sequence_in='1') then
  next_state <= One;
 else
  next_state <= OneZero;
 end if;
end case;
```

```
end process;
-- Output logic of the VHDL MOORE FSM Sequence Detector
process(current_state)
begin
 case current_state is
 when Zero =>
  detector_out <= '0';
 when One =>
  detector_out <= '0';
 when OneZero =>
  detector_out <= '0';
 when OneZeroZero =>
  detector_out <= '0';
 when OneZeroZeroOne =>
  detector_out <= '1';
 end case;
end process;
end Behavioral;
```

Testbench

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;


ENTITY tb_VHDL_Moore_FSM_Sequence_Detector IS

END tb_VHDL_Moore_FSM_Sequence_Detector;


ARCHITECTURE behavior OF tb_VHDL_Moore_FSM_Sequence_Detector IS


  -- Component Declaration for the Moore FSM Sequence Detector in VHDL


  COMPONENT VHDL_MOORE_FSM_Sequence_Detector
  PORT(clock : IN  std_logic;
       reset : IN  std_logic;
       sequence_in : IN  std_logic;
       detector_out : OUT  std_logic
      );
  END COMPONENT;
```

z


```
  --Inputs
  signal clock : std_logic := '0';
  signal reset : std_logic := '0';
  signal sequence_in : std_logic := '0';

 --Outputs
  signal detector_out : std_logic;

  -- Clock period definitions
  constant clock_period : time := 10 ns;

BEGIN

 -- Instantiate the Moore FSM Sequence Detector in VHDL
  uut: VHDL_MOORE_FSM_Sequence_Detector PORT MAP (
        clock => clock,
        reset => reset,
        sequence_in => sequence_in,
        detector_out => detector_out
       );
```

z


```
 -- Clock process definitions
 clock_process :process
 begin
clock <= '0';
wait for clock_period/2;
clock <= '1';
wait for clock_period/2;
 end process;


-- Stimulus process
 stim_proc: process
 begin
    -- hold reset state for 100 ns.
sequence_in <= '0';
reset <= '1';
-- Wait 100 ns for global reset to finish
wait for 30 ns;
    reset <= '0';
wait for 40 ns;
sequence_in <= '1';
wait for 10 ns;
sequence_in <= '0';
```

z


```
 wait for 10 ns;

 sequence_in <= '1';

 wait for 20 ns;

 sequence_in <= '0';

 wait for 20 ns;

 sequence_in <= '1';

 wait for 20 ns;

 sequence_in <= '0';

    -- insert stimulus here

    wait;

  end process;


END;
```

A sequence recognizer is a special kind of sequential circuit that looks for a special bit pattern in some input

The recognizer circuit has only one input, X

One bit of input is supplied on every clock cycle

This is an easy way to permit arbitrarily long input sequences

There is one output, Z, which is 1 when the desired pattern is found

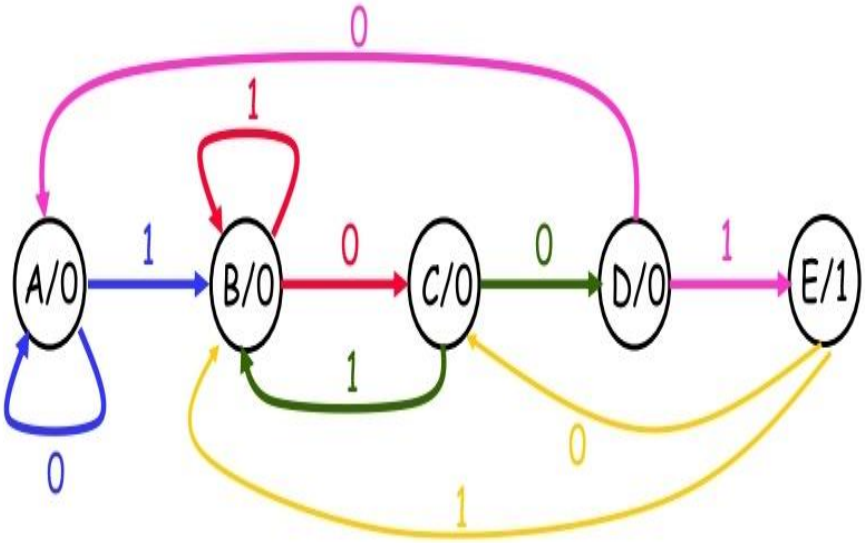
Our example will detect the bit pattern 1001:

Inputs: 1 1 1 001 1 0 1 00 1 001 1 0…

Outputs: 0 00 001 00 0 00 1 001 0 0…

• A sequential circuit is required because the circuit has to ―remember‖ the inputs from previous clock cycles, in order to determine whether or not a match was found

Moore state diagram & table



Present State	Input	Next State	Output
A	0	A	0
A	1	B	0
B	0	C	0
B	1	B	0
C	0	D	0
C	1	B	0
D	0	A	0
D	1	E	0
E	0	C	1
E	1	B	1

A: 000 D: 100
B: 001 E: 101
C: 010

Z

Q_0 \ Q_2Q_1	00	01	11	10
0				
1				1

$Z = Q_2 Q_1' Q_0$

25