ACS College of Engineering
Approved by AICTE New Delhi, Affiliated to VTU, Belagavi
(A Unit of RajaRajeswari Group of Institutions)

NAAC 'A' Accredited

Department of Electronics and Communication Engineering

HARDWARE DISCRIPTION LABORATORY MANUAL

Subject Code: 18ECL58

Prepared by
Mrs. Vijaya Dalawai
Assistant Professor, Dept. Of ECE

Dr. H B Bhuvaneswari
HOD, Dept. Of ECE

Affiliated to Visvesvaraya Technological

University, Belagavi, Karnataka - 590018

2020-21

CONTENTS

1. SYLLABUS

Subject code: 18ECL58 **IA marks: 40**

No. of practical Hrs. /week: 03 **Exam hours: 03**

Exam marks: 60

……

(ACCORDING TO VTU SYLLABUS)

PART – A

PROGRAMMING (using VHDL and Verilog)

1. Write Verilog program for the following combinational design along with test bench to verify the design:
   a. 2 to 4 decoder realization using NAND gates only (structural model)
   b. 8 to 3 encoder with priority and without priority (behavioural model)
   c. 8 to 1 multiplexer using case statement and if statements
   d. 4-bit binary to gray converter using 1-bit gray to binary converter 1-bit adder and subtractor

2. Model in Verilog for a full adder and add functionality to perform logical operations of XOR, XNOR, AND and OR gates. Write test bench with appropriate input patterns to verify the modeled behaviour.

3. Verilog 32-bit ALU shown in figure below and verify the functionality of ALU by selecting appropriate test patterns. The functionality of the ALU is presented in Table 1.
   a. Write test bench to verify the functionality of the ALU considering all possible input patterns
   b. The enable signal will set the output to required functions if enabled, if disabled all the outputs are set to tri-state
   c. The acknowledge signal is set high after every operation is completed

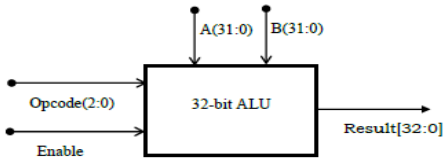


Figure 1 ALU top level block diagram

Table 1 ALU Functions

Opcode (2:0)	ALU Operation	Remarks	
000	A + B	Addition of two numbers	Both A and B are in two's complement format
001	A – B	Subtraction of two numbers	
010	A + 1	Increment Accumulator by 1	A is in two's complement format
011	A - 1	Decrement accumulator by 1	
100	A	True	Inputs can be in any format
101	A Complement	Complement	
110	A OR B	Logical OR	
111	A AND B	Logical AND	

4. Write Verilog code for SR, D and JK and verify the flip flop.

5. Write Verilog code for 4-bit BCD synchronous counter.

6. Write Verilog code for counter with given input clock and check whether it works as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.

PART - B

INTERFACING (at least four of the following must be covered using VHDL/ Verilog)

1. Write a Verilog code to design a clock divider circuit that generates 1/2, 1/3rd and 1/4thclock from a given input clock. Port the design to FPGA and validate the functionality through oscilloscope.
2. Interface a DC motor to FPGA and write Verilog code to change its speed and direction.
3. Interface a Stepper motor to FPGA and write Verilog code to control the Stepper motor rotation which in turn may control a Robotic Arm. External switches to be used for different controls like rotate the Stepper motor (i) +N steps if Switch no.1 of a Dip switch is closed (ii) +N/2 steps if Switch no. 2 of a Dip switch is closed (iii) –N steps if Switch no. 3 of a Dip switch is closed etc.
4. Interface a DAC to FPGA and write Verilog code to generate Sine wave of frequency F KHz (eg. 200 KHz) frequency. Modify the code to down sample the frequency to F/2 KHz. Display the Original and Down sampled signals by connecting them to an oscilloscope.
5. Write Verilog code using FSM to simulate elevator operation.
6. Write Verilog code to convert an analog input of a sensor to digital form and to display the same on a suitable display like set of simple LEDs, 7-segment display digits or LCD display.

2. OVERVIEW OF HDL LAB

2.1 HDL

In electronics, a hardware description language or HDL is any language from a class of Computer languages for formal description of electronic circuits. It can describe the circuit's operation, its design and organization, and tests to verify its operation by means of simulation

HDLs are standard text-based expressions of the spatial, temporal structure and behavior of electronic systems. In contrast to a software programming language, HDL syntax, semantics include explicit notations for expressing time and concurrency, which are the attributes of hardware. Languages whose only characteristic is to express circuit connectivity between a hierarchies of blocks are properly classified as net list languages.

HDLs are used to write executable specifications of some piece of hardware. A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time, provides the hardware designer with the ability to model a piece of hardware before it is created physically. It is this execute ability that gives HDLs the illusion of being programming languages. Simulators capable of supporting discrete-event and continuous-time (analog) modeling exist, and HDLs targeted for each are available.

It is certainly possible to represent hardware semantics using traditional programming languages such as C++, although to function such programs must be augmented with extensive and unwieldy class libraries. Primarily, however, software programming languages function as hardware description language

Using the proper subset of virtually any language, a software program called a synthesizer can infer hardware logic operations from the language statements and produce an equivalent net list of generic hardware primitives to implement the specified behavior. This typically requires the synthesize r to ignore the expression of any timing constructs in the text.

The two most widely-used and well-supported HDL varieties used in industry are

- VHDL (VHSICHDL)
- Verilog

2.2 VHDL

VHDL (Very High Speed Integrated Circuit Hardware Description Language) is commonly used as a design-entry language for field-programmable gate arrays and application-specific integrated circuits in electronic design automation of digital circuits.

VHDL is a fairly general-purpose language, and it doesn't require a simulator on which to run the code. There are a lot of VHDL compilers, which build executable binaries. It can read and write files on the host computer, so a VHDL program can be written that generates another VHDL program to be incorporated in the design being developed. Because of this general-purpose nature, it is possible to use VHDL to write a test bench that verifies with the user, and compares results with those expected. This is similar to the capabilities of the Verilog language

VHDL is not a case sensitive language. One can design hardware in a VHDL IDE (such as Xilinx or Quartus) to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software (such as Model Sim) which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test bench. To generate an appropriate test bench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required.

The key advantage of VHDL when used for systems design is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools

translate the design into real hardware (gates and wires). When a VHDL model is translated into the "gates and wires" that are mapped onto a programmable logic device such as a CPLD or FPGA, then it is the actual hardware being configured, rather than the VHDL code being "executed" as if on some form of a process or chip.

Both VHDL and Verilog emerged as the dominant HDLs in the electronics industry while older and less-capable HDLs gradually disappeared from use. But VHDL and Verilog share many of the same limitations: neither HDL is suitable for analog/mixed-signal circuit simulation. Neither possesses language constructs to describe recursively-generated logic structures.

2.3 Verilog

Verilog is a hardware description language (HDL) used to model electronic systems. The language supports the design, verification, and implementation of analog, digital, and mixed - signal circuits at various levels of abstraction

The designers of Verilog wanted a language with syntax similar to the C programming language so that it would be familiar to engineers and readily accepted. The language is case-sensitive, has a preprocessor like C, and the major control flow keywords, such as "if" and "while", are similar. The formatting mechanism in the printing routines and language operators and their precedence real so similar

The language differs in some fundamental ways. Verilog uses Begin/End instead of curly braces to define a block of code. The concept of time, so important to a HDL won't be found in C The language differs from a conventional programming language in that the execution of statements is not strictly sequential. A Verilog design consists of a hierarchy of modules are defined with a set of input, output, and bidirectional ports. Internally, a module contains a list of wires and registers. Concurrent and sequential statements define the behavior of the module by defining the relationships between the ports, wires, and registers Sequential statements are placed inside a begin/end block and executed in sequential order within the block. But all concurrent statements and all begin/end blocks in the design are executed in parallel, qualifying Verilog as a Dataflow language. A module can also contain one or more instances of another modul eto define sub-behavior

A subset of statements in the language is synthesizable. If the modules in a design contains a netlist that describes the basic components and connections to be implemented in hardware only synthesizable statements, software can be used to transform or synthesize the design into the net

list may then be transformed into, for example, a form describing the standard cells of an integrated circuit (e.g. ASIC) or a bit stream for a programmable logic device (e.g. FPGA)

Describing a design

In VHDL an entity is used to describe a hardware module

An entity can be described using,

1. Entity declaration
2. Architecture.

1. Entity declaration

It defines the names, input output signals and modes of a hardware module

Syntax

Entity entity _ name *is*

port declaration

end entity name

An entity declaration should start with "entity" and ends with "end" keywords. Ports are interfaces through which an entity can communicate with its environment. Each port must have a name, direction and a type. An entity may have no port declaration also. The direction will be input, output or inout.

In	Port can be read
Out	Port can be written
Inout	Port can be read and written
Buffer	Port can be read and written, it can have only one source.

2. Architecture:

It describes the internal description of design or it tells what is there inside design each entity has at least one architecture and an entity can have many architectures.

Architecture can be described using structural, dataflow, behavioral or mixed style.

Syntax:

Architecture **architecture name** of **entity name** is

architecture declaration part;

begin

statements;

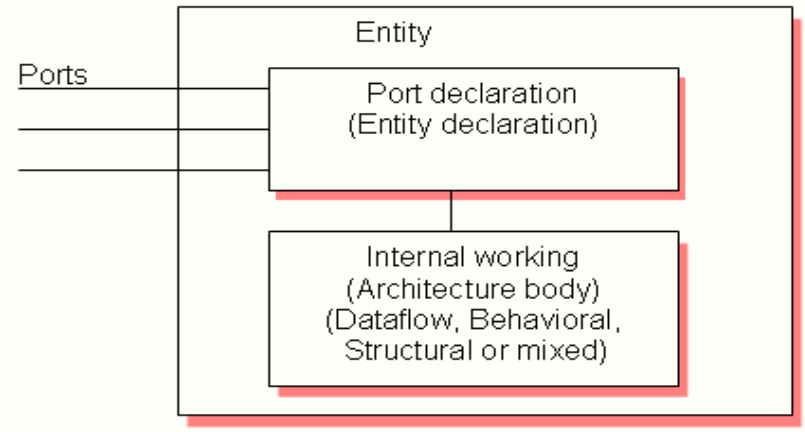
end*architecture_name;*

Here we should specify the entity name for which we are writing the architecture body. The architecture statements should be inside the begin and end keyword. Architecture declarative part may contain variables, constants, or component declaration.

The internal working of an entity can be defined using different modeling styles inside architecture body. They are

- Data flow modeling
- Behavioral modeling
- Structural modeling.

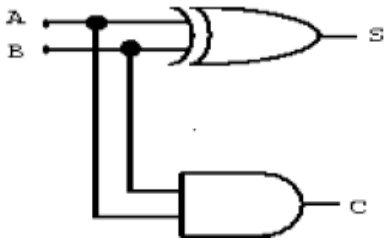
Structure of an entity:



- **Data flow modeling**

In this style of modeling, the internal working of an entity can be implemented using concurrent signal assignment.

Consider a half adder as an example which is having one XOR gate and a AND gate as shown below



Program

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ha is
        port (A,B:in bit;S,C:out bit);
end ha;

architecture ha  arch of ha is
begin
        S<=A xor B;
        C<=A and B;

end ha_ar;
```

Here STD_LOGIC_1164 is an IEEE standard which defines a nine-value logic type, called STD_ULOGIC. Use is a keyword, which imports all the declarations from this package. The architecture body consists of concurrent signal assignments, which describes the functionality of the design. Whenever there is a change in RHS, the expression is evaluated and the value is assigned to LHS.

Behavioral modeling:

In this style of modeling, the internal working of an entity can be implemented using set of statements.

It contains:

Process

statements

Sequential statements

Signal assignment statements

Process statement is the primary mechanism used to model the behavior of an entity it

contains sequential statements, variable assignment (:=) statements or signal assignment (<=) statements etc. It may or may not contain sensitivity list. If there is an event occurs on any of the signals in the sensitivity list, the statements within the process are executed. Inside the process the execution of statements will be sequential and if one entity is having two processes the execution of these processes will be concurrent. At the end it waits for another event to occur.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ha is
    port(
        A : in BIT;
        B : in BIT;
        S : out BIT;
        C : out BIT
        );
end ha;

 architecture ha_arch of ha is
begin
    process(A,B)
    begin
       S<= A xor B;
       C<=A and B;
    end process;

end ha_arch;
```

Here whenever there is a change in the value of A OR B the process statements are executed.

Structural modeling

The implementation of an entity is done through set of interconnected components. It contains

Signal declaration.

Component instances

Port maps.

Wait statements.

Component declaration:

Syntax:

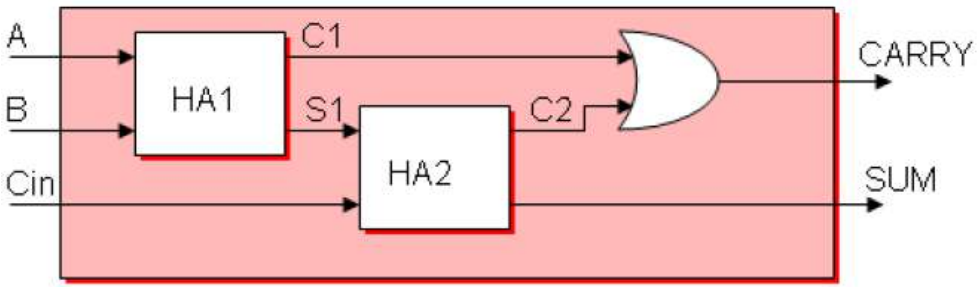
Component Component name

List of interface ports;

end component component_name;

Before instantiating the component it should be declared using component declaration as shown above. Component declaration declares the name of the entity and interface of a component.

Consider the example of full adder using 2 half adder and 1 OR gate.



Schematic Diagram of full adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity fa is
   port(A,B,Cin:in bit; SUM, CARRY:out bit);
end fa;

architecture fa_arch of fa is

component ha
   port(A,B:in bit;S,C:out bit);
end component;

signal C1,C2,S1:bit;

begin
   HA1:ha port map(A,B,S1,C1);
   HA2:ha port map(S1,Cin,SUM,C2);
   CARRY <= C1 or C2;
end fa_arch;
```

The program written for half adder in dataflow modeling is instantiated as shown above. HA is the name of the entity in dataflow modeling. C1, C2, S1 are the signals used for internal connections of the component which are declared using the keyword signal. Port map is used to

connectdifferentcomponentsaswellasconnectcomponentstoportsoftheentity.

Component instantiation is done as follows.

Componentlabel: component name *port map* (signal_list);

Signal list is the architecture signals which will be connected to component ports. This can be done in different ways. What is declared above is positional binding. One more type is the named binding.

The above can be written as,

HA1: ha port map (A => A, B => B, S => S1, C => C1);

HA2: ha port map (A => S1, B => Cin, S=> SUM, C => C2);

2.4 Design using HDL

The vast majority of modern digital circuit design revolves around an HDL description of the desired circuit, device, or subsystem

Most designs begin as a written set of requirements or a high-level architectural diagram. The process of writing the HDL description is highly dependent on the designer's diagram. The process of writing the HDL description is highly dependent on the designer's background and the circuit's nature. The HDL is merely the 'capture language'–often begin with a high-level algorithmic description such as MATLAB or a C++ mathematical model Control and decision structures are often prototyped in flowchart applications, or entered in a state-diagram editor. Designers even use scripting languages (such as PERL) to automatically generate repetitive circuit structures in the HDL language. Advanced text editors (such as PERL) to automatically generate repetitive circuit structures in the HDL language. Advanced text editors (such as Emacs) offer editor templates for automatic indentation, syntax-dependent coloration, and macro-based expansion of entity/architecture/signal declaration.

As the design's implementation is fleshed out, the HDL code invariably must undergo code review, or auditing. In preparation for synthesis, the HDL description is subject to an array of automated checkers. The checkers enforce standardized code guidelines, identifying ambiguous code construct before they can cause misinterpretation by downstream synthesis, and check for common logical coding errors, such as dangling ports or shorted outputs.

In industry parlance, HDL design generally ends at the synthesis stage. Once the synthesis tool has mapped the HDL description into a gate net list, this net list is passed off to the back -

end stage. Depending on the physical technology (FPGA, ASIC gate-array, ASIC standard-cell), HDLs may or may not play a significant role in the back-end flow. In general, as the design flow progresses toward a physically realizable form, the design database becomes progressively more laden with technology-specific information, which cannot be becomes progressively more laden with technology-specific information, which cannot be stored in a generic HDL-description. Finally, a silicon chip is manufactured in a lab.

2.5 Simulating and debugging HDL code

Essential to HDL design is the ability to simulate HDL programs. Simulation allows a HDL description of a design (called a model) to pass design verification, an important milestone that validates the design's intended function (specification) against the code implementation in the HDL description. It also permits architectural exploration. The engineer can experiment with design choices by writing multiple variations of a base design, then comparing their behavior in simulation .Thus, simulation is critical for successful HDL design

To simulate an HDL model, an engineer writes a top-level simulation environment (called a test bench). At minimum, a test bench contains an instantiation of the model (called the device under test or DUT), pin/signal declarations for the model's I/O, and a clock waveform. An HDL simulator–the program that executes the test bench–maintains the simulator clock, which is the master reference for all events in the test bench simulation Events occur only at the instants dictated by the test bench HDL, or in reaction to stimulus and triggering events.

Design verification is often the most time-consuming portion of the design process, due to the disconnect between a device's functional specification, the designer's interpretation of the specification, and the imprecision of the HDL language. The majority of the initial test/debug cycle is conducted in the HDL simulator environment, as the early stage of the design is subject to frequent and major circuit changes. An HDL description can also be prototyped and tested in hardware–programmable logic devices are often used for this purpose. Hardware prototyping is comparatively more expensive than HDL simulation, but offers a real-world view of the design. Prototyping is the best way to check interfacing against other hardware devices, and hardware prototypes, even those running on slow FPGAs, offer much faster simulation times than pure HDL simulation.

2.6. Requirements & Procedure

Requirements:

1. HDL software with front-end (Design entry, synthesis, simulation implementation and programming)
2. FPGAkitwithminimum400,000gatedensity

Procedure:

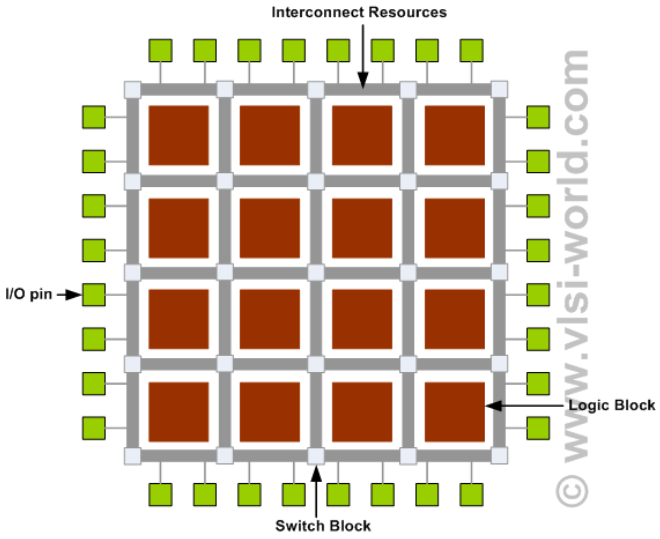
Software part

1. Click on the Project navigator icon on the desktop of your PC. Write the vhdl code, check syntax and perform the functional simulation using Model sim XE.
2. Open a new UCF file and lock the pins of the design with FPGAI/O pins.
3. Implementthedesignbydoubleclickingontheimplementationtoolselection
4. Check the implementation reports.
5. Create programming file.

3. INTRODUCTION TO FPGA (FIELD PROGRAMMABLE GATE ARRAY)

FPGA contains a two dimensional arrays of logic blocks and interconnections between logic blocks. Both the logic blocks and interconnects are programmable. Logic blocks are programmed to implement a desired function and the interconnects are programmed using the switch boxes to connect the logic blocks.
To implement a complex design (CPU for instance), the design is divided into small sub functions and each sub function is implemented using one logic block. All the sub functions implemented in logic blocks must be connected and this is done by programming the interconnects.

3.1 INTERNAL STRUCTURE OF ANFPGA



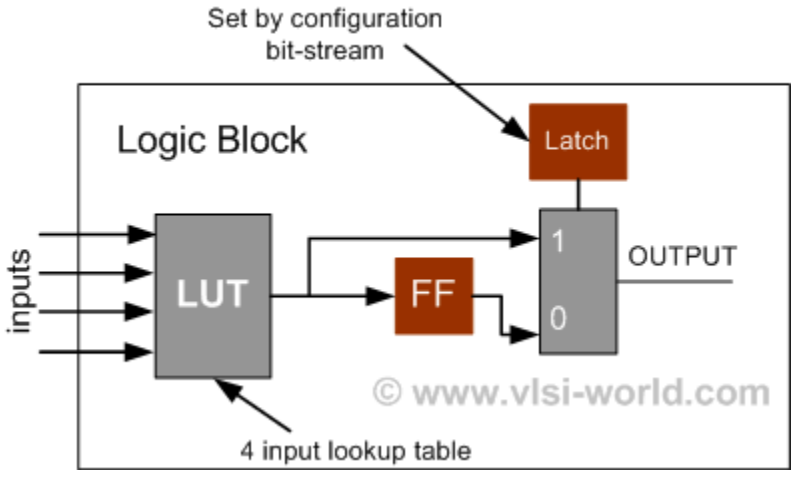
FPGAs, alternative to the custom ICs, can be used to implement an entire System On one Chip (SOC). The main advantage of FPGA is ability to reprogram. User can reprogram an FPGA to implement a design and this is done after the FPGA is manufactured. This brings the name "Field Programmable."

Custom ICs are expensive and takes long time to design so they are useful when produced in bulk amounts. But FPGAs are easy to implement within a short time with the help of Computer Aided Designing (CAD) tools.

3.2 XILINXFPGA

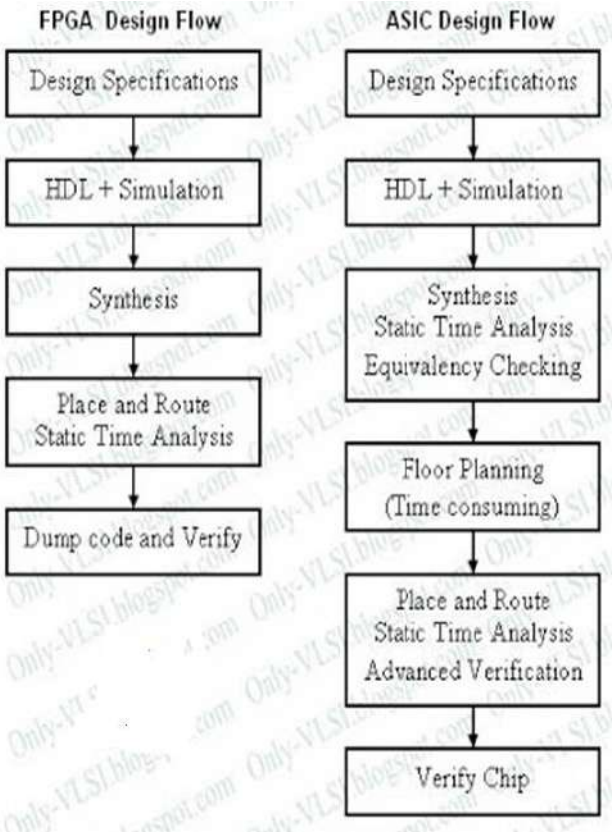
Xilinx logic block consists of one Look Up Table (LUT) and one Flip Flop. An LUT is used to implement number of different functionality. The input lines to the logic block go into the LUT and enable it. The output of the LUT gives the result of the logic function that it implements and the output of logic block is registered or unregistered output from the LUT.

4-INPUT LUT BASED IMPLEMENTATION OF LOGIC BLOCK.



Xilinx LUT

3.3 FPGA/ASIC Design Flow Overview



The ISE (Integrated Synthesis Environment) design flow comprises the following steps:

1. Design Entry

Create an ISE project as follows:

1. Create a project.
2. Create files and add them to your project, including a user constraints (UCF)file.
3. Add any existing files to your project.

Functional Verification

You can verify the functionality of your design at different points in the design flow as follows:

- Before synthesis, run behavioral simulation (also known as RTL simulation).

2. Design Synthesis

Synthesize your design.

3. Design Implementation

Implement your design as follows:

1. Implement your design, which includes the following steps:
   - Translate
   - Map
   - Place and Route

4. Xilinx Device Programming

Program your Xilinx device as follows:

1. Create a programming file (BIT) to program your FPGA.
2. Generate a PROM or ACE file for debugging or to download to your device. Optionally, create a JTAG file.
3. Use IMPACT to program the device with a programming cable.

3.4 PIN SHEET OFXC3S400-5TQ144

FRC1			FRC2			FRC3		
1	**74**	**IO**	1	**84**	**IO**	1	**100**	**IO**
2	**76**	**IO**	2	**85**	**IO**	2	**102**	**IO**
3	**77**	**IO**	3	**86**	**IO**	3		
4	**79**	**IO**	4	**87**	**IO**	4	**103**	**IO**
5	**78**	**IO**	5	**89**	**IO**	5	**105**	**IO**
6	**82**	**IO**	6	**90**	**IO**	6	**107**	**IO**
7	**80**	**IO**	7	**92**	**IO**	7	**108**	**IO**
8	**83**	**IO**	8	**96**	**IO**	8	**113**	**IO**
9	**VCC**	**POWER**	9	**VCC**	**POWER**	9	**VCC**	**POWER**
10	**GND**	**SUPPLY**	10	**GND**	**SUPPLY**	10	**GND**	**SUPPLY**

FRC4			FRC6			FRC7		
1	**112**	**IO**	1	**28**	**IO**	1	**57**	**IO**
2	**116**	**IO**	2	**31**	**IO**	2	**59**	**IO**
3	**119**	**IO**	3	**33**	**IO**	3	**63**	**IO**
4	**118**	**IO**	4	**44**	**IO**	4	**69**	**IO**
5	**123**	**IO**	5	**46**	**IO**	5	**68**	**IO**
6	**131**	**IO**	6	**47**	**IO**	6	**73**	**IO**
7	**130**	**IO**	7	**50**	**IO**	7	**70**	**IO**
8	**137**	**IO**	8	**51**	**IO**	8	**20**	**IO**
9	**VCC**	**POWER**	9	**VCC**	**POWER**	9	**VCC**	**POWER**
10	**GND**	**SUPPLY**	10	**GND**	**SUPPLY**	10	**GND**	**SUPPLY**

FRC5			FRC8			FRC10		
1	**1**	**IO**	1	**93**	**IO**	1	**60**	**IO**
2	**12**	**IO**	2	**95**	**IO**	2	**56**	**IO**
3	**13**	**IO**	3	**97**	**IO**	3	**41**	**IO**

4	**14**	**IO**	4	**98**	**IO**	4	**40**	**IO**
5	**15**	**IO**	5	**99**	**IO**	5	**36**	**IO**
6	**17**	**IO**	6	**194**	**IO**	6	**35**	**IO**
7	**18**	**IO**	7		**IO**	7	**32**	**IO**
8	**21**	**IO**	8	**122**	**IO**	8	**10**	**IO**
9	**23**	**IO**	9	**129**	**IO**	9	**11**	**IO**
10	**24**	**IO**	10	**132**	**IO**	10	**8**	**IO**
11	**26**	**IO**	11	**135**	**IO**	11	**7**	**IO**
12	**27**	**IO**	12	**140**	**IO**	12	**6**	**IO**
13	**5**		13	**5**		13	**5**	
14	**-5**		14	**-5**		14	**-5**	
15	**VCC**		15	**VCC**		15	**VCC**	
16	**GND**	**SUPPLY**	16	**GND**	**SUPPLY**	16	**GND**	**SUPPLY**

FRC9		
1	**5**	**IO**
2	**4**	**IO**
3	**2**	**IO**
4	**141**	**IO**
5	**NA**	**IO**
6	**NA**	**IO**
7	**NA**	**IO**
8	**NA**	**IO**
9	**VCC**	**POWER SUPPLY**
10	**GND**	

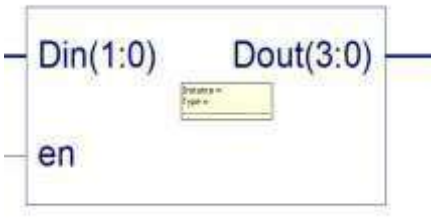
CLK	**52**

PART– A PROGRAMS

EXPERIMENT NO.1

AIM: Write HDL codes for the following combinational circuits.

1 a) 2 TO 4 DECODER



RTL SCHEMATIC

Truth Table

EN	Din(1)	Din(0)	Dout(3)	Dout(2)	Dout(1)	Dout(0)
1	X	x	0	0	0	0
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:00:22 12/23/2020
// Design Name:
// Module Name:    s4btg
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
        input a1;
       wire w1,w2,w3,w4,w5,w6;
       nand x1(w1,a0);
       nand x2(w2,a1);
       nand x3(w3,w1,w2);
       nand x4(w4,a0,w2);
       nand x5(w5,w1,a1);
       nand x6(w6,a0,a1);
       nand x7(d0,w3,w3);
       nand x8(d1,w4,w4);
       nand x9(d2,w5,w5);
       nand x10(d3,w6,w6);
       endmodule
```

1 b) 8 TO 3 ENCODER WITHOUT PRIORITY

RTL
Schematic

Truth Table

INPUTS									OUTPUTS		
en	**Din(0)**	**Din(1)**	**Din(2)**	**Din(3)**	**Din(4)**	**Din(5)**	**Din(6)**	**Din(7)**	**Dout(0)**	**Dout(1)**	**Dout(3)**
1	X	x	x	x	x	x	x	X	z	z	z
0	0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	1	0	0	1	0	1

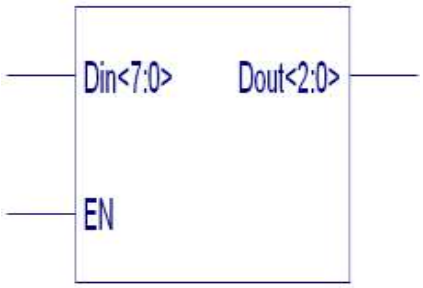
0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:00:22 12/23/2020
// Design Name:
// Module Name:    s4btg
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
    output dout0;
    output dout1;
    output dout2;
    input din0;
    input din1;
    input din2;
    input din3;
    input din4;
    input din5;
    input din6;
    input din7;
reg[2:0]dout;
```

```
always@(din)
begin
case(din)
8'b00000001:dout=3'b000;
8'b00000010:dout=3'b001;
8'b00000100:dout=3'b010;
8'b00001000:dout=3'b011;
8'b00010000:dout=3'b100;
8'b00100000:dout=3'b101;
8'b01000000:dout=3'b110;
8'b10000000:dout=3'b111;
defualt:dout=3'bzzz;
endcase
end
endmodule
```

8 TO 3 ENCODER WITH PRIORITY



RTL Schematic

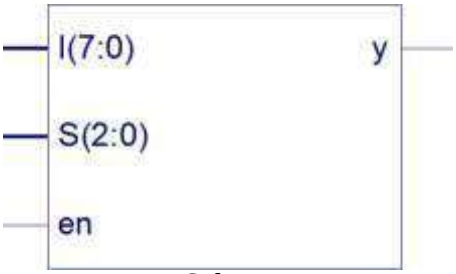
Truth Table

INPUTS									OUTPUTS		
en	Din(0)	Din(1)	Din(2)	Din(3)	Din(4)	Din(5)	Din(6)	Din(7)	Dout(0)	Dout(1)	Dout(3)
0	X	x	x	x	x	x	x	x	z	z	z
1	X	x	x	x	x	x	x	1	1	1	1
1	X	x	x	x	x	x	1	0	1	1	0
1	X	x	x	x	x	1	0	0	1	0	1
1	X	x	x	x	1	0	0	0	1	0	0
1	X	x	x	1	0	0	0	0	0	1	1
1	X	x	1	0	0	0	0	0	0	1	0
1	X	1	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:00:22 12/23/2020
// Design Name:
// Module Name:    s4btg
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
reg y;
always@(s,i)
begin
if(s==3'b000)y=i[0];
else if (s==3'b001)y=i[1];
else if (s==3'b010)y=i[2];
else if (s==3'b011)y=i[3];
else if (s==3'b100)y=i[4];
else if (s==3'b101)y=i[5];
else if (s==3'b110)y=i[6];
else if (s==3'b111)y=i[7];
else y=3'dz;
end
```

1 c) 8 TO 1 MULTIPLEXER



RTL Schematic

Truth Table:

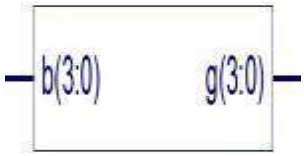
S(2)	S(1)	S(0)	Y
0	0	0	I(0)
0	0	1	I(1)
0	1	0	I(2)
0	1	1	I(3)
1	0	0	I(4)
1	0	1	I(5)
1	1	0	I(6)
1	1	1	I(7)

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:00:22 12/23/2020
// Design Name:
// Module Name:    s4btg
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
reg y;
always@(s,i)
begin
if(s==3'b000)y=i[0];
else if (s==3'b001)y=i[1];
```

```
else if (s==3'b010)y=i[2];
else if (s==3'b011)y=i[3];
else if (s==3'b100)y=i[4];
else if (s==3'b101)y=i[5];
else if (s==3'b110)y=i[6];
else if (s==3'b111)y=i[7];
else y=3'dz;
end
```

1. d) 4-BIT BINARY TO GRAYCONVERTER



RTL Schematic

Truth Table:

INPUTS				OUTPUTS			
B(3)	B(2)	B(1)	B(0)	G(3)	G(2)	G(1)	G(0)
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Verilog Code:

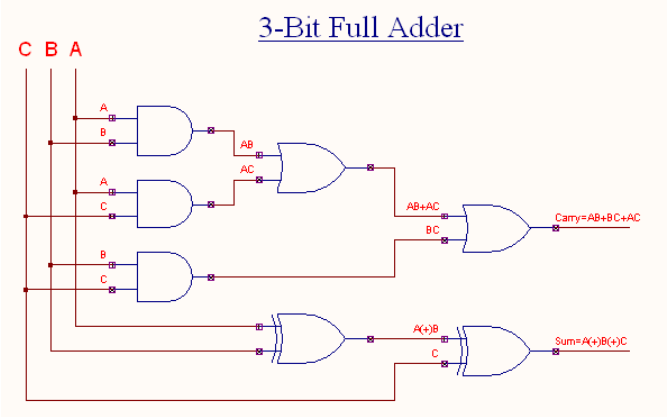
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:00:22 12/23/2020
// Design Name:
// Module Name:    s4btg
// Project Name:
// Target Devices:
// Tool versions:
```

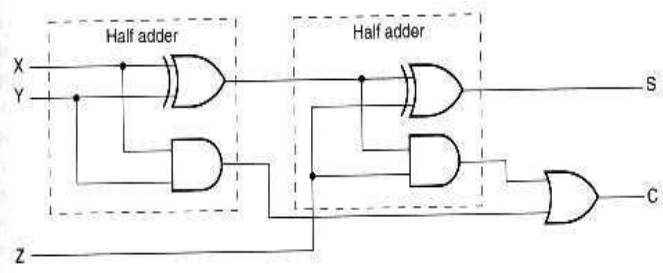
```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module s4btg(B,G);
output[3:0]G;
input[3:0]B;
assign G[3]=B[3];
Gray_bin_1 x1(G[2],G[3],G[2]);
Fulladder_1 x2(G[1], ,1'b0,B[2],B[1]);
Subtractor_1 x3(G[0], ,1'b0,B[1],B[0]);
endmodule

module Gray_bin_1(b0,g1,g0);
output b0;
input g0,g1;
assign b0=g0^g1;
endmodule

module Fulladder_1(sum,cout,a,b,c);
output sum,cout;
input a,b,c;
assign sum=a^b^c;
assign cout=(a&b)|(b&c)|(c&a);
endmodule

module Subtractor_1(diff,bout,a,b,c);
output diff,bout;
input a,b,c;
assign diff=a^b^c;
assign bout=((~a)&b)|((((~a)|b))&c);
endmodule
```

EXPERIMENT NO.2

AIM: Write HDL code to describe the functions of a full Adder Using three modeling styles.

RTL Schematic





Truth Table

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
timescale 1ns / 1ps
/`//////////////////////////////////////////////////////////////////////////////////
```

```
// Company:
// Engineer:
//
// Create Date:    10:11:38 01/28/2021
// Design Name:
// Module Name:    fulladder
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module fulladder(sum,cout,yxor,yxnor,yand,yor,a,b,c);
output sum,cout,yxor,yxnor,yand,yor;
inout a,b,c;
assign sum=a^b^c;
assign cout=(a@b)|(b@c)|(c@a);
assign yxor=a^b^c;
assign yxnor=~(a^b^c);
assign yand=a@b@c;
assign yor=a|b|c;
endmodule
```

EXPERIMENT NO. 3

AIM: Write a model for 4/8/32 bit Arithmetic Logic Unit using the schematic diagram shown below.

RTL Schematic

Truth table

Operation	Opcode	A	B	Zout
A+B	000	1111	0000	00001111
A-B	001	1110	0010	00001100
A or B	010	1111	1000	00001111
A and B	011	1001	1000	00001000
Not A	100	1111	0000	11110000
A1*B1	101	1111	1111	11100001
A nand B	110	1111	0010	11111101
A xor B	111	0000	0100	00000100

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:41:32 01/12/2021
// Design Name:
// Module Name:    program1
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module program1(result,a,b,opcode,enable);
output[32:0]result;
input signed [31:0]a,b;
input[2:0] opcode;
input enable;
reg[32:0] result;
always@(opcode,a,b,enable)
begin
if(enable==0)
begin
result=31'bx;
end
else
begin
case(opcode)
3'b000:begin result=a+b;end
3'b001:begin result=a-b;end
```

```
3'b010:begin result=a+1;end
3'b011:begin result=a-1;end
3'b100:begin result=!a;end
3'b101:begin result=~a;end
3'b110:begin result=a/b;end
3'b111:begin result=a&b;end
endcase
result[32]=1'b1;
end
end
endmodule
```

EXPERIMENT NO.4

AIM: Develop the HDL code for the following flip-flop: T, D, SR and JK.

D-FLIP FLOP

RTL Schematic

Truth Table

CLK	D	Q	Qb	OPERATION
0	x	Q	Qb	No change
1	0	0	1	Reset
1	1	1	0	Set

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:41:44 01/12/2021
// Design Name:
// Module Name:    sr_ff
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module dff(q,qb,d,clk);
output q,qb;
```

```
input d,clk;
reg q=0,qb=1;
always@(posedge clk)
begin
q=d;
qb=~q;
end


endmodule
```

SR FLIP – FLOP

RTL Schematic

Truth Table

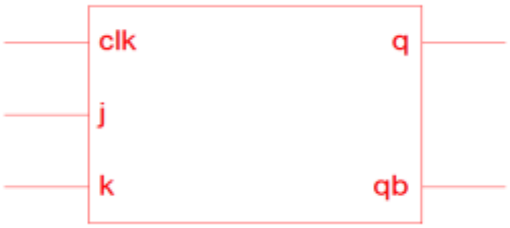
CLK	s	r	Q	Qb
0	x	x	Q	Qb
1	0	1	0	1
1	1	0	1	0
1	1	1	Not defined	
1	0	0	Q	Qb

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:41:44 01/12/2021
// Design Name:
// Module Name:    sr_ff
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module srff(q,qb,sr,clk);
output q,qb;
input clk;
input[1:0]sr;
```

```
reg q=0,qb=1;
always@(posedge clk)
begin
case(sr)
2'b00:q=q;
2'b01:q=0;
2'b10:q=1;
2'b11:q=1'bz;
endcase
qb=~q;
end
endmodule
```

JK-FLIP FLOP

RTLSchematic



TruthTable

CLK	J	K	Q	Qb
0	x	x	Q	Qb
1	0	1	0	1
1	1	0	1	0
1	1	1	Qb	Q
1	0	0	Q	Qb

```
/ 1ps
/ Engineer`timescale:
//
// Create Date:    15:35:53 12/23/2020
// 1ns D//////////////////////////////////////////////////////////////////////////////
// Company:
//esign Name:
// Module Name:    jkff
// Project Name:
// Target Devices:
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////

module jkff(q,qb,jk,clk);
output q,qb;
input clk;
input[1:0]jk;
reg q=0,qb=1
```

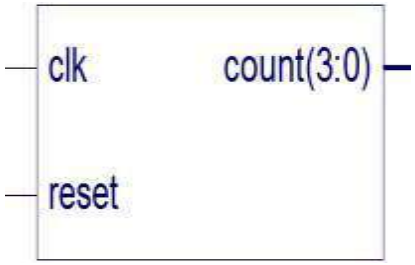
```
// Tool versions:
// Description:
//;
always@(posedge clk)
begin
case(jk)
2'b00:q=q;
2'b01:q=0;
2'b10:q=1;
2'b11:q=~q;
endcase
qb=~q;
end
endmodule
```

EXPERIMENT-5

4-bit BCD Synchronous counter.

Write Verilog code for 4-bit BCD synchronous counter.

RTL Schematic



TruthTable

Clock	Reset	Current State	Next State
1	1	xxx	0000
1	0	0000	0001
1	0	0001	0010
1	0	0010	0011
1	0	0011	0100
1	0	0100	0101
1	0	0101	0110
1	0	0110	0111
1	0	0111	1000
1	0	1000	1001
1	0	1001	1010
1	0	1010	1011
1	0	1011	1100
1	0	1100	1101
1	0	1101	1110
1	0	1110	1111
0	x	1111	0000

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:06:13 01/27/2021
// Design Name:
// Module Name:    bcd
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module bcd(count, clk,reset);
   output [3:0] count;
   input clk,reset;
        reg[3:0]count=4'b0000;
        always@(posedge clk)
        begin
        if((reset==1)|(count==4'b1001))
        count = 4'b0000;
        else
        count = count+1;
        end
        endmodule
```

EXPERIMENT-6

Clock divider performing division of clock by 2, 4, 8 and 16

Write Verilog code for counter with given input clock and check whether it works as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:53:24 12/28/2020
// Design Name:
// Module Name:    freediv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module freediv(clk2,clk4,clk8,clk16,clk,reset);
output clk2,clk4,clk8,clk16;
input clk,reset;
reg clk2,clk4,clk8,clk16;
reg [3:0]count=4'b0000;
always@ (posedge  clk)
begin
if (reset==1)
begin count=4'b0000;end
else
begin count=count+1;end
clk2=count[0];
clk4=count[1];
clk8=count[2];
clk16=count[3];
end
endmodule
```

PART- B
INTERFACING PROGRAMS

EXPERIMENT-1

Clock divider circuit that generates 1/2, 1/3rd and 1/4

Write a Verilog code to design a clock divider circuit that generates 1/2, 1/3rd and 1/4thclock from a given input clock. Port the design to FPGA and validate the functionality through oscilloscope

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:53:24 12/28/2020
// Design Name:
// Module Name:    freediv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
 module vcode(clk_2,clk_4,clk_8,clk_16,clk,reset);
output clk_2,clk_4,clk_8,clk_16;
input clk,reset;
reg clk_2,clk_4,clk_8,clk_16;
reg[3:0]count=4'b0000;
always@(posedge clk)
begin
if(reset==1)
begin count=4'b0000;end
else
begin count=count+1;end
clk_2=count[0];
clk_4=count[1];
clk_8=count[2];
clk_16=count[3];
end
```

EXPERIMENT-2

DCMOTOR:

Interface a DC motor to FPGA and write Verilog code to change its speed and direction.

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

entity dcmotor is
   Port ( clk : in STD_LOGIC;
        reset,dir : inSTD_LOGIC;
        pwm : out STD_LOGIC_VECTOR (1 downto 0); rly :
        out STD_LOGIC;
        row : in STD_LOGIC_VECTOR (3 downto 0)); end
dcmotor;

architecture Behavioral of dcmotor is
signal counter:STD_LOGIC_VECTOR (7 downto 0):="11111110";
signal div_reg:STD_LOGIC_VECTOR (16 downto 0);
signal dclk,ddclk,datain,tick:STD_LOGIC;
signal dcycle:integer range 0 to 255 ; begin
process(clk,div_reg)
begin
if(clk'event and clk='1')then
div_reg<= div_reg+1;
end if;
end process;

ddclk<=div_reg(12);
tick<=row(0)and row(1)and row(2)and row(3);

process(tick)
begin
if falling_edge(tick)then
case row is
    when "1110"=> dcycle<=255;---speed highest
    when "1101"=> dcycle<=200;
    when "1011"=> dcycle<=150;
    when "0111"=> dcycle<=100;---speed lowest
    when others=> dcycle<=100;
 end case; end if;
 end process;


   process(ddclk,reset)
   begin
   if reset='0'then
   counter<="00000000"; pwm<="01";
   elsif(ddclk'event and ddclk='1')then
   counter<=counter+1;
```

```
if(counter >=dcycle)then
pwm(1)<='0';
else pwm(1)<='1';
end if;
end if;
end process; rly<=dir;
end Behavioral;
```

```
NET "clk"  LOC = "p52"  ;
NET "dir"  LOC = "p76"  ;
NET "pwm<0>"  LOC = "p4";
NET "pwm<1>" LOC = "p141" ;
NET "reset"  LOC = "p74"  ;
NET "rly"  LOC = "p5";
NET "row<0>" LOC = "p69" ;
NET "row<1>" LOC = "p63" ;
NET "row<2>" LOC = "p59" ;
NET "row<3>"  LOC = "p57";
```

Reset	Direction/rly	PWM	operation
0	0	01	stop
1	1	11	Anticlockwise
1	0	11	clockwise

EXPERIMENT-3

STEPPERMOTOR:

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:53:24 12/28/2020
// Design Name:
// Module Name:    freediv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
//
 Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

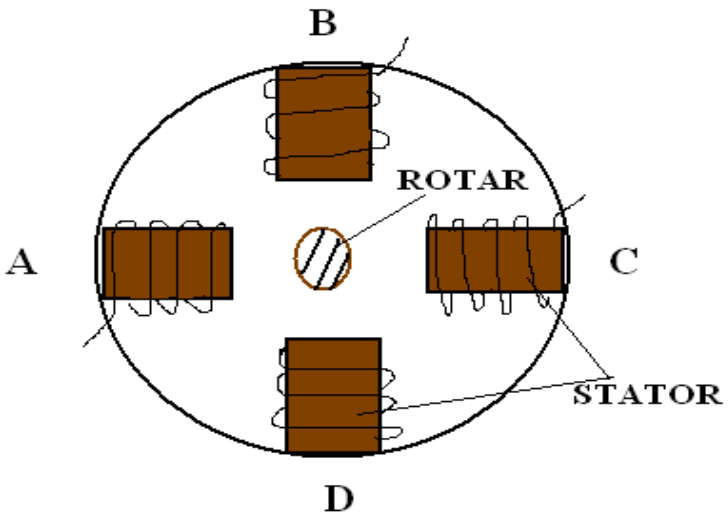
```
//////////////////////////////////////////////////////////////////////////////////
module seeppermotor(
input clk;
input reset;
input dir;
input[1:0]row;
output reg[3:0]dout
);
reg[25:0]divclk;
reg intclk;
reg[3:0]register;
always@(posedge clk)
begin
divc1k=divclk+1;
end
always@(row)
begin
case(row)
2'b00:intclk=divclk[21];
2'b00:intclk=divclk[19];
2'b00:intclk=divclk[17];
2'b00:intclk=divclk[15];
default:intclk=divclk[21];
endcase
end
always@(posedge intclk)
begin
if(!reset)
register=4'b1001;
else
begin
if(!dir)
register={register[0],register[3:1]};
else
register={register[2:0],register[3]};
end
dout=register;
end

endmodule
```

```
NET "clk" LOC =
"p52"; NET "dir" LOC =
"p85"; NET "rst" LOC =
"p84";
NET "dout<0>" LOC = "p112";
NET "dout<1>" LOC = "p116";
NET "dout<2>" LOC = "p119";
```

CLOCKWISE (DIR= '1')					ANTICLOCKWISE (DIR='0')			
A	B	C	D		A	B	C	D
1	0	0	0		0	0	1	0

0	1	0	0		0	1	0	0
0	0	1	0		1	0	0	0
0	0	0	1		0	0	0	1
ABCD ABCD ABCD ABCD…..					CBAD CBAD CBAD CBAD…..			

Reset	Direction	operation
0	X	stop
1	1	clockwise
1	0	anticlockwise



When current is passed through the coil, the circular magnetic field is generated.

EXPERIMENT-4

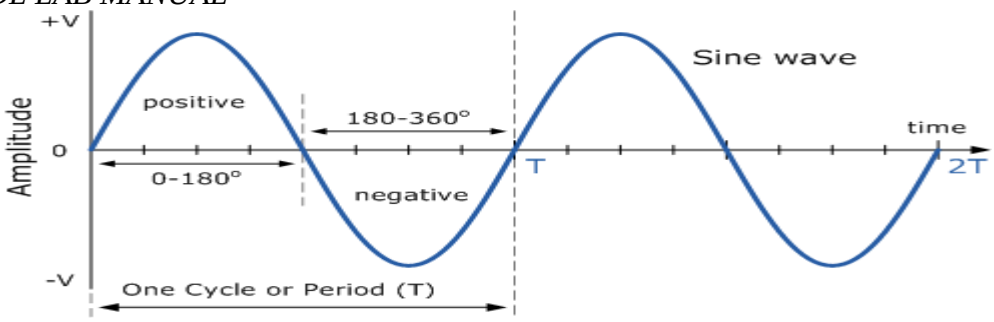
DAC- SINEWAVE:

Interface a DAC to FPGA and write Verilog code to generate Sine wave of frequency F KHz (eg. 200 KHz) frequency.

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

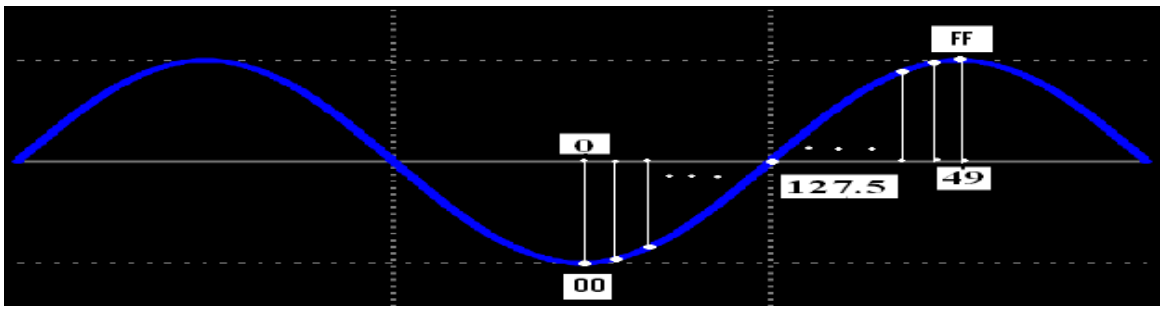
```
entity sinewave is Port
(clk:in std_logic;
      dout : out std_logic_vector(0 downto 7)); end
sinewave;

architecture behavioral of sinewave is signal
a: integer range 1 to 1000000:=1; signal b:
integer range 0 to 49:=0;
signalr:std_logic:='1';
begin
process(clk,b,r)
type sine is array (0 to 49) of std_logic_vector (0 downto 7);
constant sinedata:sine:=(x"00", x"01",  x"02",x"04", x"06", x"09", x"0C", x"0F",
                         x"14", x"18", x"1D", x"22",x"28", x"2E",x"34", x"3B",
                         x"42",x"49",x"50", x"58",x"5F",x"67",     x"6F",x"77",
                         x"7F",x"87",x"8F",x"97",x"9F",x"A7",x"AE",    x"B5",
                         x"BD",x"C3",x"CA",x"D0",x"D6",x"DC",x"E1",x"E6",
                         x"EB",x"FF",x"F3",x"F6",x"F8",x"FB", x"FC",x"FD", x"FE",x"FF");
    begin
    if (clk'event and clk='1')then
          a<=a+1;
            if (r='1') then
                 if  (a=1) then
                        dout <=sinedata(b);
                        b<=b+1;
                         a<=1;
                        end if;

                 if  (b=48) then
                         r<='0';
                         end if;
            elsif (r='0') then
                  if  (a=1) then
                        dout <= sinedata(b);
                        b<=b-1;
                         a<=1;
                        end if;
                if  (b=1) then
                      r<='1';
                        end if; end if; end if; end
                        process;
                         end behavioral;
```

```
NET "clk" LOC = "p52";
NET "dout<7>" LOC ="p1";
NET "dout<6>" LOC ="p12";
NET "dout<5>" LOC ="p13";
NET "dout<4>" LOC ="p14";
NET "dout<3>" LOC ="p15";
NET "dout<2>" LOC ="p17";
NET "dout<1>" LOC="p18";
NET "dout<0>" LOC ="p21";
```



Small ‘x’ --- Hexadecimal number
Capital ‘X’--- Unknown or don’t care
θ = 180° / 50 intervals = 3.6° = Each interval
During Negative Quarter (1/4) cycle = 127.5-127.5 Sin(nθ) $0 \leq n \leq 24$

n=24 ---- 00 h
n=23 ---- 01 h
n=22 ---- 02 h
:
n=2 ---- 6Fh
n=1 ---- 77 h
n=0 ---- 7F h corresponds to127.5



During Positive Quarter (1/4) cycle = 127.5+127.5 Sin(nθ) $1 \leq n \leq 25$

n=1 ---- 87 h
n=2 ---- 8F h
n=3 ---- 97 h
:
n=24 ---- FEh
n=25 ---- FF h corresponds to255

EXPERIMENT-5

ELEVATOR

Write Verilog code using FSM to simulate elevator operation:

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


entity ssg is
   Port ( keyreturn : in  STD_LOGIC_VECTOR (3 downto 0);
          keyscan : buffer STD_LOGIC_VECTOR (3 downto 0):="1000"; segm :
          out  STD_LOGIC_VECTOR (3 downto 0);
          clk : inSTD_LOGIC;
          dis : out STD_LOGIC_VECTOR (6 downto 0):="0000000"); end
ssg;

architecture Behavioral of ssg is
signal a,temp:integer range 0 to 15:=0;----initial content to be displayed after dumping
signal b:integer range 0 to 2000009;
begin
process(clk)
begin
if(clk'event and clk='1')then keyscan<=keyscan(0)

 & keyscan(3 downto 1);

 if    keyscan="0001" and keyreturn="0001" thena<=0; elsif
 keyscan="0001" and keyreturn="0010" then a<=1; elsif
 keyscan="0001" and keyreturn="0100" then a<=2; elsif
 keyscan="0001" and keyreturn="1000" then a<=3; elsif
 keyscan="0010" and keyreturn="0001" then a<=4; elsif
 keyscan="0010" and keyreturn="0010" then a<=5; elsif
 keyscan="0010" and keyreturn="0100" then a<=6; elsif
 keyscan="0010" and keyreturn="1000" then a<=7; elsif
 keyscan="0100" and keyreturn="0001" then a<=8; elsif
 keyscan="0100" and keyreturn="0010" then a<=9; elsif
 keyscan="0100" and keyreturn="0100" then a<=10; elsif
 keyscan="0100" and keyreturn="1000" then a<=11; elsif
 keyscan="1000" and keyreturn="0001" then a<=12; elsif
 keyscan="1000" and keyreturn="0010" then a<=13; elsif
 keyscan="1000" and keyreturn="0100" then a<=14; elsif
 keyscan="1000" and keyreturn="1000" then a<=15; endif;
```

```
end if;
endprocess;

 Process(clk,a,temp)
 begin
  if(clk'event and clk='1')then
     b<=b+1;
 if(b=2000000)then ------Delay between one floor to next floor
   if(temp<a) then   -------a=current floor  and temp= destination floor
     temp<=temp+1 ;
     b<=0;
elsif(temp/=a) then
     temp<=temp-1 ;
     b<=0;
end      if;
end      if;
end if;
end process;


 process(temp) ---- when the key of destination floor is pressed process will be activated type
          sevseg is array (0 to 15 )of std_logic_vector(6 downto 0);
 constantsegdis:sevseg:=  (     "1111110","0110000","1101101","1111001",
                                "0110011","1011011","1011111","1110000",
                                "1111111","1111011","1110111","0011111",
                                "1001110","0111101","1001111","1000111");
 begin
   dis<=segdis(temp);
   segm<="1110";
end process; end
Behavioral;
```

CURRENT FLOOR	DESTINATION FLOOR	INCREMENT/DECREMENT
0	5	increment from 0 to 5
1	4	increment from 1 to 4
2	6	increment from 2 to 6
3	8	increment from 3 to 8
4	A	increment from 4 to A
5	7	increment from 5 to 7
6	E	increment from 6 to E
7	6	decrement from 7 to 6
8	1	decrement from 8 to 1
9	3	decrement from 9 to 3
A	2	decrement from A to 2
B	8	decrement from B to 8
C	1	decrement from C to 1
D	5	decrement from D to 5
E	6	decrement from E to 6
F	3	decrement from F to 3

EXPERIMENT-6

SEVEN SEGMENTDISPLAY

Write Verilog code to convert an analog input of a sensor to digital form and to display the same on a suitable display like set of simple LEDs, 7-segment display digits or LCD display.

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

entity ssg is
   Port ( keyreturn : in  STD_LOGIC_VECTOR (3 downto 0);
          keyscan : buffer STD_LOGIC_VECTOR (3 downto 0):="1000"; segm :
          out  STD_LOGIC_VECTOR (3 downto 0);
          clk : inSTD_LOGIC;
          dis : out STD_LOGIC_VECTOR (6 downto 0):="0000000"); end
ssg;

architecture Behavioral of ssg is
signal a:integer range 0 to 15:=0;
begin
process(clk)
begin
if(clk'event and clk='1')then keyscan<=keyscan(0)
  & keyscan(3 downto 1);
if keyscan="0001" and keyreturn="0001"  then a<=0; elsif
 keyscan="0001" and keyreturn="0010" then a<=1; elsif
 keyscan="0001" and keyreturn="0100" then a<=2; elsif
 keyscan="0001" and keyreturn="1000" then a<=3; elsif
 keyscan="0010" and keyreturn="0001" then a<=4; elsif
 keyscan="0010" and keyreturn="0010" then a<=5; elsif
 keyscan="0010" and keyreturn="0100" then a<=6; elsif
 keyscan="0010" and keyreturn="1000" then a<=7; elsif
 keyscan="0100" and keyreturn="0001" then a<=8; elsif
 keyscan="0100" and keyreturn="0010" then a<=9; elsif
 keyscan="0100" and keyreturn="0100" then a<=10; elsif
 keyscan="0100" and keyreturn="1000" then a<=11; elsif
 keyscan="1000" and keyreturn="0001" then a<=12; elsif
 keyscan="1000" and keyreturn="0010" then a<=13; elsif
 keyscan="1000" and keyreturn="0100" then a<=14; elsif
 keyscan="1000" and keyreturn="1000" then a<=15; endif;
 end if;
 end process; process(a)
```

```
        type sevseg is array (0 to 15 )of std_logic_vector(6 downto 0);
constantsegdis:sevseg:=     ("1111110","0110000","1101101","1111001",

                                "0110011","1011011","1011111","1110000",

"1111111","1111011","1110111","0011111",

"1001110","0111101","1001111","1000111");

begin
   dis<=segdis(a);
segm<="1110";---To activate one segment out of four segments
end process;
end Behavioral;
```

```
NET "clk" LOC = "p52"  ;
NET "dis<0>" LOC = "p18" ;
NET "dis<1>" LOC = "p17" ;
NET "dis<2>" LOC = "p15" ;
NET "dis<3>" LOC = "p14" ;
NET "dis<4>" LOC = "p13" ;
NET "dis<5>" LOC = "p12" ;
NET "dis<6>"  LOC = "p1";
NET "keyreturn<0>" LOC = "p112" ;
NET "keyreturn<1>" LOC = "p116" ;
NET "keyreturn<2>" LOC = "p119" ;
NET "keyreturn<3>" LOC = "p118" ;
NET "keyscan<0>" LOC = "p123" ;
NET "keyscan<1>" LOC = "p131" ;
NET "keyscan<2>" LOC = "p130" ;
NET "keyscan<3>" LOC = "p137" ;
NET "segm<0>"  LOC = "p27";
NET "segm<1>" LOC = "p26" ;
NET "segm<2>" LOC = "p24" ;
NET "segm<3>"  LOC = "p23";
```

If common cathode, a=b=c=d=e=f=g= 1
Ifcommonanode, a=b=c=d=e=f=g=0

Keyscan / keyreturn	0111	1011	1101	1110
0111	0	1	2	3
1011	4	5	6	7
1101	8	9	A	B
1110	C	D	E	F

Display	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
A	1	1	1	0	1	1	1
B	0	0	1	1	1	1	1
C	1	0	0	1	1	1	0
D	0	1	1	1	1	0	1
E	1	0	0	1	1	1	1
F	1	0	0	0	1	1	1

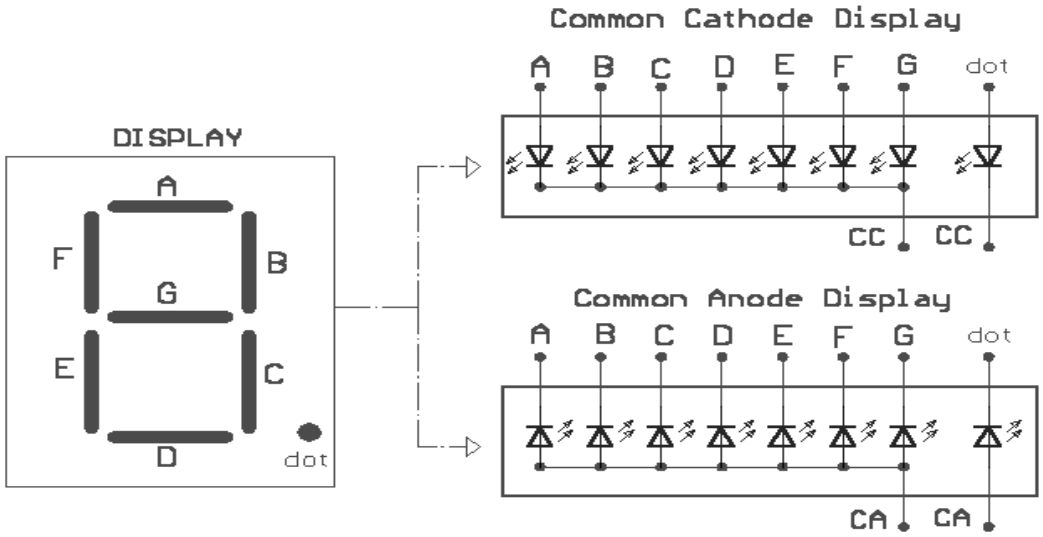


Fig.7-- Common Anode/Cathode DISPLAY Sam 6/02