Lab1: Introduction to VHDL and implementing its model of architecture

Objective: To understand the basic VHDL structure and implement basic combinational circuits using different architecture of VHDL.

Theory:

VHDL is a hardware description language that can be used to model a digital system. The digital system can be as simple as a logic gate or as complex as a complete electronic system. A hardware abstraction of this digital system is called an entity in this text. An entity X, when used in another entity Y, becomes a component for the entity Y. Therefore a component is also an entity, depending on the level at which you are trying to model. To describe an entity, VHDL provides five different types of primary constructs, called design units. They are: 1. Entity declaration 2. Architecture body 3. Configuration declaration 4. Package declaration 5. Package body

An entity is modeled using an entity declaration and at least one architecture body. The entity declaration describes the external view of the entity; for example, the input and output signal names. The architecture body contains the internal description of the entity; for example, as a set of concurrent or sequential statements that represents the behavior of the entity.

A configuration declaration is used to create a configuration for an entity. It specifies the binding of one architecture body from many architecture bodies that may be associated with the entity. It may also specify the bindings of components used in the selected architecture body to other entities.

A package declaration encapsulates a set of related declarations, such as type declarations, subtype declarations, and subprogram declarations, which can be shared across two or more design units. A package body contains the definitions of subprograms declared in a package declaration.

VHDL invariants:

- Not case sensitive.
- Not sensitive to white Space.
- Comments begin with two consecutive dashes (--)
- Every statement is terminated with a semicolon.
- Statements are inherently concurrent.

Basic Structure of VHDL

VHDL Code basically comprises of three parts

1) Library

2) Entity

3) Architecture

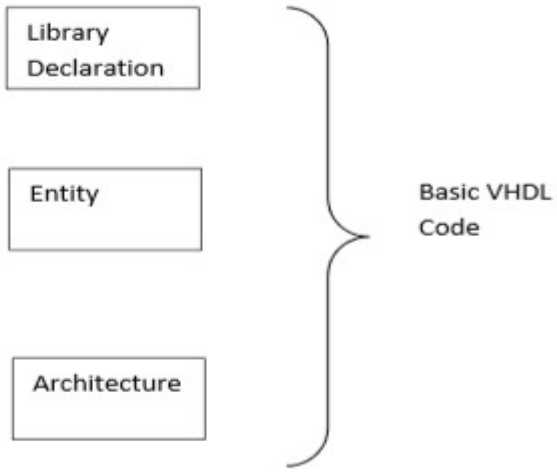


Figure 1: Basic VHDL code structure

1. Library: Contains a list of libraries to be used in the design e.g. a) ieee b) std c) work etc.

2 Entity: Specifies the I/O pins of the circuit.

3 Architecture: Contains the VHDL code which describes how the circuit should behave.

Library Declaration:

A Library is a commonly used piece of code. Placing such pieces inside a library allows them to be reused or shared by other designs. To declare a Library (that is to make it visible to the design) 2 lines of code are needed, one containing the name of the library, and the other use a clause as shown below:

Library library_name; Use library_name.package_name.package_parts;

At least 3 packages from 3 different libraries are needed

a) ieee.std_logic_1164 (from ieee)

b) standard (from the std library)

c) work (work library)

The declaration looks like:

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

IEEE library consists of:

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

Purpose of IEEE library packages

(i) Std_logic_1164:- specifies the std_logic (8 levels) and std_ulogic (9 levels) for multivalued logic systems.

(ii) std_logic_arith:- specifies the signed and unsigned data types and related arithmetic and comparison operation. It also contains several data conversion function which allows one type to be converted to another for example conv_integer(p) which converts a parameter p to a type integer. Conv_unsigned(p,b) converts a parameter p to unsigned value of b bits.

(iii) std _logic_unsigned:- contains functions that allow operations with std_logic_vector to be performed as if data were of type unsigned.

Entity

An entity is a list with specification of all input and output pins (PORTS) of the circuit. Its syntax is shown as below

ENTITY entity_name IS

PORT (

port_name : signal_mode signal_type;

port_name : signal_mode signal_type; ……..);

END entity name;

The mode of the signal can be IN, OUT, INOUT, BUFFER. IN and OUT are truly unidirectional pins while INOUT is bidirectional. BUFFER is employed when the output signal must be used (read) internally. The type of the signal can be BIT, STD_LOGIC, and INTEGER etc. Name of the entity can be basically any name except VHDL reserved words.

```
ENTITY and_gate IS
        PORT(
                a_in: IN STD_LOGIC;
                b_in: IN STD_LOGIC;
                z_out: OUT STD_LOGIC );
END and_gate;
```

Architecture

Architecture is a description of how the circuit should behave and its syntax is as follows:

ARCHITECTURE architecture_name of entity_name IS

[declaration]

BEGIN

(code)

END architecture_name;

Note that declarative part is optional where signals and constants are declared and the code part is written down from BEGIN. For example

ARCHITECTURE and_gate_arch OF and_gate IS

BEGIN

Z <= a AND b;

END and_gate_arch;

VHDL: Architecture Model:

Data Flow Modeling*:*

Dataflow model specifies a circuit as a concurrent representation of the flow of data through the circuit. In this modelling, the internal working of a system is implemented using concurrent statements. It can be used for small and primitive circuits but not for complex designs. In this style of architecture, whenever there is change in signal of right hand side, the expression is evaluated and assigned to Left hand side.

Logic diagram and logic equation of OR gate

OR gates have two inputs and one output and they implement the boolean logic of multiplication. Its equation is as follows

Z (A or B) = A+B

Coding for OR gates:

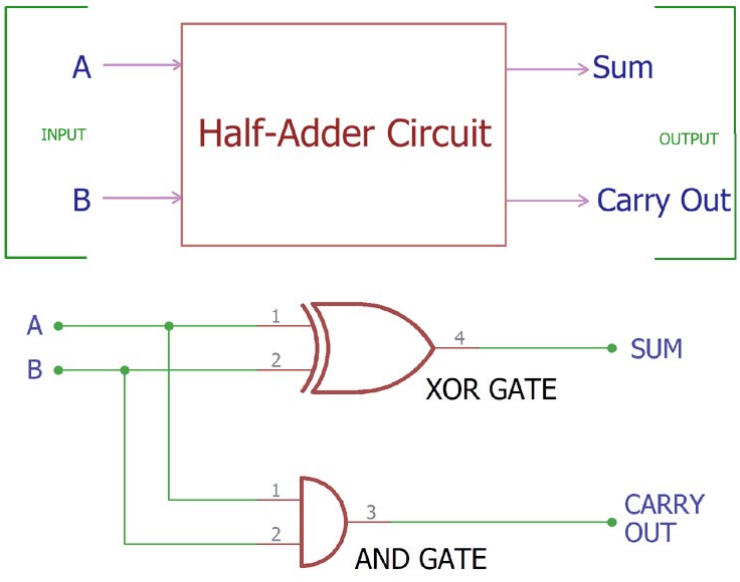
```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity OR_gate is
        Port(
                A,B: in std_logic;
                Z:out std_logic);
```

```
End OR_gate;
Architecture OR_arch is
Begin
        Z<=A or B;
End OR_arch;
```

Test Bench:

```
--Stimulus Process
stim_proc:process
begin
wait for 10ns;
a<='1';
b<='0';
wait for 10ns;
a<='0';
b<='1';
wait for 10ns;
a<='0';
b<='0';
wait for 10ns;
a<='1';
b<='1';
wait for 10ns;
end process;
```

Behavioral Modelling: The behavioral style models how the circuit output will behave to the circuit inputs. This model does not reflect how the circuit is implemented when it is synthesized. Process statement is the core part of behavioral style architecture. In this style, the internal working is implemented using sequential statements within the process statements.



```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity half_adder is
Port(
        A,B: In std_logic;
        SUM,CARRY_OUT:out std_logic);
End half_adder;
Architecture Half_adder_arch of half_adder is
    Begin
            Process_adder: process (A,B);
            Begin
                    SUM<= A xor B;
                    CARRY_OUT<=A and B;
            End process Process_adder;
End Half_adder_arch;
```

Test Bench:

```
stim_proc:process
begin
wait for 10ns;
A<='1';
B<='0';
…
 wait for 10ns;
A<='0';
B<='1';
 wait for 10ns;
end process;
end Behavioral;
```

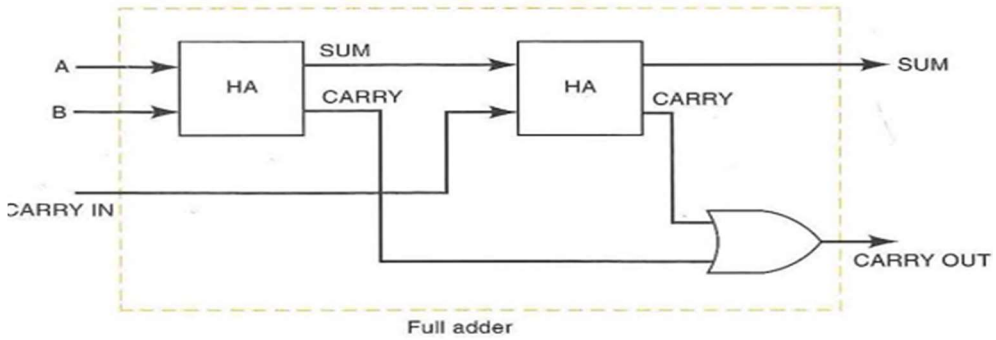
Structural Modelling:

In this modeling, an entity is described as a set of interconnected components. A component instantiation statement is a concurrent statement. Therefore, the order of these statements is not important. The structural style of modeling describes only an interconnection of components (viewed as black boxes), without implying any behavior of the components themselves nor of the entity that they collectively represent.

In Structural modeling, architecture body is composed of two parts − the declarative part (before the keyword begin) and the statement part (after the keyword begin).

Structural Model for Full Adder:

It consists of two components: i.e. half adder and OR gates. So, using OR and Half adder as components, we can code structural model of VHDL.




```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fulladd is
   Port ( x : in  STD_LOGIC;
          y : in  STD_LOGIC;
          cin : in  STD_LOGIC;
          sumf : out  STD_LOGIC;
          cout : out  STD_LOGIC);
end fulladd;
architecture Structural of fulladd is
          component OR_gate  is
          Port(
                    A,B: in std_logic;
                    Z:out std_logic);
End component OR_gate  ;
 Component Half_adder is
          Port(
                    A,B: In std_logic;
                    SUM,CARRY_OUT:out std_logic);
          End component half_adder;
signal s1, C2, C1: std_logic;
begin
HA1: Half_adder portmap
                         ( A=>x,
                          B=>y,
                         SUM=>s1,
                         CARRY_OUT=>  C1);
HA2: Half_adder portmap
                         (A=>s1,
                         B=> cin,
                         SUM=>sumf,
                         CARRY_OUT=>C2);
```

OR gate as a components

Half adder as another components

```
OR1: OR_gate portmap
                    (       A=> C1,
                            B=> C2,
                            Z=>Cout);
End architecture structural;

Stim_proc: process
Begin
        x<='0';
        y<='0';
        cin<='0';
wait for 10ns;
x<='1';
        y<='0';
        cin<='1';
wait for 10ns;
…
Wait;
End process stim_proc;
```

Observations:

1) RTL of the above combinational circuits.
2) Timing diagram of above defined circuit.

Discussion and conclusion:

Comparison of dataflow, behavioral and structural model of VHDL and conclude.

Lab 2: Implementation of different combinational circuit using VHDL

Objective: To implement the combinational logical circuit (MUX, DeMUX, encoder, decoder, comparator) using VHDL and observe its waveform and RTL.

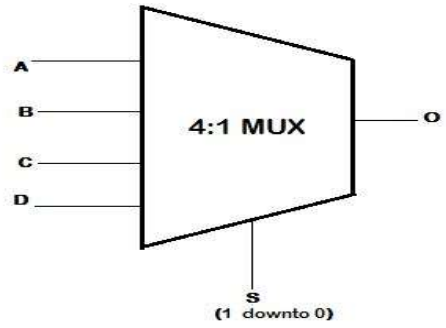
Theory:

Multiplexer

When we implement a digital hardware architecture, we often need to select an input to our logic between several different inputs. This selection logic is called digital multiplexer or **MUX**.

We name it digital multiplexer, to distinguish it from an analog multiplexer. An analog multiplexer implements the same function as digital MUX selecting the source of a signal from different analog source instead of digital.

As clear in Figure1, a MUX can be visualized as an n-way virtual switch whose output can be connected to one of the different input sources. On the left side of the Figure1, you can see the typical MUX representation. The number near the input ports indicates the selector value used to route the selected input to the output port.



MUX using various control statements: If-else Statement

```
entity mux4to1 is
port( A,B,C,D: in std_logic;
            S: in std_logic_vector(1 downto 0);
           O: out std_logic);
end mux4to1:
Architecture behavioral of mux4to1 is
Begin
        Process(S,A,B,C,D)
                variable temp : std_logic;          // variable declaration
                Begin
                        if(S="00")then
                                temp:=A;
                        elsif(S="01")then               // note that it is 'elsif' not 'else if' of C
```

```
                    temp:=B;
                elsif(S="10")then
                    temp:=C;

                else
                    temp:=D;
                end if;                    // used to terminate the if statement
                O<=temp;                   // passing on the value of the variable
            end Process;
end behavioral;
```

MUX using different Statements

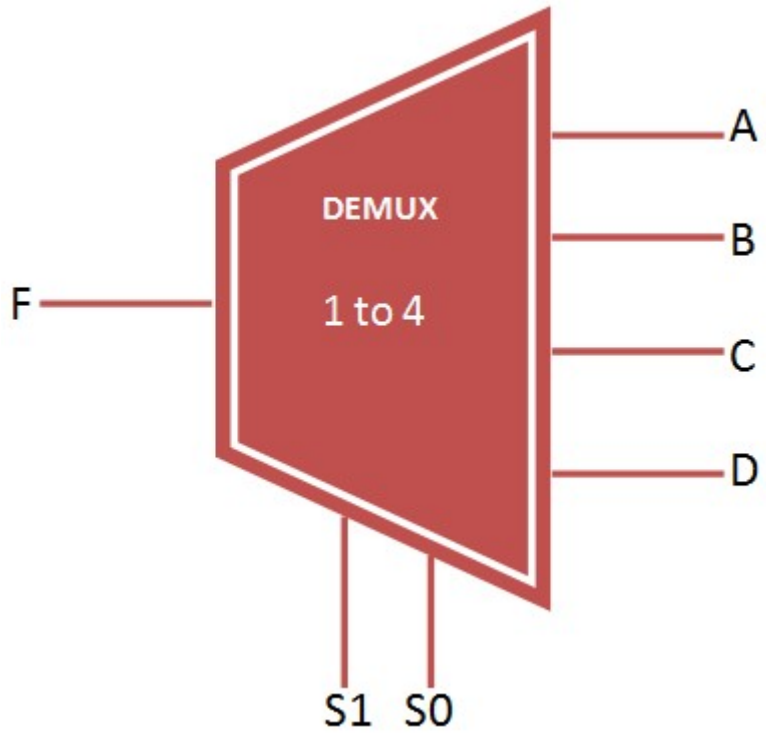
```
Library IEEE;
Use IEEE.std_logic_1164.all;
entity mux4to1 is
port( A,B,C,D: in std_logic;
        S: in std_logic_vector(1 downto 0);
        O: out std_logic);
end mux4to1;

Architecture behavioral of mux4to1 is
begin
Process(S,A,B,C,D)
variable temp:std_logic;
Begin
case S is
    when "00" => temp:=A;
    when "01" => temp:=B;
    when "10" => temp:=C;
    when Others => temp:=D;
end case;
O<=temp;
end Process;
end behavioral;
```

```
entity mux4to1 is
port( A,B,C,D,S1,S2: in std_logic;
                O: out std_logic);
end mux4to1;

Architecture behave of mux4to1 is
Begin

O<= A when (S1='0' and S2='0') else
    B when (S1='0' and S2='1') else
    C when (S1='1' and S2='0') else
    D;

end behavioral;
```

```
entity mux4to1 is
port(A,B,C,D,S1,S2: in std_logic;
                O: out std_logic);

end mux4to1;

Architecture behavioral of mux4to1 is

begin

with S select

O<= A when "00",
    B when "01",
    C when "10",
    D when others;
end behavioral;
```

DeMUX:

Demultiplexer (DEMUX) select one output from the multiple output line and fetch the single input through selection line. It consist of 1 input and 2 power n output. The output data lines are controlled by n selection lines. For Example, if n = 2 then the demux will be of 1 to 4 mux with 1 input, 2 selection line and 4 output as shown below. Also VHDL Code for 1 to 4 Demux described below.




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

entity demultiplexer1_4 is
    port(
        din : in STD_LOGIC;
        sel : in STD_LOGIC_VECTOR(1 downto 0);
        dout : out STD_LOGIC_VECTOR (3 downto
0)
        );
end demultiplexer1_4;

architecture demultiplexer1_4_arc of
demultiplexer1_4 is
begin

   demux : process (din,sel) is
   begin
      if (sel="00") then
         dout <= din & "000";
      elsif (sel="01") then
         dout <= '0' & din & "00";
      elsif (sel="10") then
```

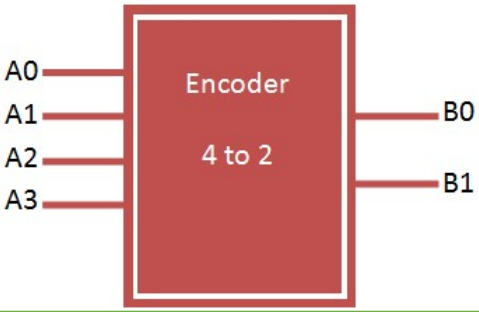
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity demultiplexer_case is
    port(
        din : in STD_LOGIC;
        sel : in STD_LOGIC_VECTOR(1 downto 0);
        dout : out STD_LOGIC_VECTOR(3 downto 0)
        );
end demultiplexer_case;


architecture demultiplexer_case_arc of
demultiplexer_case is
begin

   demux : process (din,sel) is
   begin
      case sel is
         when "00" => dout <= din & "000";
```

```
  dout <= "00" & din & '0';
      else
         dout <= "000" & din;
      end if;
   end process demux;

end demultiplexer1_4_arc;
```

```
when "01" => dout <= '0' & din & "00";
          when "10" => dout <= "00" & din & '0';
          when others => dout <= "000" & din;
       end case;
   end process demux;

end demultiplexer_case_arc;
```

Binary Encoder

Binary encoder has 2n input lines and n-bit output lines. It can be 4-to-2, 8-to-3 and 16-to-4 line configurations. VHDL Code for 4 to 2 encoder can be designed both in structural and behavioral modelling.



INPUT				OUTPUT	
A3	A2	A1	A0	B1	B0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity encoder is
 port(
 a : in STD_LOGIC_VECTOR(3 downto
0);
 b : out STD_LOGIC_VECTOR(1
downto 0)
 );
end encoder;
architecture bhv of encoder is
begin
process(a)
begin
 case a is
 when "1000" => b <= "00";
when "0100" => b <= "01";
when "0010" => b <= "10";
 when "0001" => b <= "11";
 when others => b <= "ZZ";
 end case;
end process;
end bhv;
```

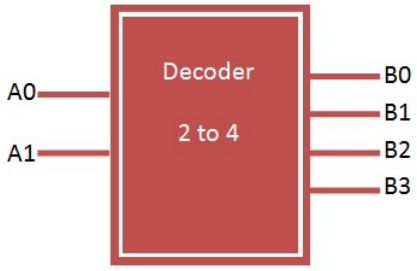
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity encoder1 is
 port(a : in STD_LOGIC_VECTOR(3 downto 0);
 b : out STD_LOGIC_VECTOR(1 downto 0) );
end encoder1;
architecture bhv of encoder1 is
begin
process(a)
begin
 if (a="1000") then
 b <= "00";
 elsif (a="0100") then
 b <= "01";
 elsif (a="0010") then
 b <= "10";
 elsif (a="0001") then
 b <= "11";
 else
 b <= "ZZ";
 end if;
end process;
end bhv;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity encoder2 is
 port(
 a : in STD_LOGIC_VECTOR(3 downto 0);
 b : out STD_LOGIC_VECTOR(1 downto 0) );
end encoder2;
architecture bhv of encoder2 is
begin
b(0) <= a(1) or a(2);
b(1) <= a(1) or a(3);
end bhv;
```

```
stim_proc: process
begin
-- hold reset state for 100 ns.
wait for 100 ns; a <= "0000";
wait for 100 ns; a <= "0001";
wait for 100 ns;
a <= "0010";
wait for 100 ns;
a <= "0100";
wait for 100 ns;
a <= "1000";
wait;
end process;
```

Decoder

Binary decoder has n-bit input lines and 2 power n output lines. It can be 2-to-4, 3-to-8 and 4-to-16 line configurations. Binary decoder can be easily constructed using basic logic gates. VHDL Code of 2 to 4 decoder can be easily implemented with structural and behavioral modelling.



INPUT		OUTPUT			
A1	A0	B3	B2	B1	B0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity decoder is
port(
a : in STD_LOGIC_VECTOR(1 downto 0);
b : out STD_LOGIC_VECTOR(3 downto 0)
);
end decoder;
architecture bhv of decoder is
begin
process(a)
begin
case a is
when "00" => b <= "0001";
when "01" => b <= "0010";
when "10" => b <= "0100";
when "11" => b <= "1000";
end case;
end process;
end bhv;
```

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

entity decoder2 is
 port(
 a : in STD_LOGIC_VECTOR(1 downto
0);
 b : out STD_LOGIC_VECTOR(3
downto 0)
 );
end decoder2;

architecture bhv of decoder2 is
begin

b(0) <= not a(0) and not a(1);
b(1) <= not a(0) and a(1);
b(2) <= a(0) and not a(1);
b(3) <= a(0) and a(1);

end bhv;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity decoder1 is
 port(
 a : in STD_LOGIC_VECTOR(1 downto 0);
 b : out STD_LOGIC_VECTOR(3 downto 0));
end decoder1;
architecture bhv of decoder1 is
begin
process(a)
begin
 if (a="00") then b <= "0001";
 elsif (a="01") then b <= "0010";
 elsif (a="10") then
 b <= "0100";
 else
 b <= "1000";
 end if;
end process;
end bhv;
```

```
stim_proc: process
 begin
 -- hold reset state for 100 ns.
 wait for 100 ns;
 a <= "00";
 wait for 100 ns;
 a <= "01";
 wait for 100 ns;
 a <= "10";
 wait for 100 ns;
 a <= "11";
 wait;
 end process;
END;
```

Comparator

Binary comparator compare two 4-bit binary number. It is also known as magnitude comparator and digital comparator. Analog form comparator is voltage comparator. The functionality of this comparator circuit is, It consist of 3 outputs Greater, Equal and Smaller. If inp-A is greater then inp-B then greater output is high, if both inp-A and inp-B are same then equal output is high, else smaller output is high.

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity VHDL_Binary_Comparator is
  port (
inp-A,inp-B   : in std_logic_vector(3 downto 0);
greater, equal, smaller  : out std_logic
   );
end VHDL_Binary_Comparator ;
architecture bhv of VHDL_Binary_Comparator
is
begin
greater <= '1' when (inp-A > inp-B)
else '0';
```

```
equal <= '1' when (inp-A = inp-B)
else '0';
smaller <= '1' when (inp-A < inp-B)
else '0';
end bhv;
```

```
Stim_proc:process

Begin

inp-A<=”0001”;

inp_B<=”1000”;

wait

end process Stim_proc;
```

Observation:

1) RTL for all combinational circuit.
2) Timing diagram for all of above combinational circuit.
3) Verify the output for input.

Discussion and conclusion:
Different problems encountered during lab are discussed here and the conclusion for the lab activity should be mentioned.

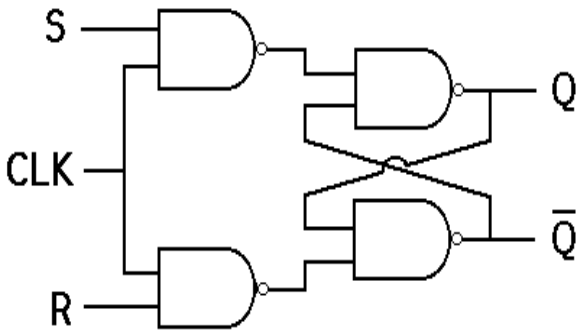
LAB 3

SEQUENTIAL CIRCUIT DESIGN USING VHDL

Objective: To design and implement SR, D, JK, T flip-flop and Counter using VHDL.

Theory: 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.

SR flipflop: A flip-flop circuit can be constructed from two NAND gates or two NOR gates. These flip-flops are shown in Figure. Each flip-flop has two outputs, Q and Q', and two inputs, set and reset. This type of flip-flop is referred to as an SR flip-flop.

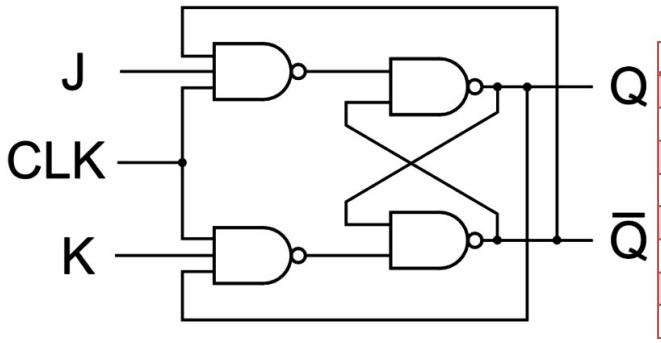


Q	S	R	Q(T+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	UNKNOWN
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	UNKNOWN

VHDL code for SR flipflop

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity SR_FF is
PORT( S,R,CLOCK,RST: in std_logic;
Q, QBAR: out std_logic);
end SR_FF;

Architecture behavioral of SR_FF is
begin
PROCESS(CLOCK ,RST)
variable tmp: std_logic;
if (RST='0') then
Q<='0';
elsif(CLOCK='1' and CLOCK'EVENT)
then
        if(S='0' and R='0')then
        tmp:=tmp;
        elsif(S='1' and R='1')then
                tmp:='Z';
        elsif(S='0' and R='1')then
                tmp:='0';
        else
                tmp:='1';
        end if;
        end if;
Q <= tmp;
QBAR <= not tmp;
end PROCESS;
end behavioral;
```

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

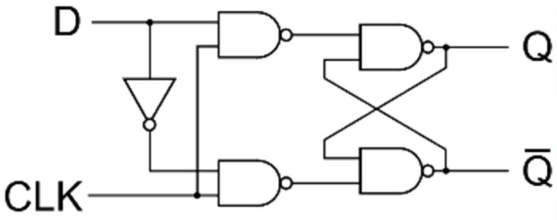


Q	J	K	Q(T+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

VHDL code for JK flipflop

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity JK_FF is
PORT( J,K,CLOCK: in std_logic;
Q, QB: out std_logic);
end JK_FF;

Architecture behavioral of JK_FF is
begin
        PROCESS(CLOCK)
        variable TMP: std_logic;
        begin
        if(CLOCK='1' and CLOCK'EVENT) then
                if(J='0' and K='0')then
                        TMP:=TMP;
                elsif(J='1' and K='1')then
                        TMP:= not TMP;
                elsif(J='0' and K='1')then
                        TMP:='0';
                else
                        TMP:='1';
                end if;
        end if;
Q<=TMP;
Q <=not TMP;
end PROCESS;
end behavioral;
```

D-Flipflop: The D flip-flop shown in figure is a modification of the clocked SR flip-flop. The D input goes directly into the S input and the complement of the D input goes to the R input. The D input is sampled during the occurrence of a clock pulse. If it is 1, the flip-flop is switched to the set state (unless it was already set). If it is 0, the flip-flop switches to the clear state.



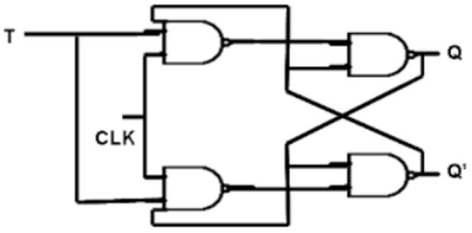
Q	D	Q(T+1)
0	0	0
0	1	1
1	0	0
1	1	1

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity D_FF is
PORT( D,CLOCK: in std_logic;
Q: out std_logic);
end D_FF;
```

```
architecture behavioral of D_FF is
begin
process(CLOCK)
begin
if(CLOCK='1'                    and
CLOCK'EVENT) then
Q <= D;
end if;
end process;
end behavioral;
```

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

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



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

entity T_FF is
port(       T: in std_logic;
Clock: in std_logic;
Q: out std_logic);
end T_FF;

architecture Behavioral of T_FF is
signal tmp: std_logic;
begin
process (Clock)
begin
if (Clock'event and Clock='1') then
if T='0' then
tmp <= tmp;
                    elsif T='1' then
                    tmp <= not (tmp);
            end if;
            end if;
            end process;
            Q <= tmp;
end Behavioral;
```

BCD counter:

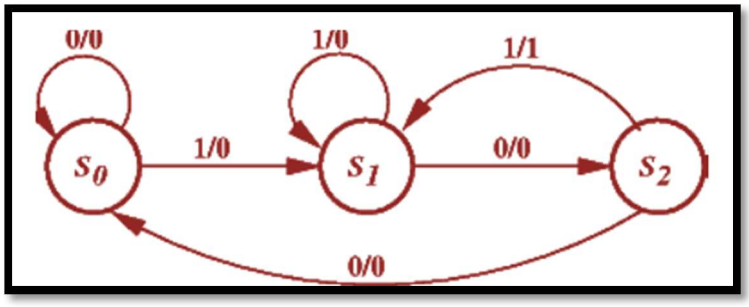
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Counter2_VHDL is
   port( Clock_enable_B: in std_logic;
            Clock: in std_logic;
            Reset: in std_logic;
            Output: out std_logic_vector(0 to 3));
end Counter2_VHDL;
 architecture Behavioral of Counter2_VHDL is
   signal temp: std_logic_vector(0 to 3);
            begin
            process(Clock,Reset)
begin
      if Reset='1' then
         temp <= "0000";
      elsif(rising_edge(Clock)) then
         if Clock_enable_B='0' then
            if temp="1001" then
               temp<="0000";
            else
               temp <= temp + 1;
            end if;
         end if;
      end if;
   end process;
   Output <= temp;
end Behavioral;
```

FSM design using VHDL

Objective: To implement Finite state machine using VHDL.

Theory: Sequential logic systems are finite state machines (FSMs). As FSMs, they consist of a set of states, some inputs, some outputs, and a set of rules for moving from state to state. When doing digital system design, it is very common to begin by defining how the system works with a finite state machine model. This design step allows the designer to think about the design from a high-level point of view without having to think much about what kind of hardware the system will be implemented on or what design tools will be required to implement the design. Once the FSM is fully designed, if it is designed well, it is easy to write out the design in a hardware description language (such as Verilog or VHDL) for implementation on a digital IC (integrated circuit).

Sequence detector for "101" using Melay Method:



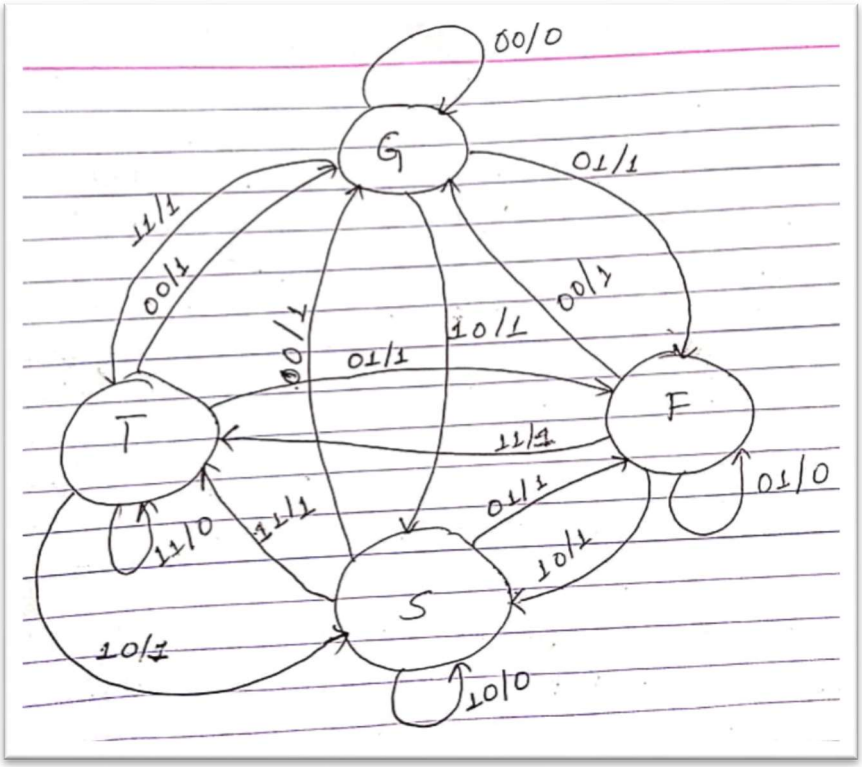

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity moore is
Port (      clk : in STD_LOGIC;
            din : in STD_LOGIC;
            rst : in STD_LOGIC;
            dout : out STD_LOGIC);
end moore;
architecture Behavioral of moore is
type state is (st0, st1, st2, st3);
signal present_state, next_state : state;
begin
            synchronous_process: process (clk)
            begin
            if rising_edge(clk) then
                        if (rst = '1') then
                        present_state <= st0;
                        else
                        present_state <= next_state;
                        end if;
            end if;
end process;
```

```
state:process(present_state, din)
begin
next_state <= st0;
case (present_state) is
when st0 =>
        if (din = '1') then
next_state <= st1;
else
next_state <= st0; end if;
        when st1 =>
                if (din = '1') then
                        next_state <= st1;
                        dout<='0';
        else
                        next_state <= st2;
                        dout<='0';
                end if;
         when st2 =>
                if (din = '1') then
                        next_state <= st3;
                        dout<='0';
                else
                        next_state <= st0;
                        dout<='0';
                 end if;
         when st3 =>
                if (din = '1') then
                        next_state <= st1;
                        dout<='0';
                else
                        next_state <= st2;
                        dout<='0';
                end if;
        when others =>
                next_state <= st0;
                dout<='0';
        end case;
end process;
```

Elevator design as an FSM:

Let us assume that we have three floor Ground Floor (G), first floor (F), Second Floor(S), Third Floor (T). We have two input switch with following configurations. SW="00" for Ground floor, SW="01" for First Floor, SW="10" for Second Floor and SW="11" for Third Floor.

From this assumption, we can draw state diagram as below.



Here output is assigned '1' if there is transition from one state and another state else output is '0'.

```
Library IEEE;
Use IEEE.std_logic_1164.all
Entity Elevator is
Port (CLK, RST: in std_logic;
        X: in std_logic_vector(1 downto 0);
        Y:output std_logic_vector);
Architecture Elevator_arch of Elevator is
Type state is (G,F,S,T);
Signal PS,NS:state;
Begin
        Sync_proc:process(CLK ,RST)
                Begin
                        If (RST='1') then
                                PS<='G';
                        Elsif(Rising_edge(CLK)) then
                                PS<=NS;
                        End if;
                End Process Sync_process;
```

```
Comb_proc:Process(PS,X)
        Begin
            Case PS is
                When G=>
                    If(X=”00”) then
                        NS<=G;
                        Z<=’0’;
                    Elsif(X=’01’) then
                        NS<=F;
                        Z<=’1’;
                    Elsif(X=’10’) then
                        NS<=S;
                        Z<=’1’;
                    Else
                        NS<=T;
                        Z<=’1’;
                    End if;
                    When F=>
                    If(X=”00”) then
                        NS<=G;
                        Z<=’1’;
                    Elsif(X=’01’) then
                        NS<=F;
                        Z<=’0’;
                    Elsif(X=’10’) then
                        NS<=S;
                        Z<=’1’;
```

```
                Else
                        NS<=T;
                        Z<='1';
                End if;
        When S=>
                If(X="00") then
                        NS<=G;
                        Z<='1';
                Elsif(X='01') then
                        NS<=F;
                        Z<='1';
                Elsif(X='10') then
                        NS<=S;
                        Z<='0';
                Else
                        NS<=T;
                        Z<='1';
                End if;
        When T=>
                If(X="00") then
                        NS<=G;
                        Z<='1';
                Elsif(X='01') then
                        NS<=F;
                        Z<='1';
```

```
                    Elsif(X='10') then
                        NS<=S;
                        Z<='1';
                    Else
                        NS<=T;
                        Z<='1';
                    End if;
                End case;
            End process comb_proc;
        End Elevator_arch;
```

- The test bench includes the value of CLK, RST and X and obtain the timing diagram.

Observation:

- The RLT of the design and Timing diagram to verify the output with respective input.

Discussion and Conclusion:

Lab 5

Programming Microcontroller

Objective: To program 8051 microcontroller using IDE and simulate its output in Proteus.

Theory:

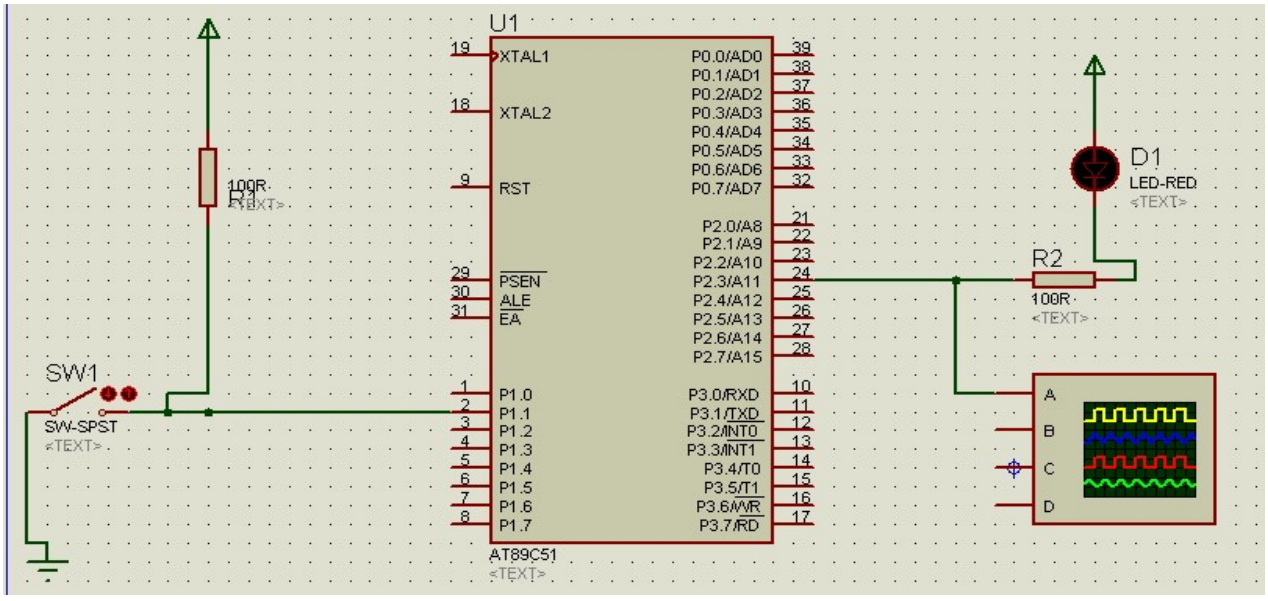
- Microcontrollers**,** 8051 microcontroller features**,** instruction set and addressing mode.

Programming:

Task 1: To generate the duty cycle of 50% using assembly.

```
        ORG 00H
        CLR P2.3
        BACK:
                        CLR P2.3
                        LCALL DELAY
                        SETB P2.3
                        LCALL DELAY
                        SJMP BACK
        ORG 300H
                DELAY:
                        MOV R5, #64H
        AGAIN:  MOV R4, #0FFH
        AGAN:   MOV R3, #08H
        AGA:  DJNZ R3, AGA
                        DJNZ R4, AGAN
                        DJNZ R5, AGAIN
                RET
END
```

Simulation:



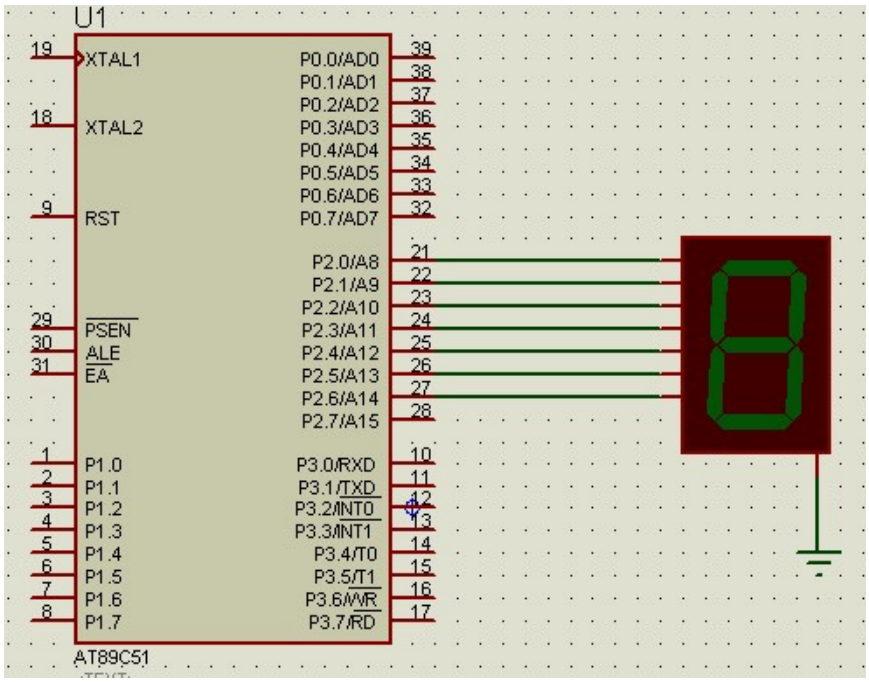
Task 2: Toggle the LEDs ON and OFF (Blinking LEDs) that are connected to PORT1 of the 8051 Microcontroller.

```
        ORG 00H                 ; Assembly Starts from 0000H.
        ; Main Program
        START:  MOV P1, #FFH            ; Move 11111111 to PORT1.
                CALL WAIT  ; Call WAIT
                MOV A, P1    ; Move P1 value to ACC
                CPL A               ; Complement ACC
                MOV P1, A    ; Move ACC value to P1
                CALL WAIT  ; Call WAIT
                SJMP START          ; Jump to START
        WAIT:   MOV R2, #10         ; Load Register R2 with 10 (0x0A)
        WAIT1:  MOV R3, #200                ; Load Register R3 with 10 (0xC8)
        WAIT2:  MOV R4, #200                ; Load Register R4 with 10 (0xC8)
                DJNZ R4, $    ; Decrement R4 till it is 0. Stay there if not 0.
                DJNZ R3, WAIT2      ; Decrement R3 till it is 0. Jump to WAIT2 if not 0.
                DJNZ R2, WAIT1      ; Decrement R2 till it is 0. Jump to WAIT1 if not 0.
                RET             ; Return to Main Program
        END
```

Task 3: Interfacing Seven Segment with 8051 microcontroller.

```
        ORG 00H
        MOV P2, #00H
        MOV R6,#00H
        MOV DPTR, #DIGITS
        MAIN:
        MOV A, R6
        MOVC A, @A+DPTR
        MOV P2, A
        LCALL DELAY
        INC R6
        CJNE R6,#0AH, MAIN
        MOV R6, #00H
        SJMP MAIN
        DELAY: MOV R3, #0F0H
        DEL1: MOV R2, #0FAH
        DEL2: DJNZ R2, DEL2
        DJNZ R3, DEL1
        RET
        DIGITS:
                DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 6FH
END
```

Simulation



Discussion and conclusion: