# Combinational Logic Circuit Design and VHDL Implementation of -

All the following codes are implemented using some common basic coding and each has an alternation of its own in regard to how the code is required to work . That is all codes have an **Entity** and an **Architecture .**

The entity is an external view of any design and it is so named as it shows how the entire system is connected to the outside or other modules . This remains same for all codes as can be seen in all these codes as it only shows the names of both input and output , besides in and out , buffer , linkage , inout may also be used . They will be explained further if and when they appear in any of the following codes .

As for the Architecture , it is also called the internal view of a design as it consists of the proper definition of the design as

1) An interconnection of components (Structural)

2) Flow of data in the design (Dataflow)

3) The behaviour of the hardware (Behavioural)

Unless specified with certain codes or within process statement in behavioural coding , all commands are concurrent i.e they will be performed simultaneously until told otherwise . However in the process statement , all commands are performed sequentially .

## HALF ADDER

```
library IEEE;

--This command line depicts the usage of the library IEEE

use IEEE.STD_LOGIC_1164.ALL;

  --This command line is used to tell which part of IEEE is to be used i.e STD_LOGIC_1164 and
ALL signifies all content of this part of IEEE library so all functions , types , packages ,
subpackages etc within it can be used .

entity HA is --Entity Declaration

   port ( A,B : in BIT ;

                  SUM,CARRY : out BIT ) ; -- Assignment of ports
```

```
end HA ;

architecture FUNCT of HA is     -- Architecture declaration
begin
   SUM <= A xor B after 8ns ; -- Sum signal is given the value of (A xor B) after 8 ns
   COUT <= A and B after 4ns ;
-- After command dictates the time period after which the signal is to be assigned the RHS value
end FUNCT ;
```

Hence a Half adder is created using VHDL language , where the input is taken as a,b and output is SUM and CARRY which are (A xor B) and (A and B) respectively . Thereby giving the sum and any carry in the output without considering any previous carries .

## FULL ADDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FA is
   port ( A,B,CIN : in BIT ;
                  SUM,COUT : out BIT ) ;
end FA ;

architecture FUNCT of FA is
begin
   SUM <= A xor B xor CIN after 15ns ;
   COUT <= (A and B) or (B and CIN) or (CIN and A) after 10ns ;
```

```
end FUNCT ;
```

The functions of all commands used here have already been explained . This VHDL code is used to take the input A,B,CIN and give output SUM and COUT which are A xor B xor CIN and ((A and B) or (B and CIN) or (CIN and A)) respectively . Hence giving a sum and carry at the output after considering any previous carries .

## HALF SUBTRACTOR

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity HS is

        Port ( a,b : in    BIT;

                          DIFF,BORROW : out    BIT );

end HS ;


architecture FUNCT of HS is

  begin

      DIFF <= a xor b;

      BORROW <= (not a) and b;

end FUNCT;
```

This half subtractor VHDL code has 2 inputs a , b and 2 outputs DIFF(Difference) and BORROW which are (a xor b) and ((not a) and b) respectively . Thereby giving the difference and any borrow in the output without considering any previous borrows .

## FULL SUBTRACTOR

```
library IEEE ;

use IEEE.STD_LOGIC_1164.ALL ;

entity FS is

port (a,b,c : in BIT ;

                  DIFF,BORROW : out BIT );

end FS ;


architecture FUNCT of FS is

  begin

    DIFF<=a xor b xor c ;

    BORROW <= ((not a) and b) or ( b and c ) or ( c and (not a)) ;

end FUNCT ;
```

This Full subtractor VHDL code has 3 inputs a , b , c( previous borrow ) and 2 outputs DIFF(Difference) and BORROW which are (a xor b xor c ) and ((not a) and b) or ( b and c ) or ( c and (not a)) respectively . Thereby giving the difference and any borrow in the output after considering any previous borrows .

## 4:2 ENCODER

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity ENC is

port ( I0,I1,I2,I3 : in BIT ;

                  O0,O1 : out BIT );

end ENC ;
```

```
architecture FUNCT of ENC is
  begin
      O0    <=    I3 or I1 ;
      O1    <=    I2 or I0 ;
end FUNCT ;
```

This Encoder VHDL code has 4 inputs I0 , I1 , I2 , I3 and 2 outputs O0 and O1 which are (I3 or I1) and (I2 and I0) respectively . Hence depending on the input , the output is binary . like 00 for 0001 , 01 for 0010 and so on.

## 2:4 DECODER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity DEC is
port ( I0,I1 : in BIT ;
                  O0,O1,O2,O3 : out BIT );
end DEC ;


architecture FUNCT of DEC is
  begin
      O0    <=    (not I0) or (not I1) ;
      O1    <=    I0 or (not I1) ;
      O2    <= (not I0) or I1 ;
      O3    <= I0 or I1 ;
end FUNCT ;
```

This Decoder VHDL code has 2 inputs I0 and I1 and 4 outputs O0 , O1 , O2 , O3 which are ((not

I0) or (not I1)) , (I0 or (not I1) ) , ((not I0) or I1 ) and (I0 or I1) respectively . Hence the output is 1 on the pin given by binary input . O0 pin if input is 00 ,O1 pin if input is 01 , etc.

## 4:1 MULTIPLEXER

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity MUX is

port ( I0,I1,I2,I3,S0,S1    : in BIT ;

                   O : out BIT );

end DEC ;


architecture FUNCT of MUX is

  begin

      O <=    (((not S0) or (not S1)) and I0) or ((S0 or (not S1)) and I1) or (((not S0) or S1) and I2) or
((S0 or S1) and I3) ;

end FUNCT ;
```

This Multiplexer VHDL code has 4 inputs I0 , I1 , I2 , I3 and 2 selection inputs S0 and S1 and 1 output O which has the value (((not S0) or (not S1)) and I0) or ((S0 or (not S1)) and I1) or (((not S0) or S1) and I2) or ((S0 or S1) and I3). Hence giving input at a desired output pin .

## ALU

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity ALU is
```

```
  Port ( a,b : in BIT_VECTOR (3 downto 0);
                    sel : in BIT_VECTOR (2 downto 0);
                  out : out BIT_VECTOR (3 downto 0) );
end ALU ;


architecture FUNCT of ALU is
  begin
    process(a, b, sel)
  -- Process statement is a statement that has its own sensitivity list i.e a,b,sel in this case . If any
of these signals is activated or enabled , the process statement and its contents start getting
executed sequentially , a process is never eliminated and is only suspended until any of these
signals is enabled again
      begin    -- Process block begins
            case sel is
-- Case statement is used to check some value (sel in this case) and when it has a        certain
value , a certain value is assigned to out as shown .
            when "000" =>
            out <=    a + b; --addition
            when "001" =>
            out <=    a - b; --subtraction
            when "010" =>
            out <= a - 1; --sub 1
            when "011" =>
            out <= a + 1; --add 1
            when "100" =>
            out <=    a and b; --AND gate
            when "101" =>

            out <= a or b; --OR gate

            when "110" =>

            out <= not a ; --NOT gate

            when "111" =>

            out <= a xor b; --XOR gate

            when others =>

            NULL; --No action is to be taken

        end case;     -- end of case statement

    end process;     -- end of process block

end FUNCT;
```

This ALU VHDL code has 3 inputs a (4 bit word) ,b (4 bit word) , sel (3 bit word) and 1 output O (4 Bit word) which have values as shown and explained . Hence performing arithmetic and logic operations on input a and b or either one .

## BARREL SHIFTER

```
library IEEE;

use IEEE.STD_LOGIC_ALL.1164 ;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity BARREL is

    port (

        d_in               : in    BIT_VECTOR(7 downto 0);      -- input vector
        d_out             : out BIT_VECTOR(7 downto 0);      -- shifted output

        shift_lt_rt : in    BIT;

          -- 0=>left_operation    1=>right_operation

        shift_by        : in    BIT_VECTOR(2 downto 0);      -- shift amount

        clk , rst_a , p_load                 : in    BIT );

                                        --    clock signal ,reset signal and Parallel load

end barrel_shifter;


architecture FUNCT of BARREL is

begin                   -- FUNCT

  p1: process (clk,rst_a,shift_by,shift_lt_rt)

variable x,y : BIT_VECTOR(7 downto 0);      -- Variables a and y are declared

variable ctrl0,ctrl1,ctrl2 : BIT_VECTOR(1 downto 0);


            begin    -- process p1

            ctrl0 := shift_by(0) & shift_lt_rt;

                            -- control 0 is given value    Shift_by(0) and Shift_lt_rt

            ctrl1 := shift_by(1) & shift_lt_rt;

            ctrl2 := shift_by(2) & shift_lt_rt;


                    if(rst_a = '1') then

                    d_out<="00000000"; -- Reset the output to default value


            elsif ( clk'event and clk = '1' ) then    -- If positive edge triggering occurs

            if (p_load='0') then
```

```
    assert(false) report "Parallel load low" severity warning;

                                        -- Prompt appears to show parallel load is low

elsif (shift_lt_rt='1') then

    assert(false) report "right shift" severity warning;

                                        -- Prompt appears to show Shift towards right is enabled for
barrel shifter

elsif(shift_lt_rt='0')then

    assert(false) report "left shift" severity warning;

                                        -- Prompt appears to show Shift towards left is enabled for
barrel shifter

end if;


if p_load='1' then

                -- execute following statements if parallel load is high

    case ctrl0 is

          when "00"|"01" => x:=d_in ; -- x is assigned d_in if ctrl 0 is 00 or 01

          when "10" =>    x:=d_in(6 downto 0) & d_in(7);

                                            --shift left by 1 bit (& defines concatenation )

          when "11" =>    x:=d_in(0) & d_in(7 downto 1);    --shift right by 1 bit

          when others => null;

      end case;


              case ctrl1 is

              when "00"|"01" =>y:=x;

              when "10" =>y:=x(5 downto 0) & x(7 downto 6);    --shift left by 2 bits
```

```
                    when "11" =>y:=x(1 downto 0) & x(7 downto 2);    --shift right by 2 bits

                    when others => null;

                    end case;


                  case ctrl2 is

          when "00"|"01" =>d_out<=y ;

         when "10"|"11" =>d_out<= y(3 downto 0) & y(7 downto 4);

                                             --shift right/left by 4 bits

         when others => null;

          end case;

        end if;

       end if;

   end process p1;

end beh;
```

Hence in the above VHDL code , a barrel shifter is created that can shift input word by 1 ,2 or 4 bits in either direction .

## MULTIPLIER

```
library ieee;

use ieee.std_logic_1164.all;


entity MUL is
```

```
port ( fi : in BIT_VECTOR;                -- Input signal fi
                    f0 : out BIT_VECTOR );       -- fo = 2*fi
end MUL;

architecture FUNC of MUL is
signal q,clk : BIT ;
begin
            process(fi)      -- As soon as fi is activated , start process block
                        begin
                                    if rising_edge(clk) then --At rising edge of clock
                                    q<= not q;
                                    end if;
                                    clk <= not (q xor fi); -- When q and fi are same ,clk is given value 1
                                    f0 <= clk;
            end process;
end FUNCT;
```

Hence the frequency of input signal is multiplied by 2

## DIVIDER

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

entity Clock_Divider is
port ( clk,reset: in BIT;
clock_out: out BIT);
end Clock_Divider;


architecture FUNCT of Clock_Divider is
signal count: integer := 0;    --count is given integer value of 0
signal tmp : std_logic := '1';
   begin
      process(clk,reset)
       begin
       if(reset='0') then
       count<=0;
       tmp<='1';
       elsif(clk'event and clk='1') then -- When clock has an event and becomes 1
       count <=count+1;
       if (count = 25000) then
       tmp <= NOT tmp;
       count <= 0;
         end if;
       end if;
   clock_out <= tmp; -- clock_out is given the value temp
  end process;
end FUNCT;
```

Hence clock time gets divided_

# HAMMING CODE ENCODER

```
library ieee;

use ieee.std_logic_1164.all;

Entity hamming_encoder is

  port( datain : IN BIT_VECTOR(0 TO 3); --d0 d1 d2 d3

                       hamout : OUT BIT_VECTOR(0 TO 6)); --d0 d1 d2 d3 p0 p1 p2

end hamming_encoder;


architecture FUNCT of hamming_encoder is

SIGNAL p0, p1, p2 : BIT;

begin

--generate check bits

  p0 <= (datain(0) XOR datain(1)) XOR datain(3);

  p1 <= (datain(0) XOR datain(2)) XOR datain(3);

  p2 <= (datain(1) XOR datain(2)) XOR datain(3);

--connect outputs

hamout(4 TO 6) <= (p0, p1, p2);        -- last 3 bits of HAMOUT are p0,p1,p2

hamout(0 TO 3) <= datain(0 TO 3);

end funct;
```

Hence the above code creates 7 bit hamming code from 4 bit input with the first 4 bits same as input and last 3 (p0 ,p1 ,p2) as (datain(0) XOR datain(1)) XOR datain(3) ,    (datain(0) XOR datain(2)) XOR datain(3) and    (datain(1) XOR datain(2)) XOR datain(3) respectively .