1. **OR gate.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity OR\_Gate is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Y : out STD\_LOGIC -- Output Y

);

end OR\_Gate;

-- Architecture Definition

architecture Behavioral of OR\_Gate is

begin

-- Process to describe OR Gate behavior

Y <= A or B; -- OR operation

end Behavioral;

1. **AND gate.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity AND\_Gate is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Y : out STD\_LOGIC -- Output Y

);

end AND\_Gate;

-- Architecture Definition

architecture Behavioral of AND\_Gate is

begin

-- Process to describe AND Gate behavior

Y <= A and B; -- AND operation

end Behavioral;

1. **NOT gate.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity NOT\_Gate is

Port (

A : in STD\_LOGIC; -- Input A

Y : out STD\_LOGIC -- Output Y

);

end NOT\_Gate;

-- Architecture Definition

architecture Behavioral of NOT\_Gate is

begin

-- Process to describe NOT Gate behavior

Y <= not A; -- NOT operation

end Behavioral;

1. **NOR gate.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity NOR\_Gate is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Y : out STD\_LOGIC -- Output Y

);

end NOR\_Gate;

-- Architecture Definition

architecture Behavioral of NOR\_Gate is

begin

-- Process to describe NOR Gate behavior

Y <= not (A or B); -- NOR operation

end Behavioral;

1. **NAND gate.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity NAND\_Gate is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Y : out STD\_LOGIC -- Output Y

);

end NAND\_Gate;

-- Architecture Definition

architecture Behavioral of NAND\_Gate is

begin

-- Process to describe NAND Gate behavior

Y <= not (A and B); -- NAND operation

end Behavioral;

1. XOR gate.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity XOR\_Gate is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Y : out STD\_LOGIC -- Output Y

);

end XOR\_Gate;

-- Architecture Definition

architecture Behavioral of XOR\_Gate is

begin

-- Process to describe XOR Gate behavior

Y <= A xor B; -- XOR operation

end Behavioral;

1. **XNOR gate.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity XNOR\_Gate is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Y : out STD\_LOGIC -- Output Y

);

end XNOR\_Gate;

-- Architecture Definition

architecture Behavioral of XNOR\_Gate is

begin

-- Process to describe XNOR Gate behavior

Y <= not (A xor B); -- XNOR operation

end Behavioral;

1. **HALF ADDER .**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity Half\_Adder is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Sum : out STD\_LOGIC; -- Sum output

Carry : out STD\_LOGIC -- Carry output

);

end Half\_Adder;

-- Architecture Definition

architecture Behavioral of Half\_Adder is

begin

-- Process to describe Half Adder behavior

Sum <= A xor B; -- Sum is A XOR B

Carry <= A and B; -- Carry is A AND B

end Behavioral;

1. **FULL ADDER.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity Full\_Adder is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Cin : in STD\_LOGIC; -- Carry input

Sum : out STD\_LOGIC; -- Sum output

Cout : out STD\_LOGIC -- Carry output

);

end Full\_Adder;

-- Architecture Definition

architecture Behavioral of Full\_Adder is

begin

-- Process to describe Full Adder behavior

Sum <= A xor B xor Cin; -- Sum is A XOR B XOR Cin

Cout <= (A and B) or (B and Cin) or (Cin and A); -- Carry is the OR of partial carries

end Behavioral;

1. **HALF SUBTRACTOR.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity Half\_Subtractor is

Port (

A : in STD\_LOGIC; -- Input A (minuend)

B : in STD\_LOGIC; -- Input B (subtrahend)

Diff : out STD\_LOGIC; -- Difference output

Borrow : out STD\_LOGIC -- Borrow output

);

end Half\_Subtractor;

-- Architecture Definition

architecture Behavioral of Half\_Subtractor is

begin

-- Process to describe Half Subtractor behavior

Diff <= A xor B; -- Difference is A XOR B

Borrow <= not A and B; -- Borrow is NOT A AND B

end Behavioral;

1. **FULL SUBTRACTOR.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration

entity Full\_Subtractor is

Port (

A : in STD\_LOGIC; -- Input A (minuend)

B : in STD\_LOGIC; -- Input B (subtrahend)

Bin : in STD\_LOGIC; -- Borrow input

Diff : out STD\_LOGIC; -- Difference output

BorrowOut : out STD\_LOGIC -- Borrow output

);

end Full\_Subtractor;

-- Architecture Definition

architecture Behavioral of Full\_Subtractor is

begin

-- Process to describe Full Subtractor behavior

Diff <= A xor B xor Bin; -- Difference is A XOR B XOR Bin

BorrowOut <= (not A and B) or (not A and Bin) or (B and Bin); -- Borrow is calculated

end Behavioral;

1. **Make XOR with help of NAND gate.(structural).**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration for NAND Gate

entity NAND\_Gate is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Y : out STD\_LOGIC -- Output Y

);

end NAND\_Gate;

-- Architecture for NAND Gate

architecture Behavioral of NAND\_Gate is

begin

Y <= A nand B; -- Basic NAND operation

end Behavioral;

-- Entity Declaration for XOR Gate Using NAND Gates

entity XOR\_Gate\_NAND is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Y : out STD\_LOGIC -- Output Y (XOR result)

);

end XOR\_Gate\_NAND;

-- Architecture for XOR Gate Using NAND Gates

architecture Structural of XOR\_Gate\_NAND is

-- Intermediate signals for NAND operations

signal nand1, nand2, nand3, nand4 : STD\_LOGIC;

-- Component declaration for NAND Gate

component NAND\_Gate

Port (

A : in STD\_LOGIC;

B : in STD\_LOGIC;

Y : out STD\_LOGIC

);

end component;

begin

-- NAND operations to construct XOR logic

U1: NAND\_Gate port map (A, B, nand1); -- nand1 = A nand B

U2: NAND\_Gate port map (A, nand1, nand2); -- nand2 = A nand nand1

U3: NAND\_Gate port map (B, nand1, nand3); -- nand3 = B nand nand1

U4: NAND\_Gate port map (nand2, nand3, Y); -- Y = nand2 nand nand3 (Final XOR)

end Structural;

1. **HALF Adder using NAND gate.(structural).**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration for NAND Gate

entity NAND\_Gate is

Port (

A : in STD\_LOGIC;

B : in STD\_LOGIC;

Y : out STD\_LOGIC

);

end NAND\_Gate;

-- Architecture for NAND Gate

architecture Behavioral of NAND\_Gate is

begin

Y <= A nand B;

end Behavioral;

-- Entity Declaration for Half Adder

entity Half\_Adder\_NAND is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

Sum : out STD\_LOGIC; -- Sum output

Carry : out STD\_LOGIC -- Carry output

);

end Half\_Adder\_NAND;

-- Architecture for Half Adder using NAND Gates

architecture Structural of Half\_Adder\_NAND is

-- Signals for intermediate NAND gates

signal nand1, nand2, nand3, nand4 : STD\_LOGIC;

-- Component declarations for the NAND gates

component NAND\_Gate

Port (

A : in STD\_LOGIC;

B : in STD\_LOGIC;

Y : out STD\_LOGIC

);

end component;

begin

-- Sum Logic: A ⊕ B using NAND gates

U1: NAND\_Gate port map (A, B, nand1); -- nand1 = A nand B

U2: NAND\_Gate port map (A, nand1, nand2); -- nand2 = A nand nand1

U3: NAND\_Gate port map (B, nand1, nand3); -- nand3 = B nand nand1

U4: NAND\_Gate port map (nand2, nand3, Sum); -- Sum = nand2 nand nand3

-- Carry Logic: A AND B using NAND gates

U5: NAND\_Gate port map (A, B, nand4); -- nand4 = A nand B (Carry)

U6: NAND\_Gate port map (nand4, nand4, Carry); -- Carry = nand4 nand nand4 (Final carry)

end Structural;

1. **Multiplexer (MUX)(4 to 1)(behavioural).**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity mux4to1 is

Port (

A : in STD\_LOGIC; -- Input A

B : in STD\_LOGIC; -- Input B

C : in STD\_LOGIC; -- Input C

D : in STD\_LOGIC; -- Input D

S : in STD\_LOGIC\_VECTOR(1 downto 0); -- Select lines

Y : out STD\_LOGIC -- Output Y

);

end mux4to1;

architecture Behavioral of mux4to1 is

begin

process(A, B, C, D, S)

begin

case S is

when "00" =>

Y <= A; -- Select input A

when "01" =>

Y <= B; -- Select input B

when "10" =>

Y <= C; -- Select input C

when "11" =>

Y <= D; -- Select input D

when others =>

Y <= '0'; -- Default case (optional)

end case;

end process;

end Behavioral;

1. **VHDL code for DEMULTIPLEXURE(DEMUX)(1 to 4).**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration for 1-to-4 Demultiplexer

entity DEMUX\_1to4 is

Port (

D : in STD\_LOGIC; -- Input D

Sel : in STD\_LOGIC\_VECTOR(1 downto 0); -- 2-bit Select signal

Y0 : out STD\_LOGIC; -- Output Y0

Y1 : out STD\_LOGIC; -- Output Y1

Y2 : out STD\_LOGIC; -- Output Y2

Y3 : out STD\_LOGIC -- Output Y3

);

end DEMUX\_1to4;

-- Architecture for 1-to-4 Demultiplexer

architecture Behavioral of DEMUX\_1to4 is

begin

-- DEMUX Logic: Based on the select signal, route input D to the correct output

process (D, Sel)

begin

-- Default outputs are 0

Y0 <= '0';

Y1 <= '0';

Y2 <= '0';

Y3 <= '0';

-- Routing the input D to one of the outputs based on Sel

case Sel is

when "00" =>

Y0 <= D; -- Output Y0

when "01" =>

Y1 <= D; -- Output Y1

when "10" =>

Y2 <= D; -- Output Y2

when "11" =>

Y3 <= D; -- Output Y3

when others =>

-- Default case (should not happen)

Y0 <= '0';

end case;

end process;

end Behavioral;

1. **ENCODER.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration for 4-to-2 Encoder

entity Encoder\_4to2 is

Port (

I : in STD\_LOGIC\_VECTOR(3 downto 0); -- 4 input lines

Y : out STD\_LOGIC\_VECTOR(1 downto 0); -- 2-bit output

Valid : out STD\_LOGIC -- Valid output signal (indicates valid encoding)

);

end Encoder\_4to2;

-- Architecture for 4-to-2 Encoder

architecture Behavioral of Encoder\_4to2 is

begin

-- Encoding Logic

process (I)

begin

if (I = "0000") then

Y <= "00"; -- No input is active

Valid <= '0'; -- Invalid output

elsif (I(0) = '1') then

Y <= "00"; -- Input 0 active

Valid <= '1'; -- Valid encoding

elsif (I(1) = '1') then

Y <= "01"; -- Input 1 active

Valid <= '1'; -- Valid encoding

elsif (I(2) = '1') then

Y <= "10"; -- Input 2 active

Valid <= '1'; -- Valid encoding

elsif (I(3) = '1') then

Y <= "11"; -- Input 3 active

Valid <= '1'; -- Valid encoding

else

Y <= "00"; -- Default case

Valid <= '0'; -- Invalid output

end if;

end process;

end Behavioral;

1. **1-bit comparator.(dataflow).**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration for 1-bit Comparator

entity Comparator\_1bit is

Port (

A : in STD\_LOGIC; -- 1-bit input A

B : in STD\_LOGIC; -- 1-bit input B

A\_equal\_B : out STD\_LOGIC; -- Output for A = B

A\_greater\_B : out STD\_LOGIC; -- Output for A > B

A\_less\_B : out STD\_LOGIC -- Output for A < B

);

end Comparator\_1bit;

-- Architecture for 1-bit Comparator (Dataflow)

architecture Dataflow of Comparator\_1bit is

begin

-- Dataflow assignments for comparator logic

A\_equal\_B <= (A = B); -- A equals B

A\_greater\_B <= (A = '1' and B = '0'); -- A > B

A\_less\_B <= (A = '0' and B = '1'); -- A < B

end Dataflow;

1. **SET RESET flip flop.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.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(S,R,rst,CLOCK)

begin

if(rst='1')then

Q<='0';

elsif(CLOCK='1' and CLOCK'EVENT) then

if(S/=R)then

Q<=S;

QBAR<=R;

elsif(S='1' and R='1')then

Q<='Z';

QBAR<='Z';

end if;

end if;

end PROCESS;

end behavioral;

1. **D FLIP FLOP.**

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, rst: in std\_logic;

Q, QBAR: out std\_logic);

end D\_FF;

Architecture behavioral of D\_FF is

begin

PROCESS(D,rst,CLOCK)

begin

if(rst='1')then

Q<='0';

elsif(CLOCK='1' and CLOCK'EVENT) then

Q<=D;

QBAR<=not D;

end if;

end PROCESS;

end behavioral;

1. **J-K flip flop.**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity Declaration for JK Flip-Flop

entity JKFF is

Port (

J : in STD\_LOGIC; -- J input

K : in STD\_LOGIC; -- K input

CLK : in STD\_LOGIC; -- Clock input

Q : out STD\_LOGIC; -- Output Q

Q\_bar : out STD\_LOGIC -- Output Q\_bar (inverted Q)

);

end JKFF;

-- Architecture for JK Flip-Flop

architecture Behavioral of JKFF is

begin

-- Process to model the JK Flip-Flop behavior

process (CLK)

begin

if rising\_edge(CLK) then -- Trigger on the rising edge of the clock

case (J & K) is

when "00" =>

-- Hold the current state (no change)

Q <= Q;

Q\_bar <= not Q;

when "01" =>

-- Reset state (Q = 0)

Q <= '0';

Q\_bar <= '1';

when "10" =>

-- Set state (Q = 1)

Q <= '1';

Q\_bar <= '0';

when "11" =>

-- Toggle the state (Q changes)

Q <= not Q;

Q\_bar <= not Q;

when others =>

-- Default case (should not happen)

Q <= 'X';

Q\_bar <= 'X';

end case;

end if;

end process;

end Behavioral;

1. **T flip flop.**

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