1.OR_Gate

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

-- Entity declaration
entity OR_Gate is
   Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          Y : out STD_LOGIC);
end OR_Gate;

-- Architecture definition
architecture Behavioral of OR_Gate is
begin
   -- OR gate logic
   Y <= A or B;
end Behavioral;
```

2.AND_Gate

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

-- Entity declaration for AND gate
entity AND_Gate is
   Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          Y : out STD_LOGIC);
end AND_Gate;

-- Architecture definition
architecture Behavioral of AND_Gate is
begin
   -- AND gate logic
   Y <= A and B;

end Behavioral;
```

3.NOT_Gate

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


-- Entity declaration for NOT gate

entity NOT_Gate is

   Port ( A : in  STD_LOGIC;

          Y : out STD_LOGIC);

end NOT_Gate;


-- Architecture definition

architecture Behavioral of NOT_Gate is

begin

   -- NOT gate logic (inverter)

   Y <= not A;

end Behavioral;
```

4.NAND_Gate

```
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 definition
architecture Behavioral of NAND_Gate is
begin
   -- NAND gate logic
   Y <= not (A and B);
end Behavioral;
```

5.NOR_Gate

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

-- Entity declaration for NOR gate
entity NOR_Gate is
   Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          Y : out STD_LOGIC);
end NOR_Gate;

-- Architecture definition
architecture Behavioral of NOR_Gate is
begin
   -- NOR gate logic
   Y <= not (A or B);
end Behavioral;
```

6.XOR_Gate

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

-- Entity declaration for XOR gate
entity XOR_Gate is
   Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          Y : out STD_LOGIC);
end XOR_Gate;

-- Architecture definition
architecture Behavioral of XOR_Gate is
begin
   -- XOR gate logic
   Y <= A xor B;
end Behavioral;
```

7.XNOR_Gate

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

-- Entity declaration for XNOR gate
entity XNOR_Gate is
   Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          Y : out STD_LOGIC);
end XNOR_Gate;

-- Architecture definition
architecture Behavioral of XNOR_Gate is
begin
   -- XNOR gate logic
   Y <= not (A xor B);
end Behavioral;
```

8.Half_Adder

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

-- Entity declaration for Half Adder
entity Half_Adder is
   Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          Sum : out STD_LOGIC;
          Carry : out STD_LOGIC);
end Half_Adder;

-- Architecture definition
architecture Behavioral of Half_Adder is
begin
   -- Half adder logic
   Sum <= A xor B;  -- Sum is A XOR B
   Carry <= A and B;  -- Carry is A AND B
end Behavioral;
```

9.Full_Adder

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

-- Entity declaration for Full Adder
entity Full_Adder is
   Port ( A     : in  STD_LOGIC;  -- First input bit
          B     : in  STD_LOGIC;  -- Second input bit
          Cin   : in  STD_LOGIC;  -- Carry-in bit
          Sum   : out STD_LOGIC;  -- Sum output
          Cout  : out STD_LOGIC   -- Carry-out output
          );
end Full_Adder;

-- Architecture definition
architecture Behavioral of Full_Adder is
begin
   -- Full adder logic
   Sum  <= (A xor B) xor Cin;  -- Sum is the XOR of inputs A, B, and Cin
   Cout <= (A and B) or (Cin and (A xor B));  -- Carry-out logic
end Behavioral;
```

10.Half Subtractor

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

-- Entity declaration for Half Subtractor
entity Half_Subtractor is
   Port ( A     : in  STD_LOGIC;  -- Minuend (first input)
          B     : in  STD_LOGIC;  -- Subtrahend (second input)
          Difference : out STD_LOGIC;  -- Difference output
          Borrow     : out STD_LOGIC   -- Borrow output
          );
end Half_Subtractor;

-- Architecture definition
architecture Behavioral of Half_Subtractor is
begin
   -- Half subtractor logic
   Difference <= A xor B;  -- Difference is A XOR B
   Borrow <= not A and B;  -- Borrow is NOT A AND B
end Behavioral;
```

11.Full_Subtractor

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

-- Entity declaration for Full Subtractor
entity Full_Subtractor is
   Port ( A        : in  STD_LOGIC;  -- Minuend (first input)
          B        : in  STD_LOGIC;  -- Subtrahend (second input)
          Bin      : in  STD_LOGIC;  -- Borrow input (borrow from the previous bit)
          Difference : out STD_LOGIC;  -- Difference output
          Borrow     : out STD_LOGIC   -- Borrow output
          );
end Full_Subtractor;

-- Architecture definition
architecture Behavioral of Full_Subtractor is
begin
   -- Full subtractor logic
   Difference <= (A xor B) xor Bin;  -- Difference is (A XOR B) XOR Bin
   Borrow <= (not A and B) or (not (A xor B) and Bin);  -- Borrow logic
end Behavioral;
```

12.Carry Propagate Adder

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

-- Entity declaration for Carry Propagate Adder (CPA)
entity Carry_Propagate_Adder is
   Port ( A     : in  STD_LOGIC_VECTOR(3 downto 0);  -- First 4-bit input
          B     : in  STD_LOGIC_VECTOR(3 downto 0);  -- Second 4-bit input
          Cin   : in  STD_LOGIC;                     -- Carry-in bit
          Sum   : out STD_LOGIC_VECTOR(3 downto 0);   -- Sum output
          Cout  : out STD_LOGIC                       -- Carry-out output
          );
end Carry_Propagate_Adder;

-- Architecture definition
architecture Behavioral of Carry_Propagate_Adder is
   signal G, P, C : STD_LOGIC_VECTOR(3 downto 0);  -- Generate, Propagate, and Carry signals
begin
   -- Generate and Propagate Logic
   G(0) <= A(0) and B(0);  -- Generate for bit 0
   G(1) <= A(1) and B(1);  -- Generate for bit 1
   G(2) <= A(2) and B(2);  -- Generate for bit 2
   G(3) <= A(3) and B(3);  -- Generate for bit 3

   P(0) <= A(0) or B(0);  -- Propagate for bit 0
   P(1) <= A(1) or B(1);  -- Propagate for bit 1
   P(2) <= A(2) or B(2);  -- Propagate for bit 2
   P(3) <= A(3) or B(3);  -- Propagate for bit 3

   -- Carry Calculation
```

```
    C(0) <= Cin;  -- Carry-in for bit 0

    C(1) <= G(0) or (P(0) and Cin);  -- Carry for bit 1

    C(2) <= G(1) or (P(1) and C(1));  -- Carry for bit 2

    C(3) <= G(2) or (P(2) and C(2));  -- Carry for bit 3


    -- Sum Calculation

    Sum(0) <= A(0) xor B(0) xor Cin;  -- Sum for bit 0

    Sum(1) <= A(1) xor B(1) xor C(1);  -- Sum for bit 1

    Sum(2) <= A(2) xor B(2) xor C(2);  -- Sum for bit 2

    Sum(3) <= A(3) xor B(3) xor C(3);  -- Sum for bit 3


    -- Carry-out

    Cout <= C(3);  -- Carry-out is the carry of the last bit (bit 3)

end Behavioral;
```

13.Multiplexer(4to1)

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

-- Entity declaration for 4-to-1 Multiplexer
entity MUX_4to1 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); -- 2-bit select signal
          Y     : out STD_LOGIC   -- Output
          );
end MUX_4to1;

-- Architecture definition
architecture Behavioral of MUX_4to1 is
begin
   -- Multiplexer logic: Output Y is selected based on the 2-bit select signal S
   process (A, B, C, D, S)
   begin
      case S is
         when "00" => Y <= A;  -- If S = 00, select input A
         when "01" => Y <= B;  -- If S = 01, select input B
         when "10" => Y <= C;  -- If S = 10, select input C
         when "11" => Y <= D;  -- If S = 11, select input D
         when others => Y <= '0'; -- Default case, just in case
      end case;
   end process;
end Behavioral;
```

14.Multiplexer(2to1)

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

-- Entity declaration for 2-to-1 Multiplexer
entity MUX_2to1 is
   Port ( A     : in  STD_LOGIC;  -- Input A
          B     : in  STD_LOGIC;  -- Input B
          S     : in  STD_LOGIC;  -- Select signal
          Y     : out STD_LOGIC   -- Output
          );
end MUX_2to1;

-- Architecture definition
architecture Behavioral of MUX_2to1 is
begin
   -- Multiplexer logic: Y = A when S = 0, Y = B when S = 1
   Y <= A when (S = '0') else B;
end Behavioral;
```

15.Demultiplexer(1to4)

```
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 signal
        S    : in  STD_LOGIC_VECTOR(1 downto 0); -- 2-bit select signal
        Y0   : out STD_LOGIC;             -- Output 0
        Y1   : out STD_LOGIC;             -- Output 1
        Y2   : out STD_LOGIC;             -- Output 2
        Y3   : out STD_LOGIC              -- Output 3
        );
end DEMUX_1to4;

-- Architecture definition
architecture Behavioral of DEMUX_1to4 is
begin
  -- Demultiplexer logic: Distribute input D to one of the outputs based on the select signal S
  process (D, S)
  begin
    -- Default all outputs to 0
    Y0 <= '0';
    Y1 <= '0';
    Y2 <= '0';
    Y3 <= '0';

    -- Based on the value of S, assign the input D to the appropriate output
    case S is
      when "00" => Y0 <= D;  -- If S = 00, output goes to Y0
```

```
            when "01" => Y1 <= D;  -- If S = 01, output goes to Y1
            when "10" => Y2 <= D;  -- If S = 10, output goes to Y2
            when "11" => Y3 <= D;  -- If S = 11, output goes to Y3
            when others => null;   -- Default case (should never happen)
        end case;
    end process;
end Behavioral;
```

16.Demultiplexer(1to2)

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

-- Entity declaration for 1-to-2 Demultiplexer
entity DEMUX_1to2 is
   Port ( D     : in  STD_LOGIC;                -- Input signal
          S     : in  STD_LOGIC;                -- Select signal
          Y0    : out STD_LOGIC;                -- Output 0
          Y1    : out STD_LOGIC                 -- Output 1
          );
end DEMUX_1to2;

-- Architecture definition
architecture Behavioral of DEMUX_1to2 is
begin
   -- Demultiplexer logic: Distribute input D to one of the outputs based on the select signal S
   process (D, S)
   begin
      -- Default all outputs to 0
      Y0 <= '0';
      Y1 <= '0';

      -- Based on the value of S, assign the input D to the appropriate output
      if S = '0' then
         Y0 <= D;  -- If S = 0, output goes to Y0
      elsif S = '1' then
         Y1 <= D;  -- If S = 1, output goes to Y1
      end if;
   end process;
end Behavioral;
```

17.1-Bit Comparator

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

-- Entity declaration for 1-bit Comparator
entity Comparator_1bit is
   Port ( A   : in  STD_LOGIC;  -- Input A
          B   : in  STD_LOGIC;  -- Input B
          A_gt_B : out STD_LOGIC; -- Output: A > B
          A_lt_B : out STD_LOGIC; -- Output: A < B
          A_eq_B : out STD_LOGIC  -- Output: A = B
          );
end Comparator_1bit;

-- Architecture definition
architecture Behavioral of Comparator_1bit is
begin
   -- Comparator logic
   A_gt_B <= '1' when (A = '1' and B = '0') else '0';  -- A > B
   A_lt_B <= '1' when (A = '0' and B = '1') else '0';  -- A < B
   A_eq_B <= '1' when (A = B) else '0';                -- A = B
end Behavioral;
```

18.8-Bit Register

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


-- Entity declaration for 8-bit Register

entity Register_8bit is

   Port ( CLK  : in  STD_LOGIC;          -- Clock signal

         RESET : in  STD_LOGIC;         -- Reset signal

         LOAD  : in  STD_LOGIC;         -- Load signal (1 to load data, 0 to hold)

         D     : in  STD_LOGIC_VECTOR(7 downto 0); -- 8-bit input data

         Q     : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit output (stored data)

         );

end Register_8bit;


-- Architecture definition

architecture Behavioral of Register_8bit is

   signal reg : STD_LOGIC_VECTOR(7 downto 0);  -- Internal register to hold data

begin

   -- Process to handle loading and resetting the register

   process(CLK, RESET)

   begin

      if RESET = '1' then

         -- Reset the register when RESET is high

         reg <= (others => '0');

      elsif rising_edge(CLK) then

         -- Load data when LOAD is high

         if LOAD = '1' then

            reg <= D;

         end if;

      end if;
```

```
    end process;

    -- Output the stored value from the register
    Q <= reg;
end Behavioral;
```