

T FF

Library IEEE;

USE IEEE.Std\_logic\_1164.all;

entity tff is

port (clk, t: in bit; Q,Qb: inout bit);

end entity;

architecture rtl of tff is begin

tff\_proc : process(clk) begin

if rising\_edge(clk) then

if t = '1' then

Q <= not Q;

end if;

end if;

end process;

Qb <= not Q;

end;

D FF

Library IEEE;

USE IEEE.Std\_logic\_1164.all;

entity tff is

port (clk, t: in bit; Q,Qb: inout bit);

end entity;

architecture rtl of tff is begin

tff\_proc : process(clk) begin

if rising\_edge(clk) then

if t = '1' then

Q <= not Q;

end if;

end if;

end process;

Qb <= not Q;

end;

JK FF

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

entity JK\_FF\_VHDL is

port( J,K: in std\_logic;

Clock: in std\_logic;

Q,Qb: out std\_logic);

end JK\_FF\_VHDL;

architecture Behavioral of JK\_FF\_VHDL is

signal temp: std\_logic;

begin

process (Clock)

begin

if rising\_edge(Clock) then

If (J='0' and K='0') then

temp <= temp;

elsif (J='0' and K='1') then

temp <= '0';

elsif (J='1' and K='0') then

temp <= '1';

elsif (J='1' and K='1') then

temp <= not (temp);

end if;

end if;

end process;

Q <= temp;

Qb <= not temp;

end Behavioral;

SR FF

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

entity SR\_FF\_VHDL is

port( S,R: in std\_logic;

Clock: in std\_logic;

Q,Qb: out std\_logic);

end SR\_FF\_VHDL;

architecture Behavioral of SR\_FF\_VHDL is

signal temp: std\_logic;

begin

process (Clock)

begin

if rising\_edge(Clock) then

If (S='0' and R='0') then

temp <= temp;

elsif (S='0' and R='1') then

temp <= '0';

elsif (S='1' and R='0') then

temp <= '1';

end if;

end if;

end process;

Q <= temp;

Qb <= not temp;

end Behavioral;

2-Bit Counter

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

entity counter\_2bit is

port(

clk : in STD\_LOGIC;

reset : in STD\_LOGIC;

dout : out STD\_LOGIC\_VECTOR(1 downto 0)

);

end counter\_2bit;

architecture counter\_2bit\_arc of counter\_2bit is

begin

counting : process (clk,reset) is

variable m : std\_logic\_vector (1 downto 0) := "00";

begin

if (reset='1') then

m := "00";

elsif (rising\_edge (clk)) then

m := m + 1;

end if;

dout <= m;

end process counting;

end counter\_2bit\_arc;

A screen shot of a computer

Description automatically generated



library ieee;

use ieee.std\_logic\_1164.all;

entity half\_adder is

port (a, b: in std\_logic;

sum, carry: out std\_logic);

end half\_adder;

architecture dataflow of half\_adder is

begin

sum <= a xor b;

carry <= a and b;

end dataflow;

--------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

entity half\_adder is

port (a, b: in std\_logic;

sum, carry: out std\_logic);

end half\_adder;

architecture behavior of half\_adder is

begin

ha: process (a, b)

begin

if a = ‘1’ then

sum <= not b;

carry <= b;

else

sum <= b;

carry <= ‘0’;

end if;

end process ha;

end behavior;

------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

entity half\_adder is -- Entity declaration for half adder

port (a, b: in std\_logic;

sum, carry: out std\_logic);

end half\_adder;

architecture structure of half\_adder is -- Architecture body for half adder

component xor\_gate is -- xor component declaration

port (i1, i2: in std\_logic;

o1: out std\_logic);

end component;

component and\_gate is -- and component declaration

port (i1, i2: in std\_logic;

o1: out std\_logic);

end component;

begin

u1: xor\_gate port map (i1 => a, i2 => b, o1 => sum);

u2: and\_gate port map (i1 => a, i2 => b, o1 => carry);

end structure;

-- We can also use Positional Association

-- => u1: xor\_gate port map (a, b, sum);

-- => u2: and\_gate port map (a, b, carry\_out);



Library IEEE;

Use IEEE.STD\_LOGIC\_1164.ALL;

Entity FullAdder is

Port (

A : in STD\_LOGIC;

B : in STD\_LOGIC;

Cin : in STD\_LOGIC;

Sum : out STD\_LOGIC;

Cout : out STD\_LOGIC

);

End FullAdder;

Architecture Behavioral of FullAdder is

Begin

Process (A, B, Cin)

Begin

* Full Adder logic

Sum <= A xor B xor Cin;

Cout <= (A and B) or (B and Cin) or (Cin and A);

End process;

End Behavioral;

Library IEEE;

Use IEEE.STD\_LOGIC\_1164.ALL;

Entity HalfSubtractor is

Port (

A : in STD\_LOGIC;

B : in STD\_LOGIC;

Difference : out STD\_LOGIC;

Borrow : out STD\_LOGIC

);

End HalfSubtractor;

Architecture Behavioral of HalfSubtractor is

Begin

Process (A, B)

Begin

* Half Subtractor logic

Difference <= A xor B;

Borrow <= not A and B;

End process;

End Behavioral;

Library IEEE;

Use IEEE.STD\_LOGIC\_1164.ALL;

Entity FullSubtractor is

Port (

A : in STD\_LOGIC;

B : in STD\_LOGIC;

Bin : in STD\_LOGIC;

Difference : out STD\_LOGIC;

Bout : out STD\_LOGIC

);

End FullSubtractor;

Architecture Behavioral of FullSubtractor is

Begin

Process (A, B, Bin)

Begin

* Full Subtractor logic

Difference <= A xor B xor Bin;

Bout <= (not A and B) or (B and Bin) or (not A and Bin);

End process;

End Behavioral;

Library IEEE;

Use IEEE.STD\_LOGIC\_1164.ALL;

Entity Encoder4to2 is

Port (

I0 : in STD\_LOGIC;

I1 : in STD\_LOGIC;

I2 : in STD\_LOGIC;

I3 : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR (1 downto 0)

);

End Encoder4to2;

Architecture Behavioral of Encoder4to2 is

Begin

Process (I0, I1, I2, I3)

Begin

If (I0 = ‘1’) then

Y <= “00”;

Elsif (I1 = ‘1’) then

Y <= “01”;

Elsif (I2 = ‘1’) then

Y <= “10”;

Elsif (I3 = ‘1’) then

Y <= “11”;

Else

Y <= “00”; -- Default output, can be set to others based on design requirements

End if;

End process;

End Behavioral;

Library IEEE;

Use IEEE.STD\_LOGIC\_1164.ALL;

Entity Decoder2to4 is

Port (

A : in STD\_LOGIC\_VECTOR (1 downto 0);

Y0 : out STD\_LOGIC;

Y1 : out STD\_LOGIC;

Y2 : out STD\_LOGIC;

Y3 : out STD\_LOGIC

);

End Decoder2to4;

Architecture Behavioral of Decoder2to4 is

Begin

Process (A)

Begin

* Initialize all outputs to ‘0’

Y0 <= ‘0’;

Y1 <= ‘0’;

Y2 <= ‘0’;

Y3 <= ‘0’;

* Decode the input

Case A is

When “00” =>

Y0 <= ‘1’;

When “01” =>

Y1 <= ‘1’;

When “10” =>

Y2 <= ‘1’;

When “11” =>

Y3 <= ‘1’;

When others =>

* Handles any unexpected input (though not necessary for 2-bit input)

Y0 <= ‘0’;

Y1 <= ‘0’;

Y2 <= ‘0’;

Y3 <= ‘0’;

End case;

End process;

End Behavioral;