VHDL Code for an SR Latch

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity srl is
   port(r,s:in bit; q,qbar:buffer bit);
end srl;

architecture SM of srl is
   signal s1,r1:bit;
begin
   q<= s nand qbar;
   qbar<= r nand q;
end SM;
```

VHDL Code for a D Latch

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity Dl is
   port(d:in bit; q,qbar:buffer bit);
end Dl;

architecture SM of Dl is
   signal s1,r1:bit;
begin
   q<= d nand qbar;
   qbar<= d nand q;
end SM;
```

VHDL Code for an SR Flip Flop

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity srflip is
   port(r,s,clk:in bit; q,qbar:buffer bit);
end srflip;

architecture SM of srflip is
   signal s1,r1:bit;
begin
   s1<=s nand clk;
   r1<=r nand clk;
   q<= s1 nand qbar;
   qbar<= r1 nand q;
end SM;
```

VHDL Code for an JK Flip Flop

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

entity jk is
   port(
      j : in STD_LOGIC;
      k : in STD_LOGIC;
      clk : in STD_LOGIC;
      reset : in STD_LOGIC;
      q : out STD_LOGIC;
      qb : out STD_LOGIC
   );
end jk;

architecture SM of jk is
begin
   jkff : process (j,k,clk,reset) is
   variable m : std_logic := '0';

   begin
      if (reset = '1') then
         m : = '0';
      elsif (rising_edge (clk)) then
         if (j/ = k) then
            m : = j;
         elsif (j = '1' and k = '1') then
            m : = not m;
         end if;
      end if;

      q <= m;
      qb <= not m;
   end process jkff;
end SM;
```

VHDL Code for a D Flip Flop

```
Library ieee;
use ieee.std_logic_1164.all;

entity dflip is
   port(d,clk:in bit; q,qbar:buffer bit);
end dflip;

architecture SM of dflip is
   signal d1,d2:bit;
begin
   d1<=d nand clk;
   d2<=(not d) nand clk;
   q<= d1 nand qbar;
   qbar<= d2 nand q;
end SM;
```

VHDL Code for a T Flip Flop

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

entity Toggle_flip_flop is
   port(
      t : in STD_LOGIC;
      clk : in STD_LOGIC;
      reset : in STD_LOGIC;
      dout : out STD_LOGIC
   );
end Toggle_flip_flop;

architecture SM of Toggle_flip_flop is
begin
   tff : process (t,clk,reset) is
   variable m : std_logic : = '0';

   begin
      if (reset = '1') then
         m : = '0';
      elsif (rising_edge (clk)) then
         if (t = '1') then
            m : = not m;
         end if;
      end if;
      dout < = m;
   end process tff;
end SM;
```

4 bit UP Counter Design:

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

entity counter is
   port(Clock, CLR : in std_logic;
      Q : out std_logic_vector(3 downto 0)
   );
end counter;

architecture SM of counter is
   signal tmp: std_logic_vector(3 downto 0);
begin
   process (Clock, CLR)

   begin
      if (CLR = '1') then
         tmp < = "0000";
      elsif (Clock'event and Clock = '1') then
         mp <= tmp + 1;
      end if;
   end process;
   Q <= tmp;
end SM;
```

4 bit down counter design:

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

entity dcounter is
   port(Clock, CLR : in std_logic;
      Q : out std_logic_vector(3 downto 0));
end dcounter;

architecture SM of dcounter is
   signal tmp: std_logic_vector(3 downto 0);

begin
   process (Clock, CLR) vi
   begin
      if (CLR = '1') then
         tmp <= "1111";
      elsif (Clock'event and Clock = '1') then
         tmp <= tmp - 1;
      end if;
   end process;
   Q <= tmp;
end SM;
```

SIPO Serial In parallel out shift register Example:

```
library ieee;
use ieee.std_logic_1164.all;
entity sipo is
 port(
 clk, clear : in std_logic;
 Input_Data: in std_logic;
 Q: inout std_logic_vector(3 downto 0) );
end sipo;
architecture arch of sipo is
begin
 process (clk)
 begin
 if clear = '1' then
 Q <= "0000";                                  INDEX   3  2  1  0
 elsif (CLK'event and CLK='1') then  -- INPUT -> Q0,Q1,Q2,Q3
 Q(2 downto 1) <= Q(3 downto 1);
 Q(3) <= Input_Data;
 end if;
 end process;
end arch;
```

```
Parallel in Parallel shift Register Example (PIPO)
```

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

entity pipo is
 port(
 clk : in std_logic;
 D: in std_logic_vector(3 downto 0);
 Q: out std_logic_vector(3 downto 0)
 );
end pipo;
architecture arch of pipo is
begin
 process (clk,d)
 begin
 if (CLK'event and CLK='1') then
 Q <= D;
 end if;
 end process;
 end arch;
```

Sequence Detector for the pattern “101” design using mealy FSM:

--Three states are s0,s1 and s2

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

entity mealy is
      Port ( clk : in STD_LOGIC;
      din : in STD_LOGIC;
      rst : in STD_LOGIC;
      dout : out STD_LOGIC);
end mealy;

architecture Behavioral of mealy is
      type state is (st0, st1, st2); --State is user defined data
                                            --type

      signal present_state, next_state : state;
begin
      process1 : process (clk) --  process1 is label name
      begin
        if rising_edge(clk) then
          if (rst = '1') then
             present_state <= st0;
          else
            present_state <= next_state;
         end if;
       end if;
      end process;  --process end

      process2 : process(present_state, din) – process2 is label name
      begin
          dout <= '0';
      case (present_state) is when st0 =>
          if (din = '1') then
            next_state <= st1;
            dout <= '0';
          else
            next_state <= st0;
            dout <= '0';
          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 <= st1;
               dout <= '1';
            else
                next_state <= st0;
                dout <= '0';
            end if;
      when others =>
           next_state <= st0;
           dout <= '0';
    end case;
end process;
end Behavioral;

--case expression  syntax is

         --when choice => sequential_statements

         --when choice => sequential_statements

         . . .

         --end case;
```