```
entity demux_1to8 is
      port (A   : in  bit;
            Sel : in  bit_vector (2 downto 0);
            F   : out bit_vector (7 downto 0));
end entity;

architecture demux_1to8_arch of demux_1to8 is
begin
      F(7) <= A and not Sel(2) and not Sel(1) and not Sel(0);
      F(6) <= A and not Sel(2) and not Sel(1) and Sel(0);
      F(5) <= A and not Sel(2) and Sel(1) and not Sel(0);
      F(4) <= A and not Sel(2) and Sel(1) and Sel(0);
      F(3) <= A and Sel(2) and not Sel(1) and not Sel(0);
      F(2) <= A and Sel(2) and not Sel(1) and Sel(0);
      F(1) <= A and Sel(2) and Sel(1) and not Sel(0);
      F(0) <= A and Sel(2) and Sel(1) and Sel(0);
end architecture;

entity demux_1to8_TB is
end entity;

architecture demux_1to8_TB_arch of demux_1to8_TB is
component demux_1to8
      port (A   : in  bit;
            Sel : in  bit_vector (2 downto 0);
            F   : out bit_vector (7 downto 0));
end component;
signal A_TB   : bit;
signal Sel_TB : in  bit_vector (2 downto 0);
signal F_TB      : out bit_vector (7 downto 0));
begin
DUT1: SystemE port map (A => A_TB, Sel => Sel_TB, F => F_TB);
STIMULUS : process
begin
A_TB <= '0'; Sel_TB <= '001'; wait for 100 ps;
A_TB <= '1'; Sel_TB <= '001'; wait for 100 ps;
end process;
end architecture;
```

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

entity Seq_Det_behavioral is
      port (Clock, Reset : in std_logic;
            Din          : in std_logic;
            Dout         : out std_logic);
end entity;

architecture Seq_Det_behavioral_arch of Seq_Det_behavioral is
      sybtype State_Type is std_logic_vector (3 downto 0);
      constant S0 : State_Type := ‘0001’;
      constant S1 : State_Type := ‘0010’;
      constant S2 : State_Type := ‘0100’;
      constant S3 : State_Type := ‘1000’;
      signal current_state, next_state : State_Type;
begin

      STATE_MEMORY : process (Clock, Reset)
            begin
                  if (Reset=’0’) then
                        current_state <= Start;
                  elsif (Clock’event and Clock=’1’) then
                        current_state <= next_state;
                  end if;
      end process;

      NEXT_STATE_LOGIC : process (current_state, Din)
            begin
```

```
                case (current_state) is
                      when S0 => if (Din = '0') then
                                      next_state <= S0;
                                 else
                                      next_state <= S1;
                                 end if;
                      when S1 =>  if (Din = '0') then
                                      next_state <= S2;
                                 else
                                      next_state <= S3;
                                 end if;
                      when S2 =>  if (Din = '0') then
                                      next_state <= S0;
                                 else
                                      next_state <= S3;
                                 end if;
                      when S3 =>  if (Din = '0') then
                                      next_state <= S3;
                                 else
                                      next_state <= S0;
                                 end if;
                end case;
      end process;

      OUTPUT_LOGIC : process (current_state, Din)
            begin
                case (current_state) is
                      when S1   => if (Din = '1') then
                                        Dout <= '0';
                                   else
                                        Dout <= '1';
                                   end if;
                      when others   => if (Din = '1') then
                                        Dout <= '1';
                                   else
                                        Dout <= '0';
                                   end if;
                end case;
      end process;
end architecture;
```