VHDL ακολουθιακή λογική

Λογική με στοιχεία μνήμης

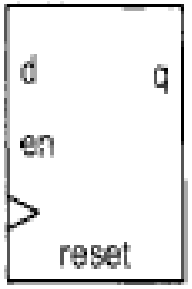
DFF

clk	q*
0	q
1	q
↓	d

(a) D FF

reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↓	d

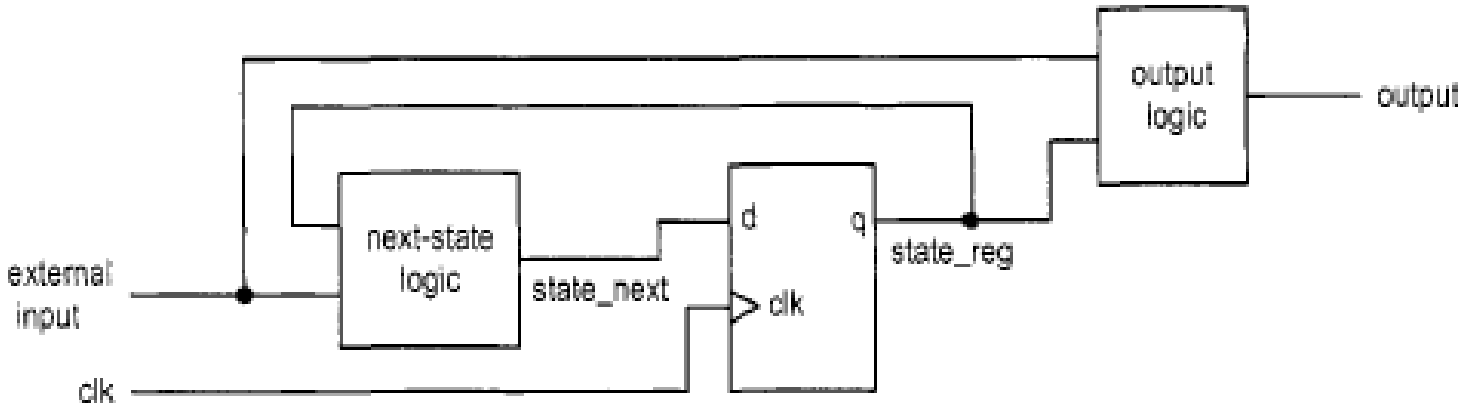
(b) D FF with asynchronous reset



reset	clk	en	q*
1	-	-	0
0	0	-	q
0	1	-	q
0	↓	0	q
0	↓	1	d

Flip-Flop με Enable

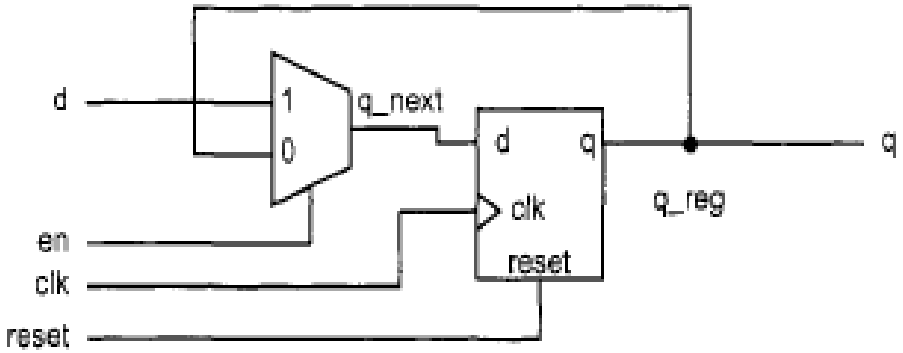
Λογική ενός ασύγχρονου συστήματος



DFF με enable

```
library ieee;
use ieee.std_logic_1164.all;
entity d_ff_en is
    port(
        clk, reset: in std_logic;
        en: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end d_ff_en;

architecture arch of d_ff_en is
begin
    process(clk,reset)
    begin
        if (reset='1') then
            q <='0';
        elsif (clk'event and clk='1') then
            if (en='1') then
                q <= d;
            end if;
        end if;
    end process;
end arch;
```

DFF με enable (mux – dff)




```
begin
    -- D FF
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <='0';
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;
    -- next-state logic
    r_next <= d when en ='1' else
                r_reg;
    -- output logic
    q <= r_reg;
end two_seg_arch;
```

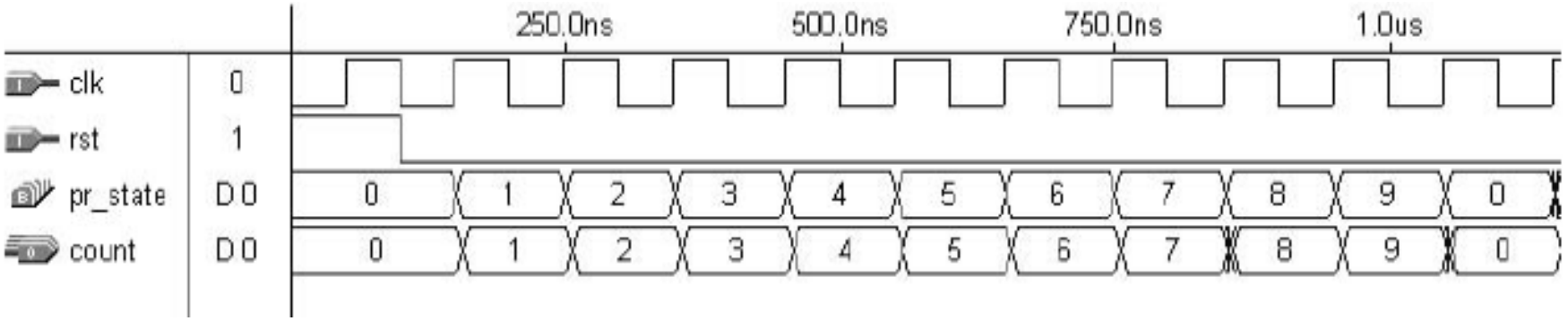
Register

```
library ieee;
use ieee.std_logic_1164.all;
entity reg_reset is
   port(
      clk, reset: in std_logic;
      d: in std_logic_vector(7 downto 0);
      q: out std_logic_vector(7 downto 0)
   );
end reg_reset;

architecture arch of reg_reset is
begin
   process(clk,reset)
   begin
      if (reset='1') then
         q <=(others=>'0');
      elsif (clk'event and clk='1') then
         q <= d;
      end if;
   end process;
end arch;
```

Shift Register

```
library ieee;
use ieee.std_logic_1164.all;
entity free_run_shift_reg is
   generic(N: integer := 8);
   port(
      clk, reset: in std_logic;
      s_in: in std_logic;
      s_out: out std_logic
   );
end free_run_shift_reg;

architecture arch of free_run_shift_reg is
   signal r_reg: std_logic_vector(N-1 downto 0);
   signal r_next: std_logic_vector(N-1 downto 0);
begin
   -- register
   process(clk,reset)
   begin
      if (reset='1') then
         r_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         r_reg <= r_next;
      end if;
   end process;
   -- next-state logic (shift right 1 bit)
   r_next <= s_in & r_reg(N-1 downto 1);
   -- output
   s_out <= r_reg(0);
end arch;
```

Μετρητής BCD

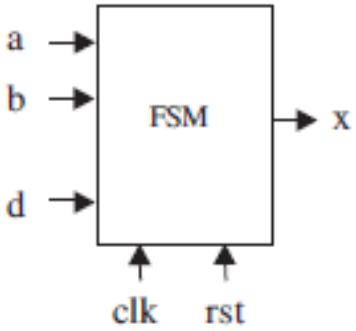


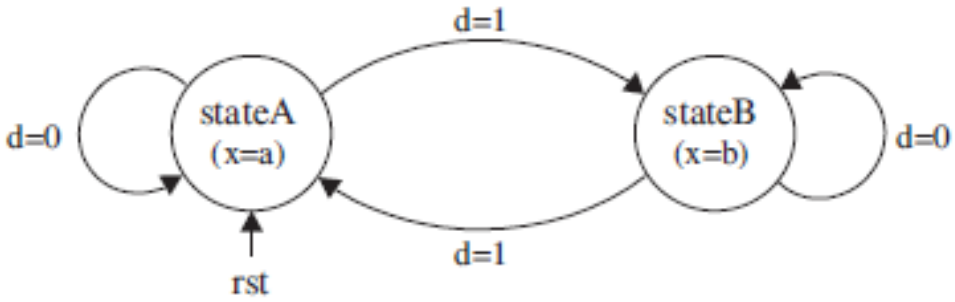

```
1  ------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------------
5  ENTITY counter IS
6     PORT ( clk, rst: IN STD_LOGIC;
7            count: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
8  END counter;
9  ------------------------------------------------
```

Μετρητής BCD (architecture)

```
10 ARCHITECTURE state_machine OF counter IS
11    TYPE state IS (zero, one, two, three, four,
12       five, six, seven, eight, nine);
13    SIGNAL pr_state, nx_state: state;
14 BEGIN
15    ------------ Lower section: -----------------
16    PROCESS (rst, clk)
17    BEGIN
18       IF (rst='1') THEN
19          pr_state <= zero;
20       ELSIF (clk'EVENT AND clk='1') THEN
21          pr_state <= nx_state;
22       END IF;
23    END PROCESS;
24    ------------ Upper section: -----------------
25    PROCESS (pr_state)
26    BEGIN
27       CASE pr_state IS
28          WHEN zero =>
29             count <= "0000";
30             nx_state <= one;
31          WHEN one =>
32             count <= "0001";
33             nx_state <= two;
34          WHEN two =>
35             count <= "0010";
36             nx_state <= three;
37          WHEN three =>
38             count <= "0011";
39             nx_state <= four;
40          WHEN four =>
41             count <= "0100";
42             nx_state <= five;
43          WHEN five =>
44             count <= "0101";
45             nx_state <= six;
46          WHEN six =>
47             count <= "0110";
48             nx_state <= seven;
49          WHEN seven =>
50             count <= "0111";
51             nx_state <= eight;
52          WHEN eight =>
53             count <= "1000";
54             nx_state <= nine;
55          WHEN nine =>
56             count <= "1001";
57             nx_state <= zero;
58       END CASE;
59    END PROCESS;
60 END state_machine;
61 ---------------------------------------------
```

Απλή FSM

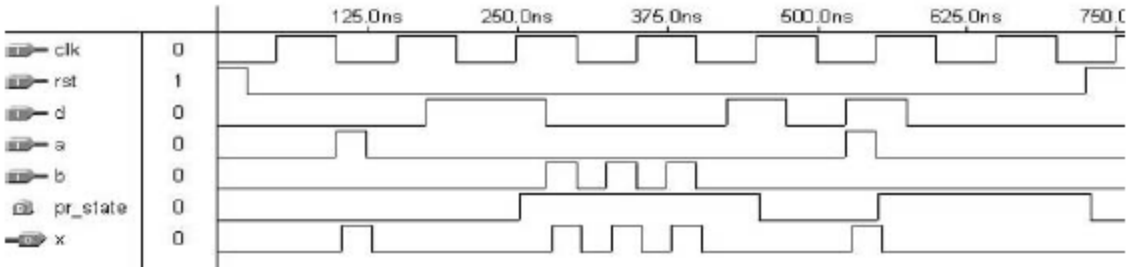


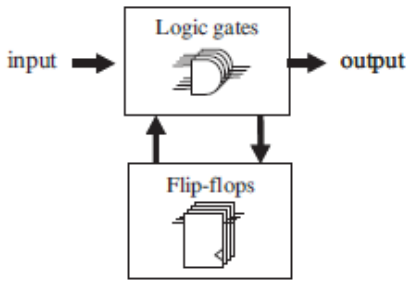



```
1  ------------------------------------------
2  ENTITY simple_fsm IS
3     PORT ( a, b, d, clk, rst: IN BIT;
4            x: OUT BIT);
5  END simple_fsm;
6  ------------------------------------------
```

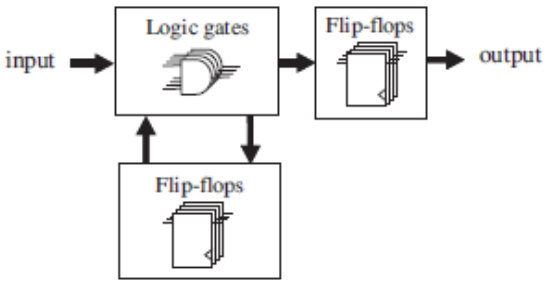
Απλή FSM (architecture)

```
7  ARCHITECTURE simple_fsm OF simple_fsm IS
8     TYPE state IS (stateA, stateB);
9     SIGNAL pr_state, nx_state: state;
10 BEGIN
11    ----- Lower section: ----------------
12    PROCESS (rst, clk)
13    BEGIN
14       IF (rst='1') THEN
15          pr_state <= stateA;
16       ELSIF (clk'EVENT AND clk='1') THEN
17          pr_state <= nx_state;
18       END IF;
19    END PROCESS;
20    ---------- Upper section: ---------------
21    PROCESS (a, b, d, pr_state)
22    BEGIN
23       CASE pr_state IS
24          WHEN stateA =>
25             x <= a;
26             IF (d='1') THEN nx_state <= stateB;
27             ELSE nx_state <= stateA;
28             END IF;
29          WHEN stateB =>
30             x <= b;
31             IF (d='1') THEN nx_state <= stateA;
32             ELSE nx_state <= stateB;
33             END IF;
34       END CASE;
35    END PROCESS;
36 END simple_fsm;
37 -----------------------------------------
```



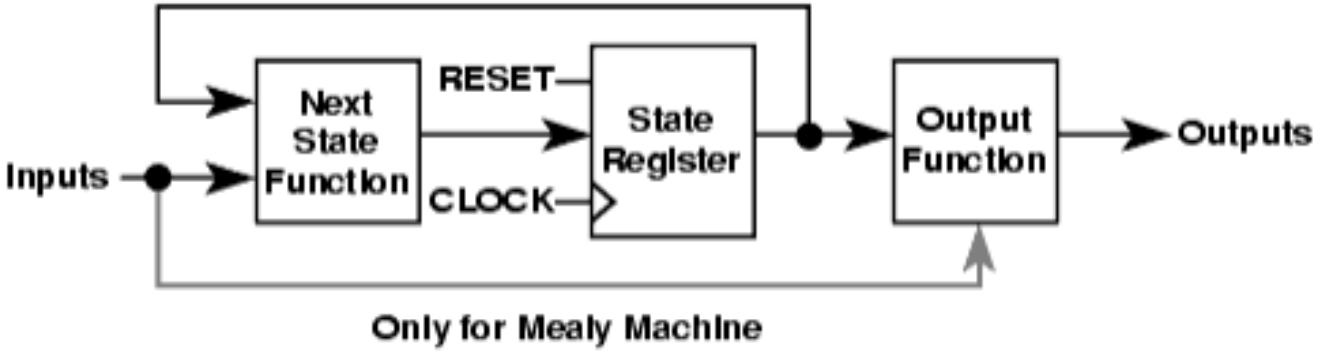


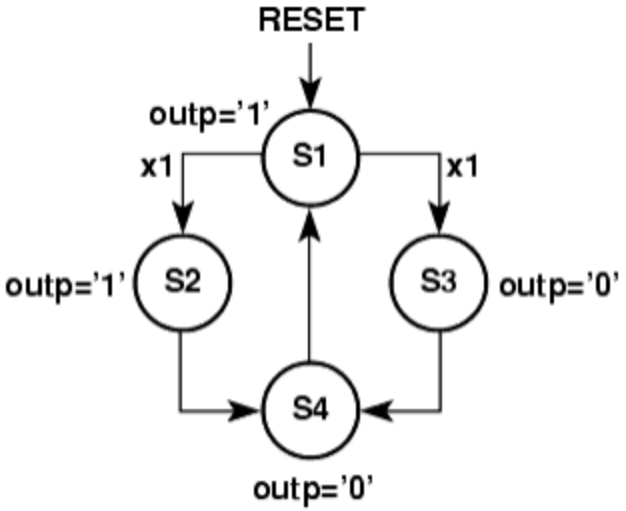
(a)



(b)

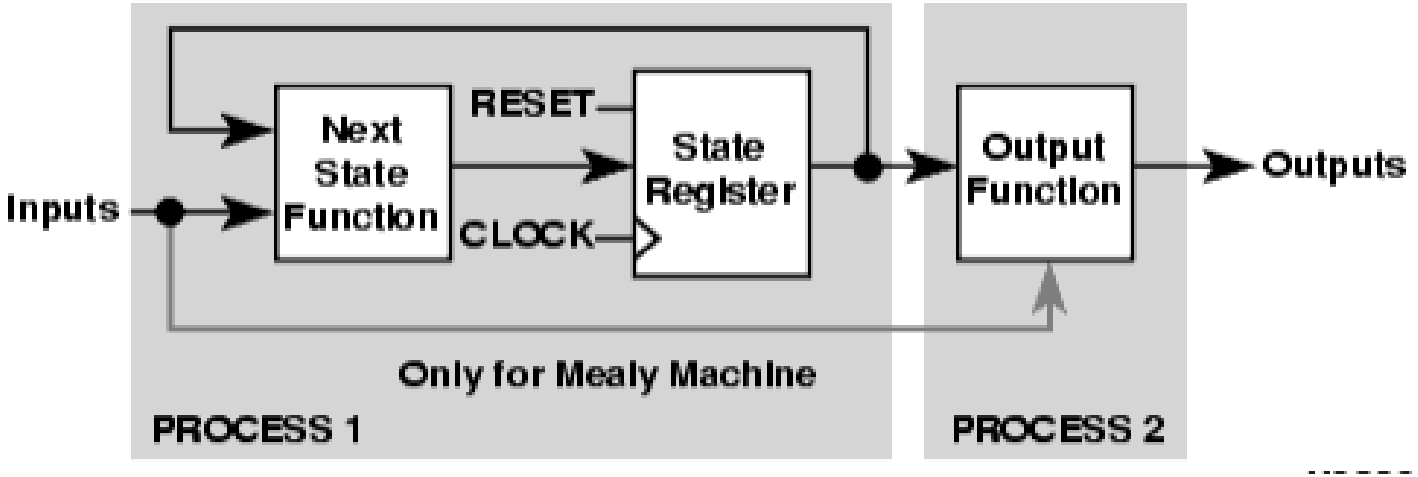
FSM Παραδείγματα





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

entity fsm is
  port ( clk, reset, x1 : IN std_logic;
                  outp : OUT std_logic);
end entity;
architecture beh1 of fsm is
  type state_type is (s1,s2,s3,s4);
  signal state: state_type ;
begin
  process (clk,reset)
  begin
    if (reset ='1') then
      state <=s1; outp<='1';
    elsif (clk='1' and clk'event) then
      case state is
        when s1 => if x1='1' then state <= s2;
                   else          state <= s3;
                   end if;
                   outp <= '1';
        when s2 => state <= s4; outp <= '1';
        when s3 => state <= s4; outp <= '0';
        when s4 => state <= s1; outp <= '0';
      end case;
    end if;
  end process;
end beh1;
```

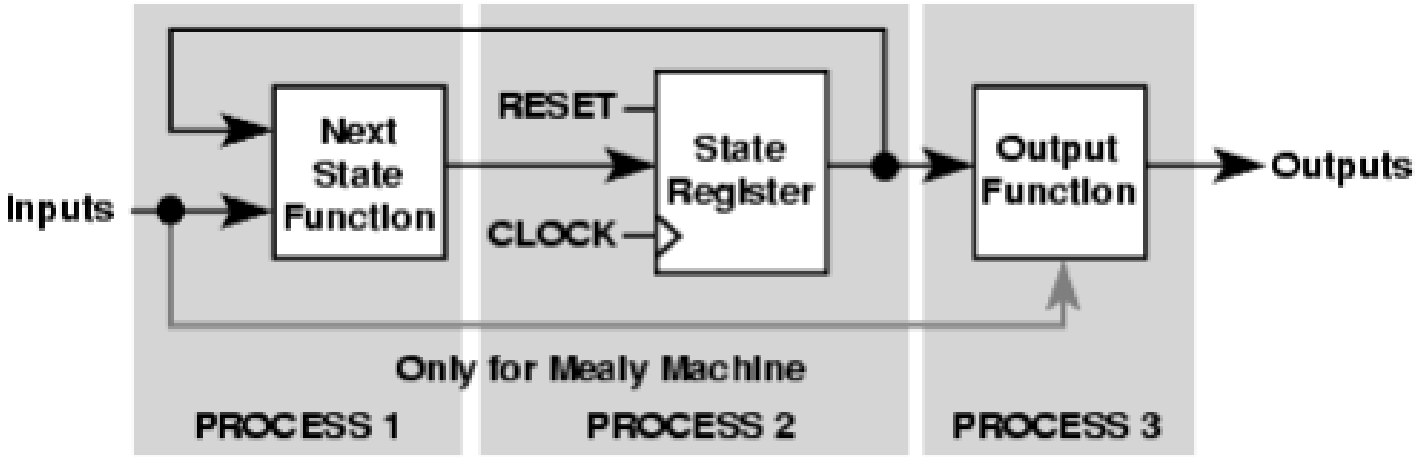
FSM with 2 processes



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

entity fsm is
  port ( clk, reset, x1 : IN std_logic;
                  outp : OUT std_logic);
end entity;
architecture beh1 of fsm is
  type state_type is (s1,s2,s3,s4);
  signal state: state_type ;
begin
  process1: process (clk,reset)
  begin
    if (reset ='1') then state <=s1;
    elsif (clk='1' and clk'Event) then
      case state is
        when s1 => if x1='1' then state <= s2;
                   else           state <= s3;
                   end if;
        when s2 => state <= s4;
        when s3 => state <= s4;
        when s4 => state <= s1;
      end case;
    end if;
  end process process1;
```

```
process2 : process (state)
  begin
    case state is
      when s1 => outp <= '1';
      when s2 => outp <= '1';
      when s3 => outp <= '0';
      when s4 => outp <= '0';
    end case;
end process process2;
end beh1;
```

FSM with 3 processes



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

entity fsm is
  port ( clk, reset, x1 : IN std_logic;
                    outp : OUT std_logic);
end entity;
architecture beh1 of fsm is
  type state_type is (s1,s2,s3,s4);
  signal state, next_state: state_type ;
begin
  process1: process (clk,reset)
  begin
    if (reset ='1') then
      state <=s1;
    elsif (clk='1' and clk'Event) then
      state <= next_state;
    end if;
  end process process1;
```

```
 process2 : process (state, x1)
  begin
    case state is
          when s1 => if x1='1' then
                       next_state <= s2;
                     else
                       next_state <= s3;
                     end if;
          when s2 => next_state <= s4;
          when s3 => next_state <= s4;
          when s4 => next_state <= s1;
    end case;
end process process2;

process3 : process (state)
 begin
    case state is
       when s1 => outp <= '1';
       when s2 => outp <= '1';
       when s3 => outp <= '0';
       when s4 => outp <= '0';
    end case;
end process process3;
end beh1;
```

```
library ieee;
    use ieee.std_logic_1164.all;
entity lfsr  is
  port (
    cout   :out std_logic_vector (7 downto 0);-- Output of the counter
    enable :in  std_logic;                    -- Enable counting
    clk    :in  std_logic;                    -- Input rlock
    reset  :in  std_logic      );
end entity;
architecture rtl of lfsr is
    signal count           :std_logic_vector (7 downto 0);
    signal linear_feedback :std_logic;
begin
    linear_feedback <= not(count(7) xor count(3));

    process (clk, reset) begin
        if (reset = '1') then
            count <= (others=>'0');
        elsif (rising_edge(clk)) then
            if (enable = '1') then
             count <= (count(6) & count(5) & count(4) & count(3)  & count(2) & count(1) & count(0) & linear_feedback);
            end if;
        end if;
    end process;
    cout <= count;
end architecture;
```