23 April 2007

Create a clock-synchronous VHDL entity with a 4-bits control input (**CTL4**) and a parallel 12-bits output (**OUT12**).

Starting from a reset, the entity must sample the input **CTL4** and receive data from a serial input (**DATAIN**). If the first 4 bits from the serial input stream are equal to the control word sampled from **CTL4**, the entity must transfer these 4 bits to the output followed by the subsequent 8 bits from **DATAIN**. The entity must set the output to high impedance in any other situation and when the enable signal is set to 0.

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


entity Key_Rec is
   port(CLK, DATAIN, ENABLE, RESET: in std_logic;
        CTL4: in std_logic_vector (3 downto 0);
        OUT12: out std_logic_vector(11 downto 0) );
end Key_Rec;


architecture Behavioral of Key_Rec is

   signal key: std_logic_vector (3 downto 0);  --For sampling the key word in CTL4.
   signal reg: std_logic_vector (11 downto 0); --For uploading the input DATAIN.
   signal cnt: integer range 0 to 15:=0;

   begin

      process (CLK, RESET)

      begin

         if (RESET='1') then
            --If a reset pulse occurs, a general reset is executed:
            reg <= (others =>'0');
            out12 <= (others =>'0');
            key <= (others =>'0');
            cnt <= 0;
         elsif (rising_edge (CLK)) then
            if(cnt < 12) then
               reg <= reg(10 downto 0) & DATAIN;

               --At the beginning of the upload (cnt=0), control input is sampled;
               --for the remaining values from DATAIN, just the increment of cnt is executed
               case cnt is
                  when 0 => key <= CTL4;
                            cnt <= cnt + 1;

                  when 1 to 12 => cnt <= cnt + 1;

                  --No action is required in this case.
                  when others => null;
               end case;
            end if;

            --After the upload is complete, the output is computed:
            --if the output is enabled and the first four bits
            --of DATAIN are equal to the control input, the serial input is sent to parallel
            --output; otherwise, the output is set to high impedance.
            if ((cnt >= 12) and (ENABLE = '1') and (reg (11 downto 8) = key)) then
               out12 <= reg;
            else
               out12 <= (others => 'Z');
            end if;

         end if;

      end process;

end Behavioral;

```

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   --If the first 4 bits which come from DIN correspond to the key
7   --which comes from CTL4, these 4 bits and the following 8 DIN bits
8   --(12 bits) are tranferred to the serial output OUT12.
9   --OUT12 is kept in high impedance state if: 1) the key doesn't correspond
10  --to the first 4 DIN bits; 2) EN=0.
11  --Asynchronous reset can be performed.
12
13  entity KeyRec is
14     port (CLK, RST, EN, DIN: in std_logic;
15           CTL4: in std_logic_vector (3 downto 0);
16           OUT12: out std_logic);
17  end KeyRec;
18
19  architecture Behavioral of KeyRec is
20     signal regK, regIN: std_logic_vector (3 downto 0):="ZZZZ";
21     signal buff: std_logic:='Z';
22     signal cnt: integer range 0 to 20:=0;
23  begin
24
25  process (CLK, EN, RST)
26  begin
27     --1) RST
28     if (RST = '1') then
29        cnt <= 0;
30        regK <= "ZZZZ";
31        regIN <= "ZZZZ";  --NB: regIN is used as a FIFO stack
32        buff <= 'Z';
33     --RST = '0'
34     elsif (rising_edge(CLK)) then
35        --2) key is stored into regK; at the same time, the first serial
36        --input bit is stored into regIN:
37        if (cnt = 0) then
38           regK <= CTL4;
39           regIN <= regIN(2 downto 0) & DIN;
40           buff <= 'Z';
41           cnt <= cnt + 1;
42        --3) 3 more serial bits are stored into regIN, which is
43        --then completely uploaded:
44        elsif (cnt > 0 and cnt < 4) then
45           regIN <= regIN(2 downto 0) & DIN;
46           buff <= 'Z';
47           cnt <= cnt + 1;
48        --4) if regIN=regK, the first bit is tranferred to buff and a new bit
49        --is stored in regIN; otherwise buff is set to high impedance and cnt is
50        --set to 0 so that a new key can be stored and the entity can restart its
51        --operations:
52        elsif (cnt = 4) then
53           if (regIN = regK) then
54              buff <= regIN(3);
55              regIN <= regIN(2 downto 0) & DIN;
56              cnt <= cnt + 1;
57           else
58              buff <= 'Z';
59              cnt <= 0;
60           end if;
61        --5) the entity can execute the following statements just if key has been
62        --recognized;
63        --regIN1 content is transferred to buff and at the same time it is
64        --replaced by 4 more DIN bits:
65        elsif (cnt > 4 and cnt < 12) then
66           buff <= regIN(3);
67           regIN <= regIN(2 downto 0) & DIN;
68           cnt <= cnt + 1;
69        --6) regIN1 content is transferred to buff, but nomore DIN
70        --bits are stored:
71        elsif (cnt > 11 and cnt < 15) then
72           buff <= regIN(3);
73           regIN <= regIN(2 downto 0) & 'Z';
74           cnt <= cnt + 1;
75
76
```

```
77          --7) as the last bit is transferred to buff, cnt is set to 0
78          --so that the entity can restart its operations:
79          elsif (cnt = 15) then
80             buff <= regIN(3);
81             cnt <= 0;
82          end if;
83       end if;
84    end process;
85
86    OUT12 <= buff when (EN = '1') else 'Z';
87
88    end Behavioral;
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY key_recognizer is
PORT  (
                CLK, ENABLE, RESET: IN std_logic;                       -- INGRESSI DI CONTROLLO
                DATAIN: IN std_logic;                                   -- DATI IN INGRESSO
                CTL4  : IN std_logic_vector(3 downto 0);                -- PAROLA CHIAVE
                OUT12 : OUT std_logic_vector(11 downto 0)               -- DATI IN USCITA
                );
END key_recognizer;

architecture Behavioral of key_recognizer is
signal CONT: integer range 0 to 15:=15;                 -- segnale di conteggio
signal data_reg:std_logic_vector(11 downto 0);          -- registro per appogiare i dati
signal key_reg:std_logic_vector(3 downto 0);            -- registro per la parola chiave

begin
process(CLK,RESET) -- il processo è attivato dal clock e dal reset asincrono
begin
        if(RESET='1') then
                key_reg <= (others=>'0');           -- azzeriamo il contenuto dei registri
                data_reg<=(others=>'0');
                CONT <=0;                           -- azzeriamo il contatore
        elsif(CLK'event and CLK='1') then
                data_reg<=data_reg(10 downto 0) & DATAIN; -- i dati entrano nel registro
                        -- il conteggio si incrementa fina a 13 e poi si ferma
                        -- inoltre quanto è 0 campioniamo pure l'ingresso CTL4
                        case CONT is
                          when 0          =>    CONT <= CONT + 1;
                                                key_reg <= CTL4;
                          when 1 to 12    =>    CONT <= CONT + 1;
                          when others     =>    null;
                        end case;
        end if;
end process;
        -- a questo punto non resta che copiare il data_reg all'uscita OUT12
        -- nelle condizioni di conteggio a 12, di abilitazione attiva
        -- e di soddisfacimento della parola chiave
        OUT12 <= data_reg
        when ((data_reg(11 downto 8)=key_reg) and ENABLE='1' and CONT=12)
        else        "ZZZZZZZZZZZZ";
end Behavioral;
```