```
--------------------------------------------------------------------------------
-- UPC - EETAC - CSD
--------------------------------------------------------------------------------
-- P7. An example of a synchronous BCD counter
--
-- FSM design style. There are better or shorter ways to write
-- this counter, but this examples can be used as a seed file
-- to adapt many others FSM.
--
-- http://digsys.upc.edu
--------------------------------------------------------------------------------

LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;


ENTITY counter_BCD_1digit IS
      Port (    CLK     : IN    STD_LOGIC;
        CD      : IN    STD_LOGIC;
        CE      : IN    STD_LOGIC;
        Q       : OUT   STD_LOGIC_VECTOR(3 DOWNTO  0);
        TC10    : OUT   STD_LOGIC
    );

END counter_BCD_1digit;

            -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF counter_BCD_1digit IS
-- Internal wires
                -- State signals declaration
                -- A BCD counter will consist of 10 diferent states
    TYPE State_type IS (Num0, Num1, Num2, Num3, Num4, Num5,
                        Num6, Num7, Num8, Num9) ;

-----> This is important: specifying to the synthesiser the code for the states
    ATTRIBUTE syn_encoding : STRING;
    ATTRIBUTE syn_encoding OF State_type : TYPE IS "sequential";
        -- (It may be:"sequential" (binary); "gray"; "one-hot", etc.
    SIGNAL present_state, next_state    : State_type ;

    -- These two lines work Ok, but if the the state machine is not recognised
    --  is better use enum_encoding:
-- ATTRIBUTE enum_encoding : string;
-- ATTRIBUTE enum_encoding of State_type : TYPE IS "sequential";


-- Constants for special states (the first and the last state)
CONSTANT Reset      : State_type := Num0; -- The fisrt state.
CONSTANT Max_count  : State_type := Num9; -- The last state.

BEGIN
---------- State Register: the only clocked block ----------------------------
--------------------------------------------------------------------------------
--------- The "memory" of the system:
--------- the next events (after a CLK rising edge) will depend on past events
--------- and the current input values

state_register: PROCESS (CD,CLK, next_state)
    BEGIN

    IF CD = '1' THEN            -- reset the state memory
        present_state <= Reset;
    ELSIF (CLK'EVENT and CLK = '1') THEN
        -- Synchronous register (D-type flip-flop)
        present_state <= next_state;
    END IF;

END PROCESS state_register;

---------- CC1: Combinational system for calculating next state ---------------
--------------------------------------------------------------------------------
```

```
74   CC_1: PROCESS (present_state, CE)
75       BEGIN
76           IF CE = '0' THEN
77               next_state <= present_state;  -- count disable
78
79           ELSE                       -- a simple state up count
80               CASE present_state IS
81                   WHEN Num0 =>
82                       next_state <= Num1 ;
83                   WHEN Num1 =>
84                       next_state <= Num2 ;
85                   WHEN Num2 =>
86                       next_state <= Num3 ;
87                   WHEN Num3 =>
88                       next_state <= Num4 ;
89                   WHEN Num4 =>
90                       next_state <= Num5 ;
91                   WHEN Num5 =>
92                       next_state <= Num6 ;
93                   WHEN Num6 =>
94                       next_state <= Num7 ;
95                   WHEN Num7 =>
96                       next_state <= Num8 ;
97                   WHEN Num8 =>
98                       next_state <= Num9 ;
99                   WHEN Num9 =>
100                      next_state <= Num0 ;
101              END CASE ;
102
103          END IF;
104
105  END PROCESS CC_1;
106
107  ---------- CC_2: combinational system for calculating outputs ------------------
108  --------------------------------------------------------------------------------
109  CC_2: PROCESS (present_state, CE)
110      BEGIN
111  -- The terminal count output
112          IF ((present_state = Max_count) AND (CE = '1') ) THEN
113              TC10 <= '1';
114          ELSE
115              TC10 <= '0';
116          END IF;
117
118  -- And now just copying the present state to the output:
119          CASE present_state IS
120                  WHEN Num0 =>
121                      Q <= "0000";
122                  WHEN Num1 =>
123                      Q <= "0001";
124                  WHEN Num2 =>
125                      Q <= "0010";
126                  WHEN Num3 =>
127                      Q <= "0011";
128                  WHEN Num4 =>
129                      Q <= "0100";
130                  WHEN Num5 =>
131                      Q <= "0101";
132                  WHEN Num6 =>
133                      Q <= "0110";
134                  WHEN Num7 =>
135                      Q <= "0111";
136                  WHEN Num8 =>
137                      Q <= "1000";
138                  WHEN Num9 =>
139                      Q <= "1001";
140          END CASE ;
141
142  END PROCESS CC_2;
143
144  -- Place here other logic if necessary
145
146  END FSM_like;
```