```
--------------------------------------------------------------------------------
-- Edge detector
--------------------------------------------------------------------------------
-- Van Winkel Jan
-- Schoeters Jurgen
--
-- Example of a FSM using an enumeration type definition
-- instead of std_logic_vectors for the FSM states
--
-- Example of the FSM using std_logic_vectors can be found
-- on p.40 of the course Introduction to VHDL
-- by lic. ing. Van Landeghem D.
--
-- Enumerations have a big advantage over std_logic_vectors that
-- new states can be easily added. Just add a new
-- name to the definition and add the state to the FSM.
--
-- If you use std_logic_vectors there is a chance that you have to
-- add an extra bit to support all possible states.
-- This will result in much more work, besides adding the new state
-- to the FSM you will have to edit all lines that contain definitions
-- about the states (constant and signal declarations, ...).
--
-- An other advantage of enumerations is that there is an extra level
-- of abstraction (no association with a specific binary number).
-- This will result in more readable code
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity moore_fsm is

  port (
    Clk     : in  std_logic;
    Reset_n : in  std_logic;
    Data    : in  std_logic;
    Puls    : out std_logic);

end moore_fsm;

architecture behave of moore_fsm is

  -- Declaration of the enumeration fsm_states.
  -- The enumeration contains 4 states, where the state w1
  -- is the default state
  type fsm_states is (w1, p1, w0, p0);

  -- Declaration of present and next state signals, which use
  -- the enumeration fsm_states as type
  signal present_state  : fsm_states;
  signal next_state     : fsm_states;

begin

  syn_moore: process (Clk)
  begin

    if Clk'event and Clk = '1' then     -- rising clock edge

      if Reset_n = '0' then

        present_state <= w1;
```

```
      else

        present_state <= next_state;

      end if;

    end if;

  end process syn_moore;

  com_moore: process (present_state, Data)
  begin

    case present_state is

      when w1 => Puls <= '0';

                 if (data='1') then
                   next_state <= p1;
                 else
                   next_state <= w1;
                 end if;

      when p1 => Puls <= '1';
                 next_state <= w0;

      when w0 => Puls <= '0';

                 if (data='1') then
                   next_state <= p0;
                 else
                   next_state <= w0;
                 end if;

      when p0 => Puls <= '1';
                 next_state <= w1;

    end case;

  end process com_moore;

end behave;
```