Example: VHDL model of a state machine

This example will show how a clock-flank-triggered automat with an asynchronous reset can be modelled. Figure 1 shows the interface of an abstract Moore-Automat with two input signals, three output signals, clock and an asynchronous reset signal.

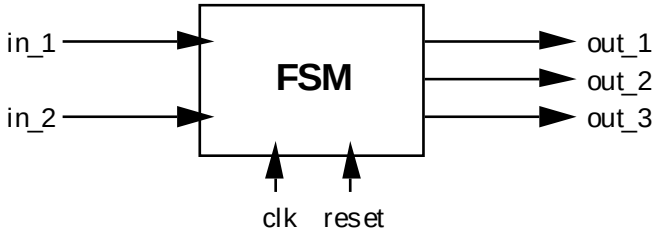

Fig. 1: Interface signals of a Moore-Example-Automat

The automat graph for this example is shown below in figure 2. It is a synchronous, front-flank-triggered Moore-Automat with four states. The single input combinations in_1 and in_2 are marked on the arrows. The output variables out_1, out_2 and out_3 are defined within the states. An asynchronous, low-active reset signal (reset = '0') initiates an instantaneous change to the state init. This is also the starting state.

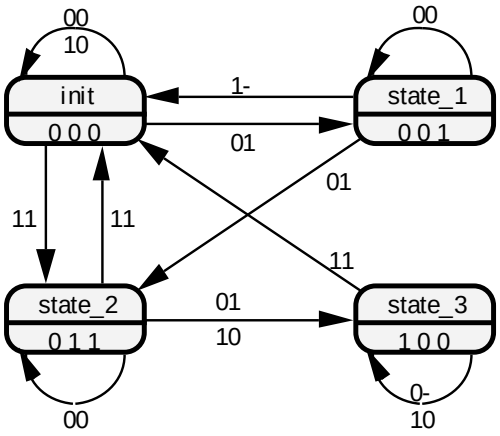

Fig..2: Automat graph of the example automat

To model such a behavior in VHDL, you have to consider the following:

- An own counter-type can be declared for an internal "state-signal" for the four possible states, that receives the single states as possible signal values.
- The flank-triggering of the state changes is done within a process, activated through clk and reset. The query of the active signal flank is done using the signal attributes. It is important that the reset signal has priority!
- Because the automat is a Moore-Type automat, the output variables are assigned according to the state. This can be done in the same process or separately as a process or quasi-parallel signal allocation.

This is the described automat behavior realized in VHDL code:

```
-------------------------------------------------------------------
-- filename:          fsm.vhd
-- title:             Behavioural Model of a simple state machine
-- author:            B. Wunder, ITIV
-------------------------------------------------------------------

ENTITY fsm IS
   PORT (clk, reset : IN bit ;
         in_1, in_2 : IN bit ;
         out_1, out_2, out_3 : OUT bit) ;
   TYPE fsm_state IS (init, state_1, state_2, state_3) ;
END fsm ;

-------------------------------------------------------------------

ARCHITECTURE behavioural OF fsm IS
   SIGNAL state : fsm_state := init ;

BEGIN
   new_state : PROCESS (clk, reset)
   BEGIN
      IF (reset = '0') THEN
         state <= init ;
      ELSIF (clk = '1' AND clk'EVENT) THEN
        CASE state IS
            WHEN init =>
               IF in_1 = '0' AND in_2 = '1' THEN
                  state <= state_1 ;
               ELSIF in_1 = '1' AND in_2 = '1' THEN
                  state <= state_2 ;
               END IF ;
            WHEN state_1 =>
               IF in_1 = '1' THEN
                  state <= init;
               ELSIF in_1 = '0' AND in_2 = '1' THEN
                  state <= state_2 ;
               END IF ;
            WHEN state_2 =>
               IF in_1 = '1' AND in_2 = '1' THEN
                  state <= init ;
               ELSIF (in_1 = '0' AND in_2 = '1')
               OR (in_1 = '1' AND in_2 = '0') THEN
                  state <= state_3 ;
               END IF ;
            WHEN state_3 =>
               IF in_1 = '1' AND in_2 = '1' THEN
                  state <= init ;
               END IF ;
        END CASE ;
      END IF ;
   END PROCESS ;

   output_assignment : PROCESS (state)
   BEGIN
      CASE state IS
         WHEN init =>
            out_1 <= '0' ;
            out_2 <= '0' ;
            out_3 <= '0' ;
         WHEN state_1 =>
            out_1 <= '0' ;
```

```
            out_2 <= '0' ;
            out_3 <= '1' ;
         WHEN state_2 =>
            out_1 <= '0' ;
            out_2 <= '1' ;
            out_3 <= '1' ;
         WHEN state_3 =>
            out_1 <= '1' ;
            out_2 <= '0' ;
            out_3 <= '0' ;
      END CASE ;
   END PROCESS ;
END behavioural ;
```