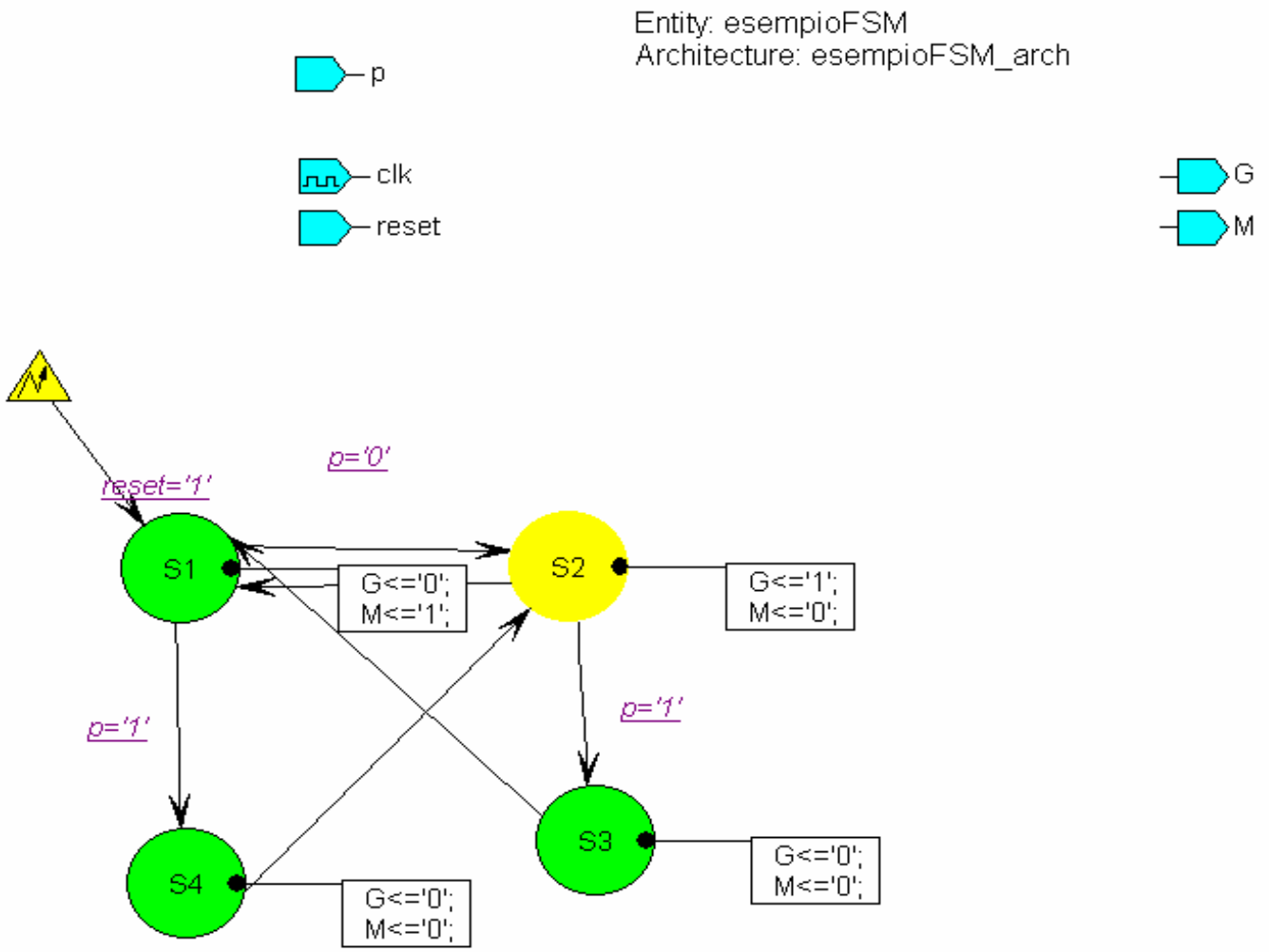
Esempio FSM


Entity: esempioFSM
Architecture: esempioFSM_arch
p
clk
reset
G
M
reset='1'
p='0'
S1
S2
G<='0';
M<='1';
G<='1';
M<='0';
p='1'
p='1'
S3
S4
G<='0';
M<='0';
G<='0';
M<='0';

```
----------------------------------------------------------------------------------
--
-- Description :
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity esempioFSM is
    port (
       clk: in STD_LOGIC;
       p: in STD_LOGIC;
       reset: in STD_LOGIC;
       G: out STD_LOGIC;
       M: out STD_LOGIC);
end;

architecture esempioFSM_arch of esempioFSM is

-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S1, S2, S3, S4);
-- attribute enum_encoding of Sreg0_type: type is ... -- enum_encoding attribute i

signal Sreg0: Sreg0_type;


begin
-- concurrent signals assignments
-- diagram ACTION


-----------------------------------------------------------------------
-- Machine: Sreg0
-----------------------------------------------------------------------
Sreg0_machine: process (clk, reset)
begin
if reset='1' then
    Sreg0 <= S1;
    -- Set default values for registered outputs/signals and for variables
    -- ...
elsif clk'event and clk = '1' then
    -- Set default values for registered outputs/signals and for variables
    -- ...
    case Sreg0 is
       when S1 =>
         if p='1' then
          Sreg0 <= S4;
         elsif p='0' then
          Sreg0 <= S2;
         end if;
       when S2 =>
         if p='1' then
          Sreg0 <= S3;
         elsif p='0' then
          Sreg0 <= S1;
         end if;
       when S3 =>
         Sreg0 <= S1;
       when S4 =>
         Sreg0 <= S2;
       when others =>
         null;
    end case;
end if;
```

```
end process;

-- signal assignment statements for combinatorial outputs
G_assignment:
G <= '0' when (Sreg0 = S1) else
     '1' when (Sreg0 = S2) else
     '0' when (Sreg0 = S3) else
     '0' when (Sreg0 = S4) else
     '0';

M_assignment:
M <= '1' when (Sreg0 = S1) else
     '0' when (Sreg0 = S2) else
     '0' when (Sreg0 = S3) else
     '0' when (Sreg0 = S4) else
     '1';

end esempioFSM_arch;
```

```
-----------------------------------------------------------------
-- An odd parity checker as an FSM using VHDL.
-- Coding style: One process Mealy.
-----------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY fsm IS
  PORT( clk, reset: IN std_logic;
        x: IN std_logic;
        y: OUT std_logic);
END ENTITY fsm;

ARCHITECTURE mealy_2p OF fsm IS
  TYPE state_type IS (even, odd);
  SIGNAL current_state: state_type;
  SIGNAL next_state: state_type;
BEGIN
  cs: PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      current_state <= even;
    ELSIF rising_edge (clk) THEN
      current_state <= next_state;
    END IF;
  END PROCESS cs;
  ns: PROCESS (current_state, x) BEGIN
    CASE current_state IS
      WHEN even =>
        IF x = '1' THEN
          y <= '1';
          next_state <= odd;
        ELSE
          y <= '0';
          next_state <= even;
        END IF;
      WHEN odd =>
        IF x = '1' THEN
          y <= '0';
          next_state <= even;
        ELSE
          y <= '1';
          next_state <= odd;
        END IF;
    END CASE;
  END PROCESS ns;
END ARCHITECTURE mealy_2p;
```

```
-----------------------------------------------------------------
-- An odd parity checker as an FSM using VHDL.
-- Sample textbench
-----------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY fsm_tb IS
END ENTITY fsm_tb;
ARCHITECTURE tb_arch OF fsm_tb IS
  COMPONENT fsm
    PORT(    clk, reset: IN std_logic;
        x: IN std_logic;
        y: OUT std_logic);
  END COMPONENT fsm;
  SIGNAL clk, reset, x, y: std_logic;
  FOR fsm_mealy_2p: fsm USE ENTITY work.fsm(mealy_2p);
BEGIN
  clock: PROCESS IS
  BEGIN
    clk <= '0';
    WAIT FOR 20 ns;
    clk <='1';
    WAIT FOR 20 ns;
  END PROCESS clock;

  sg: PROCESS IS
  BEGIN
    reset <= '1';
    WAIT FOR 10 ns;
    reset <= '0';
    x <= '0';
    WAIT FOR 10 ns;
    ASSERT y = '0'
      REPORT "Error: state even when x = 0"
      SEVERITY warning;
    WAIT FOR 10 ns;
    x <= '1';
    WAIT FOR 10 ns;
    ASSERT y = '1'
      REPORT "Error: state even when x = 1"
      SEVERITY warning;
    WAIT FOR 10 ns;
    x <= '0';
    WAIT FOR 10 ns;
    ASSERT y = '1'
      REPORT "Error: state odd when x = 0"
      SEVERITY warning;
    WAIT FOR 10 ns;
    x <= '1';
    WAIT FOR 10 ns;
    ASSERT y ='0'
      REPORT "Error: state odd when x = 1"
      SEVERITY warning;
    WAIT;
  END PROCESS sg;

  fsm_mealy_2p: fsm PORT MAP (clk, reset, x, y);

END ARCHITECTURE tb_arch;
```

```
1     ------------------------------------------------------------------
      ---------
2     --    Package name: STD_LOGIC_ARITH
      --
3     --
          --
4     --    Purpose:
      --
5     --     A set of arithemtic, conversion, and comparison functions
      --
6     --     for SIGNED, UNSIGNED, SMALL_INT, INTEGER,
      --
7     --     STD_ULOGIC, STD_LOGIC, and STD_LOGIC_VECTOR.
      --
8     ------------------------------------------------------------------
      ---------
9
10    library IEEE;
11    use IEEE.std_logic_1164.all;
12
13    package std_logic_arith is
14
15        type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
16        type SIGNED is array (NATURAL range <>) of STD_LOGIC;
17        subtype SMALL_INT is INTEGER range 0 to 1;
18
19        ----------------
20        -- add operators
21        ----------------
22        function "+"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
23
24        function "+"(L: SIGNED; R: SIGNED) return SIGNED;
25
26        [...]
27
28        function "+"(L: UNSIGNED; R: UNSIGNED) return
      STD_LOGIC_VECTOR;
29
30        function "+"(L: SIGNED; R: SIGNED) return STD_LOGIC_VECTOR;
31
32        [...]
33      ---------------------
34        -- subtract operators
35        ---------------------
36        function "-"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
37
38        [...]
39
40        ------------------
41        -- unary operators
42        ------------------
43        function "+"(L: UNSIGNED) return UNSIGNED;
44
45        function "+"(L: SIGNED) return SIGNED;
46
47        function "-"(L: SIGNED) return SIGNED;
48
49        [...]
50
51        ---------------------------
52        -- multiplication operators
```

```
53          ---------------------------
54          function "*"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
55
56          ------------------------
57          -- less_than comparison
58          ------------------------
59          function "<"(L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
60
61          ------------------------
62          -- conversion operators
63          ------------------------
64          function CONV_INTEGER(ARG: INTEGER) return INTEGER;
65
66          function CONV_INTEGER(ARG: UNSIGNED) return INTEGER;
67
68          function CONV_INTEGER(ARG: SIGNED) return INTEGER;
69
70          [...]
71
72          function CONV_UNSIGNED(ARG: INTEGER; SIZE: INTEGER) return
    UNSIGNED;
73          [...]
74
75        function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
    return STD_LOGIC_VECTOR;
76
77          function CONV_STD_LOGIC_VECTOR(ARG: UNSIGNED; SIZE: INTEGER)
    return STD_LOGIC_VECTOR;
78
79          function CONV_STD_LOGIC_VECTOR(ARG: SIGNED; SIZE: INTEGER)
    return STD_LOGIC_VECTOR;
80
81          function CONV_STD_LOGIC_VECTOR(ARG: STD_ULOGIC; SIZE: INTEGER
    ) return STD_LOGIC_VECTOR;
82
83      end Std_logic_arith;
84
85
86
87      library IEEE;
88      use IEEE.std_logic_1164.all;
89
90      package body std_logic_arith is
91          -- synopsys synthesis_off
92          type tbl_type is array (STD_ULOGIC) of STD_ULOGIC;
93          constant tbl_BINARY : tbl_type :=
94          ('X', 'X', '0', '1', 'X', 'X', '0', '1', 'X');
95          -- synopsys synthesis_on
96
97      function CONV_INTEGER(ARG: UNSIGNED) return INTEGER is
98          variable result: INTEGER;
99          variable tmp: STD_ULOGIC;
100         -- synopsys built_in SYN_UNSIGNED_TO_INTEGER
101         begin
102         -- synopsys synthesis_off
103         assert ARG'length <= 31
104             report "ARG is too large in CONV_INTEGER"
105             severity FAILURE;
106         result := 0;
107         for i in ARG'range loop
108             result := result * 2;
```

```
109           tmp := tbl_BINARY(ARG(i));
110           if tmp = '1' then
111           result := result + 1;
112           elsif tmp = 'X' then
113           assert false
114           report "CONV_INTEGER: There is an 'U'|'X'|'W'|'Z'|'-' in
      an arithmetic operand, and it has been converted to 0."
115           severity WARNING;
116           end if;
117       end loop;
118       return result;
119       -- synopsys synthesis_on
120       end;
```