**Exercise 1**

We enter the state machine on a Reset trigger. In state “S0” the output if “Q” is 0. If, in state “S0,” A is true, then the FSM goes to state “S1.” The output of “S1” is 0 still for “Q.” If B is true during state “S1” then the state goes to “S2” where the output of “Q” is now 1. If B is not true, then the FSM returns to state “S0.” State “S2” immediately returns to state “S0” after Q is marked true. If during state “S0” A is not true, then the FSM stays at state “S0.”

|  |  |  |  |
| --- | --- | --- | --- |
| Current State | Input A | Input B | Next State |
| S0 | 0 | X | S0 |
| S0 | 1 | X | S1 |
| S1 | X | 0 | S0 |
| S1 | X | 1 | S2 |
| S2 | X | X | S0 |

Diagram, schematic

Description automatically generatedOutput Table

|  |  |
| --- | --- |
| State | Output (Q) |
| 00 | 0 |
| 01 | 0 |
| 10 | 1 |

State Encoding

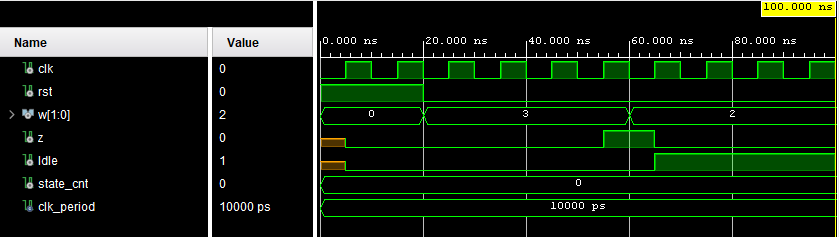
|  |  |
| --- | --- |
| State | Encoding |
| S0 | 00 |
| S1 | 01 |
| S2 | 10 |

S’(0) = (A \* S0) + (B’ \* S1) + (S2)

S’(1) = (S0 \* A)

S’(2) = (S1 \* B)

**Exercise 2**



library ieee;

use ieee.std\_logic\_1164.all;

entity Hw7Q1FSM is port (

clk , rst : in std\_logic;

w : in std\_logic\_vector (1 downto 0);

z , InIdle: out std\_logic);

end entity Hw7Q1FSM;

architecture obehavior of Hw7Q1FSM is

type state\_type is (idle, s1, s2, s3, s4);

signal state, next\_state: state\_type;

begin

state\_proc: process (clk) is begin

if rising\_edge (clk) then

if rst = '1' then

state <= idle;

else

state <= next\_state;

end if;

end if;

end process;

next\_state\_proc: process (state, w) is begin

case state is

when idle => if (w(0) = w(1)) then

next\_state <= s1;

else

next\_state <= idle;

end if;

when s1 => if (w(0) = w(1)) then

next\_state <= s2;

else

next\_state <= idle;

end if;

when s2 => if (w(0) = w(1)) then

next\_state <= s3;

else

next\_state <= idle;

end if;

when s3 => if (w(0) = w(1)) then

next\_state <= s4;

else

next\_state <= idle;

end if;

when s4 => if (w(0) = w(1)) then

next\_state <= s4;

else

next\_state <= idle;

end if;

end case;

end process;

InIdle\_proc: process (clk) is begin

if rising\_edge (clk) then

case next\_state is

when idle => InIdle <= '1';

when others => InIdle <= '0';

end case;

end if;

end process;

z\_proc: process (clk) is begin

if rising\_edge (clk) then

case next\_state is

when s4 => z <= '1';

when others => z <= '0';

end case;

end if;

end process;

end obehavior;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.math\_real.all;

entity Hw7Q1FSMSolnTB is

end Hw7Q1FSMSolnTB;

architecture tb of Hw7Q1FSMSolnTB is

signal clk, rst: std\_logic := '0';

signal w: std\_logic\_vector (1 downto 0) := (others => '0');

signal z, Idle: std\_logic;

constant clk\_period: time := 10 ns;

signal state\_cnt: integer := 0;

begin

dut: entity work.Hw7Q1FSM

port map (clk => clk, rst => rst, w => w,

z => z, InIdle => Idle);

clk <= not clk after clk\_period/2;

stimuli: process is begin

rst <= '1';

wait for clk\_period;

wait until clk = '0';

rst <= '0';

w <= "11";

wait for clk\_period;

assert z = '0' report "w same 1 time, z should be 0";

-- complete. leaving w bits the same for 2 more clk periods

-- check z = 0 after each period

w <= "11";

wait for clk\_period;

assert z = '0' report "w same 2 times, z should be 0";

w <= "11";

wait for clk\_period;

assert z = '0' report "w same 3 times, z should be 0";

w <= "11";

wait for clk\_period;

assert z = '1' report "w same 4 times, z should be 1";

-- complete. making w bits different

-- check Idle is 1

w <= "10";

wait for clk\_period;

assert Idle = '1' report "w different, Idle should be 1 now";

wait;

end process;

end tb;

Diagram, schematic

Description automatically generated

**Exercise 3**

A screenshot of a computer

Description automatically generated

library ieee;

use ieee.std\_logic\_1164.ALL;

use ieee.numeric\_std.all;

entity Hw6Q5Mux is

port (

shift\_amt: in std\_logic\_vector(1 downto 0);

y: out std\_logic\_vector(7 downto 0);

a0: in std\_logic\_vector(7 downto 0);

a1: in std\_logic\_vector(7 downto 0);

a2: in std\_logic\_vector(7 downto 0);

a3: in std\_logic\_vector(7 downto 0)

);

end Hw6Q5Mux;

architecture oh\_behavior of Hw6Q5Mux is

-- create an array of vectors to hold each of n shifters

type shifty\_array is array (3 downto 0) of std\_logic\_vector(7 downto 0);

signal y\_array: shifty\_array := (x"F3", x"A2", x"51", x"00");

begin

y\_array(0) <= a0;

y\_array(1) <= a1;

y\_array(2) <= a2;

y\_array(3) <= a3;

process(shift\_amt, y\_array) is begin

case shift\_amt is

when "00" => y <= y\_array(0);

when "01" => y <= y\_array(1);

when "10" => y <= y\_array(2);

when "11" => y <= y\_array(3);

when others => y <= (others => '0');

end case;

end process;

end oh\_behavior;

library ieee;

use ieee.std\_logic\_1164.all;

entity Exc3tb is

end Exc3tb;

architecture Behavioral of Exc3tb is

signal shift\_amt : std\_logic\_vector(1 downto 0) := (others => '0');

signal a0, a1, a2, a3 : std\_logic\_vector(7 downto 0) := (others => '0');

signal y : std\_logic\_vector(7 downto 0);

begin

dut : entity work.Hw6Q5Mux

port map(

shift\_amt => shift\_amt,

a0 => a0,

a1 => a1,

a2 => a2,

a3 => a3,

y => y

);

stim\_proc : process

begin

a0 <= x"aa";

a1 <= x"bb";

a2 <= x"cc";

a3 <= x"dd";

shift\_amt <= "11";

wait for 10 ns;

assert y = x"dd" report "Error" severity error;

shift\_amt <= "10";

wait for 10 ns;

assert y = x"cc" report "Error" severity error;

shift\_amt <= "01";

wait for 10 ns;

assert y = x"bb" report "Error" severity error;

shift\_amt <= "00";

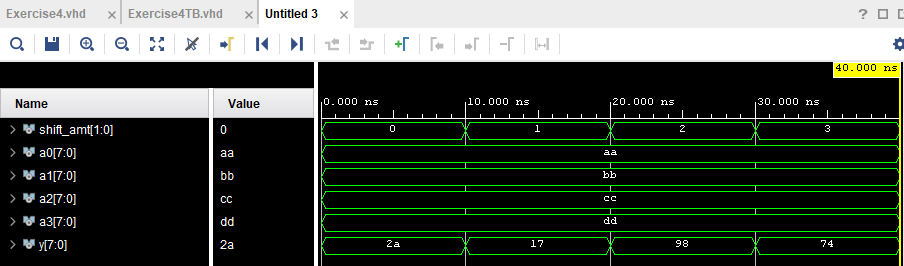
wait for 10 ns;

assert y = x"aa" report "Error" severity error;

end process;

end Behavioral;

**Exercise 4**



The notation 5:0==7:2 is referring to the shift of the contents in 7 downto 2 of the original a3[7:0] value to 5 downto 0. This notation confirms the shift.

library ieee;

use ieee.std\_logic\_1164.ALL;

use ieee.numeric\_std.all;

entity Hw6Q5Mux is

port (

shift\_amt: in std\_logic\_vector(1 downto 0);

a0: in std\_logic\_vector(7 downto 0);

a1: in std\_logic\_vector(7 downto 0);

a2: in std\_logic\_vector(7 downto 0);

a3: in std\_logic\_vector(7 downto 0);

y: out std\_logic\_vector(7 downto 0)

);

end Hw6Q5Mux;

architecture ohbehave of Hw6Q5Mux is

-- create array of vectors to hold each of n shifters

type shifty\_array is array (3 downto 0) of std\_logic\_vector(7 downto 0);

signal y\_array: shifty\_array := (x"F3", x"A2", x"51", x"00");

begin

y\_array(0) <= "00" & a0(7 downto 2);

y\_array(1) <= "000" & a1(7 downto 3);

y\_array(2) <= a2(6 downto 0) & '0';

y\_array(3) <= a3(5 downto 0) & "00";

process(shift\_amt, y\_array) is begin

case shift\_amt is

when "00" => y <= y\_array(0);

when "01" => y <= y\_array(1);

when "10" => y <= y\_array(2);

when "11" => y <= y\_array(3);

when others => y <= (others => '0');

end case;

end process;

end ohbehave;

library ieee;

use ieee.std\_logic\_1164.all;

entity Exc3tb is

end Exc3tb;

architecture Behavioral of Exc3tb is

signal shift\_amt : std\_logic\_vector(1 downto 0) := (others => '0');

signal a0, a1, a2, a3 : std\_logic\_vector(7 downto 0) := (others => '0');

signal y : std\_logic\_vector(7 downto 0);

begin

dut : entity work.Hw6Q5Mux

port map(

shift\_amt => shift\_amt,

a0 => a0,

a1 => a1,

a2 => a2,

a3 => a3,

y => y

);

stim\_proc : process

begin

a0 <= x"aa";

a1 <= x"bb";

a2 <= x"cc";

a3 <= x"dd";

shift\_amt <= "00";

wait for 10 ns;

assert y = "00101010" report "Error" severity error;

shift\_amt <= "01";

wait for 10 ns;

assert y = "00010111" report "Error" severity error;

shift\_amt <= "10";

wait for 10 ns;

assert y = "10011000" report "Error" severity error;

shift\_amt <= "11";

wait for 10 ns;

assert y = "01110100" report "Error" severity error;

end process;

end Behavioral;