Electrical and Computer Engineering

State Machine Hardware Description Language

By Prawat Nagvajara

Synopsis

This note covers a Hardware Description Language design of a state machine using the classic sequence detector as for example.

Schematic

A state machine comprises a register storing the present state, the next state logic and the output logic (see Fig. 1).

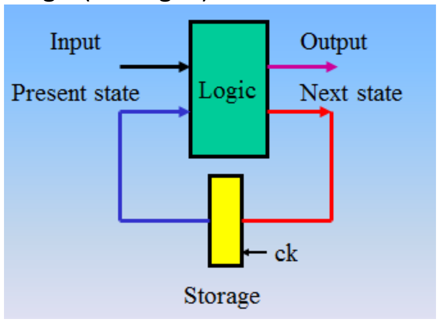


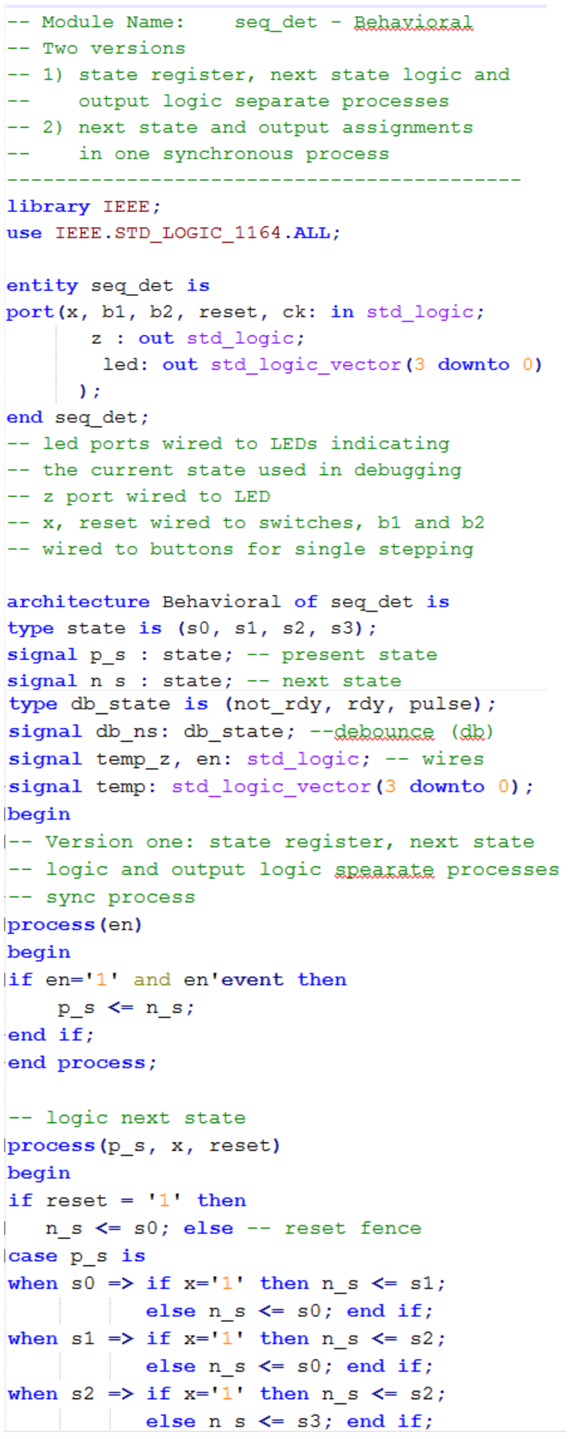
Fig. 1 State Machine

Code

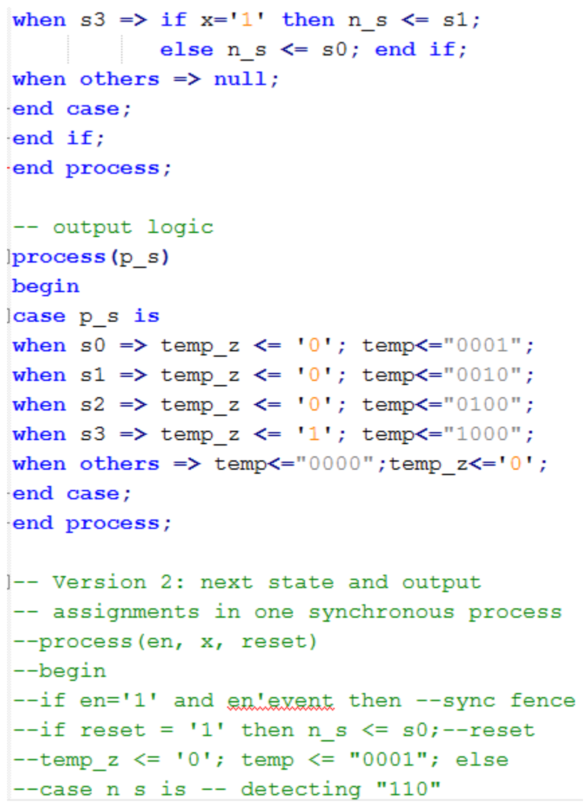
An example sequence detector is a state machine where the output Z is a one when user enters a sequence "110" of the input X.

In Version 1 of the code the state machine consists of the state storage process synchronized to clock rising edge. The design uses the single-pulse signal EN generated by a process synchronized to the clock signal CK (the last process in the code). This allows the input X to be entered using a switch and two push buttons B1 and B2. B1 press generates a pulse and B2 resets the mechanism. In addition, Version 1 describes the design using the next state logic process and the output logic (these represent combinational circuits).

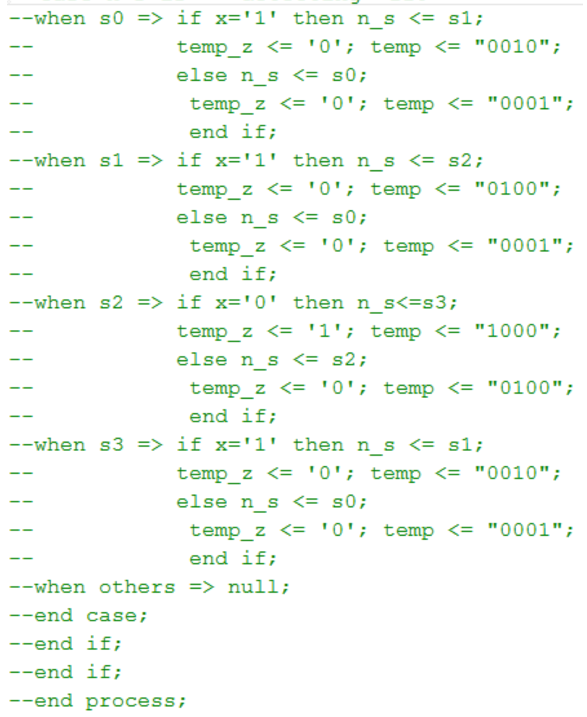
In Version 2 the design consists of a process that describes the next state transition and the output synchronized to EN.



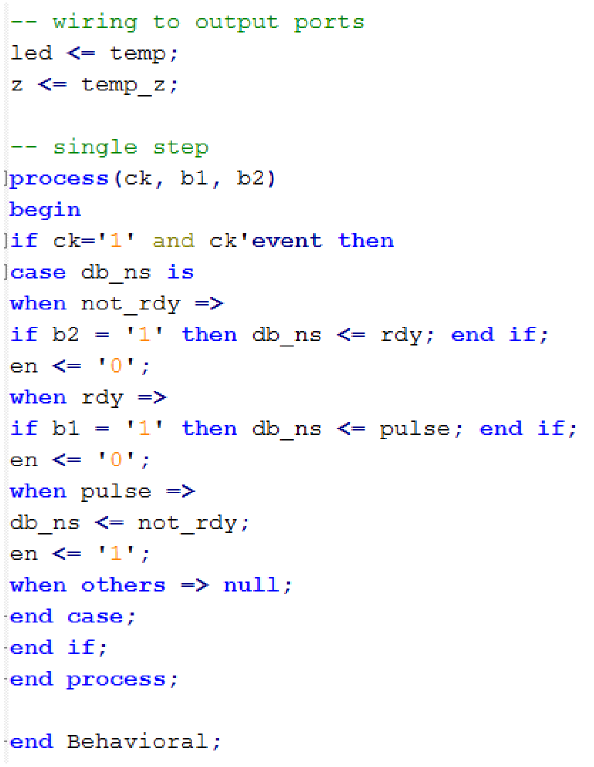
```
-- Module Name:    seq_det - Behavioral
-- Two versions
-- 1) state register, next state logic and
--    output logic separate processes
-- 2) next state and output assignments
--    in one synchronous process
-------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity seq_det is
port(x, b1, b2, reset, ck: in std_logic;
       z : out std_logic;
        led: out std_logic_vector(3 downto 0)
      );
end seq_det;
-- led ports wired to LEDs indicating
-- the current state used in debugging
-- z port wired to LED
-- x, reset wired to switches, b1 and b2
-- wired to buttons for single stepping

architecture Behavioral of seq_det is
type state is (s0, s1, s2, s3);
signal p_s : state; -- present state
signal n_s : state; -- next state
type db_state is (not_rdy, rdy, pulse);
signal db_ns: db_state; --debounce (db)
signal temp_z, en: std_logic; -- wires
signal temp: std_logic_vector(3 downto 0);
begin
-- Version one: state register, next state
-- logic and output logic spearate processes
-- sync process
process(en)
begin
if en='1' and en'event then
    p_s <= n_s;
end if;
end process;

-- logic next state
process(p_s, x, reset)
begin
if reset = '1' then
   n_s <= s0; else -- reset fence
case p_s is
when s0 => if x='1' then n_s <= s1;
           else n_s <= s0; end if;
when s1 => if x='1' then n_s <= s2;
           else n_s <= s0; end if;
when s2 => if x='1' then n_s <= s2;
           else n_s <= s3; end if;
```



```
when s3 => if x='1' then n_s <= s1;
           else n_s <= s0; end if;
when others => null;
end case;
end if;
end process;

-- output logic
process(p_s)
begin
case p_s is
when s0 => temp_z <= '0'; temp<="0001";
when s1 => temp_z <= '0'; temp<="0010";
when s2 => temp_z <= '0'; temp<="0100";
when s3 => temp_z <= '1'; temp<="1000";
when others => temp<="0000";temp_z<='0';
end case;
end process;

-- Version 2: next state and output
-- assignments in one synchronous process
--process(en, x, reset)
--begin
--if en='1' and en'event then --sync fence
--if reset = '1' then n_s <= s0;--reset
--temp_z <= '0'; temp <= "0001"; else
--case n_s is -- detecting "110"
```



```
--when s0 => if x='1' then n_s <= s1;
--            temp_z <= '0'; temp <= "0010";
--            else n_s <= s0;
--             temp_z <= '0'; temp <= "0001";
--             end if;
--when s1 => if x='1' then n_s <= s2;
--            temp_z <= '0'; temp <= "0100";
--            else n_s <= s0;
--             temp_z <= '0'; temp <= "0001";
--             end if;
--when s2 => if x='0' then n_s<=s3;
--            temp_z <= '1'; temp <= "1000";
--            else n_s <= s2;
--             temp_z <= '0'; temp <= "0100";
--             end if;
--when s3 => if x='1' then n_s <= s1;
--            temp_z <= '0'; temp <= "0010";
--            else n_s <= s0;
--             temp_z <= '0'; temp <= "0001";
--             end if;
--when others => null;
--end case;
--end if;
--end if;
--end process;
```



```
-- wiring to output ports
led <= temp;
z <= temp_z;

-- single step
process(ck, b1, b2)
begin
if ck='1' and ck'event then
case db_ns is
when not_rdy =>
if b2 = '1' then db_ns <= rdy; end if;
en <= '0';
when rdy =>
if b1 = '1' then db_ns <= pulse; end if;
en <= '0';
when pulse =>
db_ns <= not_rdy;
en <= '1';
when others => null;
end case;
end if;
end process;

end Behavioral;
```

Exploration

Design a sequence detector to detect a sequence of hexadecimal numbers “ABCD”.