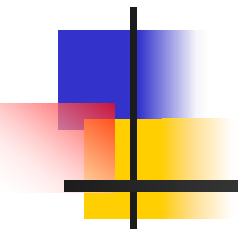

VHDL
Examples

Example 1
Odd Parity Generator

```
--- This module has two inputs, one output and one process.
--- The clock input and the input_stream are the two inputs. Whenever the clock
--- goes high then there is a loop which checks for the odd parity by using
--- the xor logic.There is package anu which is used to declare the port
--- input_stream.One can change the value of m where it is declared as constant
--- and the input array can vary accordingly.
--------------------------------------------------------------------------------
package anu is
constant m: integer :=8;
type input is array (0 to m-1) of bit;
end anu;

library ieee;
use ieee.std_logic_1164.all ;
use Work.anu.all;

entity Parity_Generator1 is
        port (    input_stream : in input;
                  clk : in std_logic ;
                  parity :out bit );
end Parity_Generator1;
```

Example 1
Odd Parity Generator (cont'd)

```
architecture odd of Parity_Generator1 is

begin

  P1: process

  variable odd : bit ;

  begin
       wait until clk'event and clk = '1';
          odd := '0';

            for I in 0 to m-1 loop
                  odd := odd xor input_stream (I);
            end loop;
       parity <= odd;
  end process;

end odd;
```

Example 1

Odd Parity Generator - Testbench

```
--- This structural code instantiate the ODD_PARITY_TB module to create a
--- testbench for the odd_parity_TB design. The processes in it are the ones
--- that create the clock and the input_stream.Explore the design in the
--- debugger by either adding to the testbench to provide stimulus for the
--- design or use assign statements in the simulator.If you want to change the
--- array width you will have to modify the a3.vhd code too by changing the
--- value of m.
-------------------------------------------------------------------------------
entity ODD_PARITY_TB is
end;
library ieee;
use ieee.std_logic_1164.all;
use WORK.anu.all;

architecture OP_TB_ARCH of ODD_PARITY_TB is

component Parity_Generator1
        port (input_stream : in input;
                    clk : in std_logic;
                    parity : out bit );
end component;
```

Example 1

Odd Parity Generator – Testbench (cont'd)

```
signal input_stream : input;
signal clk :std_logic;
signal parity :bit ;
begin
U1: Parity_Generator1
        port map(
                input_stream,
            clk,
                parity => parity
                );

input1 : process (clk)

begin
        if clk <= 'U' then clk <= '0' after 1 ns;
        else clk <= not clk after 1 ns;
        end if;
end process;
```

Example 1
Odd Parity Generator – Testbench (cont'd)

```
input2: process (input_stream)
begin
        input_stream <= "10100110" after 1 ns,
                                   "01111100" after 2 ns;
end process;
end OP_TB_ARCH;

configuration cfg_op of ODD_PARITY_TB is
for OP_TB_ARCH
end for;
end cfg_op;
```

Example 2
Pulse Generator

```
library IEEE;
use IEEE.std_logic_1164.all;

entity P_GENERATOR is
      port (   CLK          : in std_ulogic;
               RESET        : in std_ulogic;
               TRIG         : in std_ulogic;
               PULSE        : out std_ulogic);
end P_GENERATOR;

architecture STATE_MACHINE of P_GENERATOR is

type PULSEGEN_STATE_TYPE is (IDLE, GEN_PULSE_A, GEN_PULSE_B,
                                    END_PULSE, RETRIGGER);   -- enumeration type
                                                             -- declaration.
signal CURRENT_STATE, NEXT_STATE: PULSEGEN_STATE_TYPE;

signal                  COUNT :     integer range 0 to 31;
constant     WIDTH :    integer range 0 to 31 := 4;
```

Example 2
Pulse Generator (cont'd)

```
begin

STATE_MACH_PROC : process (CURRENT_STATE, TRIG, COUNT)  -- sensitivity list.

begin
    case CURRENT_STATE is        -- case-when statement specifies the following set of
                                 -- statements to execute based on the value of
                                 -- CURRENT_SIGNAL
    when IDLE            =>      if TRIG='1' then
                                         NEXT_STATE <= GEN_PULSE_A;
                                 end if;
    when GEN_PULSE_A =>          if COUNT = WIDTH then
                                         NEXT_STATE <= END_PULSE;
                                 elsif TRIG='0' then
                                         NEXT_STATE <= GEN_PULSE_B;
                                 end if;
    when END_PULSE    =>         if TRIG ='1' then
                                         NEXT_STATE <= IDLE;
                                 end if;
```

Example 2
Pulse Generator (cont'd)

```
when GEN_PULSE_B =>  if TRIG = '1' then
                                        NEXT_STATE <= RETRIGGER;

                     elsif COUNT=WIDTH then
                                        NEXT_STATE <= IDLE;
                     end if;

     when RETRIGGER    =>              NEXT_STATE <= GEN_PULSE_A;

     when OTHERS       =>              NEXT_STATE <= NEXT_STATE;

end case;

end process STATE_MACH_PROC;
```

Example 2
Pulse Generator (cont'd)

```
PULSE_PROC : process (CLK, RESET)                    -- sensitivity list

begin

if RESET = '1' then
      PULSE              <= '0';
      COUNT              <= 0;
      CURRENT_STATE      <= IDLE;

elsif (clk='1' and clk'event) then  -- clk'event is event attribute of clk to
                                    -- determine if a clock has transitioned
      CURRENT_STATE <= NEXT_STATE;
      case NEXT_STATE is
            when IDLE            => PULSE <= '0';
                                    COUNT <= 0;

            when GEN_PULSE_A     => PULSE <= '1';
                                    COUNT <= COUNT + 1;
```

Example 2
Pulse Generator (cont'd)

```
                when END_PULSE                  => PULSE <= '0';
                                                   COUNT <= 0;

                when GEN_PULSE_B      => PULSE <= '1';
                                                   COUNT <= COUNT + 1;

                when RETRIGGER                  => COUNT <= 0;

                when OTHERS                     => COUNT <= COUNT;

        end case;
end if;
end process PULSE_PROC;

end STATE_MACHINE;
```

Example 2

Pulse Generator - Testbench

```
entity STATE_MACHINE_TB is
end STATE_MACHINE_TB;

library IEEE;
use IEEE.std_logic_1164.all;

architecture ARC_STATE_MACHINE_TB of STATE_MACHINE_TB is

component P_GENERATOR

port (          CLK     : in std_ulogic;
            RESET   : in std_ulogic;
            TRIG    : in std_ulogic;
            PULSE   : out std_ulogic);
end component;

signal                      CLK     : std_ulogic;
signal      RESET    : std_ulogic;
signal      TRIG     : std_ulogic;
signal                      PULSE : std_ulogic;
```

Example 2

Pulse Generator – Testbench (cont'd)

```
begin

U1: P_GENERATOR
port map(       CLK, RESET,TRIG,PULSE);

CREATE_CLOCK:           process (clk)
 begin

      if clk <= 'U' then clk <= '0' after 1 ns;
                else clk <= not clk after 1 ns;
      end if;

end process CREATE_CLOCK;

CREATE_PULSE:           process (TRIG)

begin
      TRIG <= '0' after 10 ns,
                '1' after 15 ns,
                '0' after 20 ns;
```

Example 2

Pulse Generator – Testbench (cont'd)

```
end process CREATE_PULSE;

end ARC_STATE_MACHINE_TB;

configuration CFG_STATE_MACHINE of STATE_MACHINE_TB is

    for ARC_STATE_MACHINE_TB
    end for;

end  CFG_STATE_MACHINE;
```

Example 3
Priority Encoder

```
entity priority is
        port (I :       in bit_vector(7 downto 0);      --inputs to be prioritised
              A :       out bit_vector(2 downto 0);     --encoded output
              GS :      out bit);                       --group signal output
end priority;

architecture v1 of priority is
begin
        process (I)
        begin
                GS <= '1'; --set default outputs
                A <= "000";
                if I(7) = '1' then
                        A <= "111";
                elsif I(6) = '1' then
                        A <= "110";
```

Example 3
Priority Encoder (cont'd)

```
            elsif I(5) = '1' then
                        A <= "101";
            elsif I(4) = '1' then
                        A <= "100";
            elsif I(3) = '1' then
                        A <= "011";
            elsif I(2) = '1' then
                        A <= "010";
            elsif I(1) = '1' then
                        A <= "001";
            elsif I(0) = '1' then
                        A <= "000";
            else
                        GS <= '0';
            end if;
      end process;
end v1;
```

Example 4

Behavioral Model for 16 word, 8 bit RAM

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ram16x8 IS
      PORT (address :                  IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            csbar, oebar, webar :      IN STD_LOGIC;
            data :                     INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ram16x8;

ARCHITECTURE version1 OF ram16x8 IS
      BEGIN
            PROCESS (address, csbar, oebar, webar, data)
                  TYPE ram_array IS ARRAY (0 TO 15) OF BIT_VECTOR(7 DOWNTO 0);
                  VARIABLE index : INTEGER := 0;
                  VARIABLE ram_store : ram_array;
            BEGIN
                  IF csbar = '0' THEN
```

Example 4

Behavioral Model for 16 word, 8 bit RAM (cont'd)

```
        --calculate address as an integer
                index := 0;
                FOR i IN address'RANGE LOOP
                        IF address(i) = '1' THEN
                                index := index + 2**i;
                        END IF;
                END LOOP;
        IF rising_edge(webar) THEN
                --write to ram on rising edge of write pulse
                ram_store(index) := To_bitvector(data);
        ELSIF oebar = '0' THEN
                data <= To_StdlogicVector(ram_store(index));
        ELSE
                 data <= "ZZZZZZZZ";
        END IF;
    ELSE
        data <= "ZZZZZZZZ";
    END IF;
END PROCESS;
END version1;
```

Example 5
Incrementer - entity

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

entity incrementer is
   generic (width : integer := 8);
   port ( datain: in std_logic_vector(width-1 downto 0);
          control: in std_logic;
          dataout: out std_logic_vector(width-1 downto 0);
          flag: out std_logic);
end incrementer;
```

Example 5
Incrementer - architecture

```
architecture behv of incrementer is

signal dataout_int: std_logic_vector (width-1 downto 0);
begin

   process (datain, control)
   begin
      if (control = '1') then -- increment
         dataout_int <= datain + '1';
      else -- feedthrough
         dataout_int <= datain;
      end if;
   end process;
   dataout <= dataout_int;
   flag <= '1' when (control = '1' and datain =  To_std_logic_vector(X"FF")) else '0';

end behv;
```

Example 5
Incrementer - Testbench

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

entity tb_inc is
   generic (width : integer := 8);
end tb_inc;

architecture behv of tb_inc is

-- define the internal signal which are connected to the UUT

signal datain: std_logic_vector(width-1 downto 0);
signal control: std_logic;
signal dataout: std_logic_vector(width-1 downto 0);
signal flag: std_logic;
```

Example 5

Incrementer – Testbench (cont'd)

```
-- component declaration : required to define the interface of
-- the instantiated component
component incrementer
   generic (width : integer);
   port ( datain: in std_logic_vector(width-1 downto 0);
          control: in std_logic;
          dataout: out std_logic_vector(width-1 downto 0);
          flag: out std_logic);
end component;

begin

-- Process statement providing stimuli to UUT

P1: process
begin
   wait for 2 ns;
   control <= '1'; -- increment mode
   loop1_260: for i in 0 to 259 loop
        datain <= conv_std_logic_vector(i, width);
        wait for 10 ns;
```

Example 5
Incrementer – Testbench (cont'd)

```
 end loop;
   control <= '0'; -- feedthrough mode
   loop2_260: for i in 0 to 259 loop
       datain <= conv_std_logic_vector(i, width);
       wait for 10 ns;
   end loop;

end process;

---------- Instantiating the component for testing

I1: incrementer generic map (width => width)
    port map (datain => datain, control => control, dataout => dataout, flag => flag);

end behv;
```

Example 5

Incrementer – Testbench (cont'd)

```
-- Configuration declaration to bind component declaration to entity-architecture

configuration CFG_top of tb_inc is
for behv
  for l1: incrementer use entity work.incrementer(behv);
  end for;
end for;
end CFG_top;
```

Example 6
Barrel Shifter - entity

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

entity bs_vhdl is

port (        datain: in std_logic_vector(31 downto 0);
      direction: in std_logic;
      rotation : in std_logic;
      count: in std_logic_vector(4 downto 0);
      dataout: out std_logic_vector(31 downto 0));
end bs_vhdl;
```

Example 6
Barrel Shifter - architecture

```
architecture behv of bs_vhdl is

-- SHIFT LEFT/RIGHT FUNCTION

function barrel_shift(din: in std_logic_vector(31 downto 0);
dir: in std_logic;
cnt: in std_logic_vector(4 downto 0)) return std_logic_vector is
begin

if (dir = '1') then
return std_logic_vector((SHR(unsigned(din), unsigned(cnt))));
else
return std_logic_vector((SHL(unsigned(din), unsigned(cnt))));
end if;
end barrel_shift;
```

Example 6
Barrel Shifter – architecture (cont'd)

```
-- ROTATE LEFT/RIGHT FUNCTION

function barrel_rotate(din: in std_logic_vector(31 downto 0);
dir: in std_logic;
cnt: in std_logic_vector(4 downto 0)) return std_logic_vector is
variable temp1, temp2: std_logic_vector(63 downto 0);

begin

case dir is

when '1' => -- rotate right cnt times
temp1 := din & din;
temp2 := std_logic_vector(SHR(unsigned(temp1),unsigned(cnt)));
return temp2(31 downto 0);
```

Example 6

Barrel Shifter – architecture (cont'd)

```
when others => -- rotate left cnt times
temp1 := din & din;
temp2 := std_logic_vector(SHL(unsigned(temp1),unsigned(cnt)));
return temp2(63 downto 32);
end case;

end barrel_rotate;

begin

P1: process (datain, direction, rotation, count)
begin

if (rotation = '0') then -- shift only
      dataout <= barrel_shift(datain, direction, count);
else -- rotate only
      dataout <= barrel_rotate(datain, direction, count);
end if;

end process;
end behv;
```

Example 6
Barrel Shifter – Testbench

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

entity tb_bs is
end tb_bs;

architecture behv of tb_bs is

-- Instantiating the UUT

component bs_vhdl
port (          datain: in std_logic_vector(31 downto 0);
       direction: in std_logic;
       rotation : in std_logic;
       count: in std_logic_vector(4 downto 0);
       dataout: out std_logic_vector(31 downto 0));
end component;
```

Example 6
Barrel Shifter – Testbench (cont'd)

```
-- Defining the signals connected to the UUT

signal datain: std_logic_vector(31 downto 0);
signal direction: std_logic;
signal rotation : std_logic;
signal count: std_logic_vector(4 downto 0);
signal dataout: std_logic_vector(31 downto 0);

begin

-- Instantiating the UUT

I1: bs_vhdl port map (datain => datain,
direction => direction,
rotation => rotation,
count => count,
dataout => dataout);
```

Example 6
Barrel Shifter – Testbench (cont'd)

```
-- Applying Stimuli

P1: process
begin

wait for 2 ns;
rotation <= '0'; -- shift mode
datain <=  To_std_logic_vector(X"AAAAAAAA");
direction <= '0'; -- LEFT
loop1: for i in 0 to 31 loop
count <= conv_std_logic_vector(i, 5);
wait for 10 ns;
end loop;
```

Example 6
Barrel Shifter – Testbench (cont'd)

```
direction <= '1'; -- RIGHT
loop3: for i in 0 to 31 loop
count <= conv_std_logic_vector(i, 5);
wait for 10 ns;
end loop;

direction <= '1'; -- RIGHT
rotation <= '1'; -- barrel shift
datain <=  To_std_logic_vector(X"55555555");
loop2: for i in 0 to 31 loop
count <= conv_std_logic_vector(i, 5);
wait for 10 ns;
end loop;

direction <= '0'; -- LEFT
loop4: for i in 0 to 31 loop
count <= conv_std_logic_vector(i, 5);
wait for 10 ns;
end loop;
```

Example 6
Barrel Shifter – Testbench (cont'd)

```
end process;
end behv;

-- TOP LEVEL CONFIGURATION DECLARATION

configuration CFG_top of tb_bs is
for behv
for I1: bs_vhdl use entity work.bs_vhdl(behv);
end for;
end for;
end CFG_top;
```

Example 7
BCD to 7-Seg Decoder – entity

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

entity DISPLAY_DECODER is
      port (  VALUE                 : in bit_vector(3 downto 0);  -- Bit 3 is MSB
              ZERO_BLANK            : in bit;
              DISPLAY               : out bit_vector(6 downto 0); -- 7 bit signal
              ZERO_BLANK_OUT        : out bit);
end DISPLAY_DECODER;
```

Example 7

BCD to 7-Seg Decoder – architecture

```
architecture BEHAVIOUR of DISPLAY_DECODER is

begin
process (VALUE, ZERO_BLANK)        -- sensitivity list

begin

case VALUE is                      -- case-when statement described how decode is
                                   -- driven based on the value of the input.
    when "0000"      =>            if ZERO_BLANK='1' then
                                         DISPLAY <= "0000000";
                                         ZERO_BLANK_OUT <= '1';
                                   else
                                         DISPLAY <= "1111110";
                                   end if;
    when "0001"      =>            DISPLAY <= "0110000";
    when "0010"      =>            DISPLAY <= "1101101";
```

Example 7

BCD to 7-Seg Decoder – architecture (cont'd)

```
        when "0011"     =>      DISPLAY <= "1111001";
        when "0100"     =>      DISPLAY <= "0110011";
        when "0101"     =>      DISPLAY <= "1011011";
        when "0110"     =>      DISPLAY <= "1011111";
        when "0111"     =>      DISPLAY <= "1110000";
        when "1000"     =>      DISPLAY <= "1111111";
        when OTHERS     =>      DISPLAY <= "1001111"; -- when others, an error
                                -- is specified

    end case;
    end process;
    end BEHAVIOUR;
```

Example 7

BCD to 7-Seg Decoder – Testbench (cont'd)

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

entity DISPLAY_DECODER_TB is
end DISPLAY_DECODER_TB;

architecture ARC_DISPLAY_DECODER_TB of DISPLAY_DECODER_TB is

signal          VALUE        : bit_vector(3 downto 0);
signal      ZERO_BLANK     : bit;
signal      DISPLAY        : bit_vector(6 downto 0);
signal      ZERO_BLANK_OUT  : bit;

component DISPLAY_DECODER
    port  (  VALUE         : in bit_vector(3 downto 0);
             ZERO_BLANK     : in bit;
             DISPLAY        : out bit_vector(6 downto 0);
             ZERO_BLANK_OUT  : out bit);
end component;
```

Example 7

BCD to 7-Seg Decoder – Testbench (cont'd)

```
begin

INPUT_VALUES: process

begin

    ZERO_BLANK          <= '1';
    VALUE               <= "0000";

    wait for 5 ns;

    ZERO_BLANK      <= '0';
    VALUE           <= "0000";

    wait for 7 ns;

    ZERO_BLANK          <= '1';
    VALUE               <= "0010";


    wait for 12 ns;
```

Example 7
BCD to 7-Seg Decoder – Testbench (cont'd)

```
ZERO_BLANK      <= '0';
      VALUE                         <= "0100";

      wait for 12 ns;

      ZERO_BLANK      <= '0';
      VALUE                         <= "0110";

      wait for 7 ns;

end process INPUT_VALUES;

U1: DISPLAY_DECODER
      port map(VALUE, ZERO_BLANK, DISPLAY, ZERO_BLANK_OUT);

end ARC_DISPLAY_DECODER_TB;
```

Example 7

BCD to 7-Seg Decoder – Testbench (cont'd)

```
configuration CFG_DISPLAY_DECODER of DISPLAY_DECODER_TB is

for ARC_DISPLAY_DECODER_TB
    for U1:DISPLAY_DECODER use entity
        work.DISPLAY_DECODER(BEHAVIOUR);
    end for;
end for;

end CFG_DISPLAY_DECODER;
```

Example 8
Mealy Machine

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity MEALY is            -- Mealy machine
  port (X, CLOCK: in STD_LOGIC;
        Z: out STD_LOGIC);
end;

architecture BEHAVIOR of MEALY is
  type STATE_TYPE is (S0, S1, S2, S3);
  signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
begin
```

Example 8
Mealy Machine (cont'd)

```
-- Process to hold combinational logic.
  COMBIN: process (CURRENT_STATE, X)
  begin
    case CURRENT_STATE is
      when S0 =>
        if X = '0' then
          Z <= '0';
          NEXT_STATE <= S0;
        elsif X = '1' then
          Z <= '1';
          NEXT_STATE <= S2;
        else
          Z <= 'U';
          NEXT_STATE <= S0;
        end if;
```

Example 8
Mealy Machine (cont'd)

```
when S1 =>
      if X = '0' then
        Z <= '0';
        NEXT_STATE <= S0;
      elsif X = '1' then
        Z <= '0';
        NEXT_STATE <= S2;
      else
        Z <= 'U';
        NEXT_STATE <= S0;
      end if;
    when S2 =>
      if X = '0' then
        Z <= '1';
        NEXT_STATE <= S2;
      elsif X = '1' then
        Z <= '0';
        NEXT_STATE <= S3;
      else
```

Example 8
Mealy Machine (cont'd)

```
        Z <= 'U';
        NEXT_STATE <= S0;
      end if;
     when S3 =>
      if X = '0' then
        Z <= '0';
        NEXT_STATE <= S3;
      elsif X = '1' then
        Z <= '1';
        NEXT_STATE <= S1;
      else
        Z <= 'U';
        NEXT_STATE <= S0;
      end if;
  end case;
end process;
```

Example 8
Mealy Machine (cont'd)

```
 -- Process to hold synchronous elements (flip-flops)
  SYNCH: process
  begin
    wait until CLOCK'event and CLOCK = '1';
    CURRENT_STATE <= NEXT_STATE;
  end process;
end BEHAVIOR;
```

Example 8
Mealy Machine - Testbench

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity TB_MEALY is
end;

architecture TESTBENCH of TB_MEALY is

    signal CLK   : std_logic;
    signal X     : std_logic;
    signal Z     : std_logic;

    component MEALY
            Port ( X, CLOCK: in STD_LOGIC;
                   Z: out STD_LOGIC
                 );
    end component;
```

Example 8

Mealy Machine - Testbench (cont'd)

```
begin
        UUT : MEALY
                Port Map (X, CLK, Z);

-- CLOCK STIMULI OF 100 NS TIME PERIOD
        CLOCK: process
        begin
        CLK <= '0', '1' after 50 ns;
                wait for 100 ns;
        end process;

-- X input STIMULI
        X_Stimuli: process
        begin
                X <= '0', '1' after 30 ns,
                 'U' after 60 ns;
                wait for 90 ns;
        end process;
```

Example 8
Mealy Machine - Testbench (cont'd)

```
end TESTBENCH;


configuration CFG_TB_MEALY of TB_MEALY is
        for TESTBENCH
                for UUT : MEALY
                end for;
        end for;
end;
```

Example 9
Moore Machine

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity MOORE is                -- Moore machine
  port (X, CLOCK: in STD_LOGIC;
        Z: out STD_LOGIC);
end;

architecture BEHAVIOR of MOORE is
  type STATE_TYPE is (S0, S1, S2, S3);
  signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
begin

  -- Process to hold combinational logic
  COMBIN: process (CURRENT_STATE, X)
  begin
    case CURRENT_STATE is
```

Example 9
Moore Machine (cont'd)

```
 when S0 =>
       Z <= '0';
       if X = '0' then
         NEXT_STATE <= S0;
       else
         NEXT_STATE <= S2;
       end if;
when S1 =>
       Z <= '1';
       if X = '0' then
         NEXT_STATE <= S0;
       else
         NEXT_STATE <= S2;
       end if;
when S2 =>
       Z <= '1';
       if X = '0' then
         NEXT_STATE <= S2;
       else
         NEXT_STATE <= S3;
       end if;
```

Example 9
Moore Machine (cont'd)

```
when S3 =>
      Z <= '0';
      if X = '0' then
        NEXT_STATE <= S3;
      else
        NEXT_STATE <= S1;
      end if;
   end case;
  end process;


  -- Process to hold synchronous elements (flip-flops)
  SYNCH: process
  begin
    wait until CLOCK'event and CLOCK = '1';
    CURRENT_STATE <= NEXT_STATE;
  end process;
end BEHAVIOR;
```

Example 9
Moore Machine - Testbench

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity TB_MOORE is
end;

architecture TESTBENCH of TB_MOORE is

    signal CLK   : std_logic;
    signal X     : std_logic;
    signal Z     : std_logic;

    component MOORE
            Port (
                        X, CLOCK: in STD_LOGIC;
                Z: out STD_LOGIC
            );
    end component;
```

Example 9

Moore Machine – Testbench (cont'd)

```
procedure check(signal Z : in std_logic;
                constant Expected : in std_logic;
                constant timepoint : in time) is
    begin
        assert ( Z /= Expected OR timepoint /= now )
        report "Value on Z is OK"
        severity NOTE;
    end;

begin
    UUT : MOORE
        Port Map (X, CLK, Z);

-- CLOCK STIMULI OF 100 NS TIME PERIOD

    CLOCK: process
    begin
        CLK <= '0', '1' after 50 ns;
        wait for 100 ns;
end process;
```

Example 9
Moore Machine – Testbench (cont'd)

```
-- X input STIMULI

        X_Stimuli: process
        begin
                X <= '1', '0' after 1000 ns;
                wait for 2000 ns;
        end process;

-- Assert Process

        check(Z,'1', 50 ns);
        check(Z,'0', 150 ns);
        check(Z,'1', 250 ns);
        check(Z,'0', 450 ns);

end TESTBENCH;
```

Example 9
Moore Machine – Testbench (cont'd)

```
configuration CFG_TB_MOORE of TB_MOORE is
        for TESTBENCH
                for UUT : MOORE
                end for;
        end for;
end;
```