Ain Shams University
Faculty of Engineering

جامعة عين شمس
كلية الهندسة

CSE215

Electronic Design Automation

Name:

Mohamed Ibrahim El Ghamry

ID:

15P6019

Email:

15P6019@eng.asu.edu.eg

Project:

Vending Machine

Junior

Computer Engineering and Software Systems

Introduction

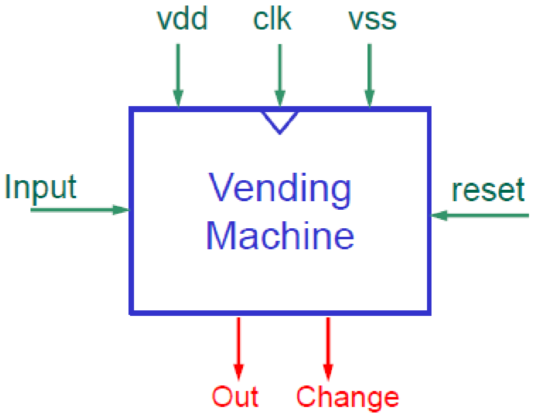
Vending Machine is an automated machine that provides items like drink and juice. The logic of this machine is implemented behaviorally using VHDL code and ModelSim.

Vending Machine Features:

- Sells soft drink and juice.
- The price of the drink is **1.25LE**.
- The machine only accepts 1LE, 0.5LE and 0.25LE.
- The user first enters the money, then selects either a **soft drink** or **juice**.
- The machine returns the change if any.

Details

Input	**(Encoding)**
0.25 LE	000
0.5 LE	001
1.0 LE	010
Soft Drink	011
Juice	100



Output	**(Encoding)**
Nothing	00
Soft Drink	01
Juice	10

Change	**(Encoding)**
No Change	00
0.25 LE	01
0.5 LE	10
0.75 LE	11

States	
S0	Idle
S1	Got 0.25 LE
S2	Got 0.5 LE
S3	Got 0.75 LE
S4	Got 1.0 LE
S5	Got 1.25 LE
S6	Got 1.5 LE
S7	Got 1.75 LE
S8	Got 2.0 LE

FSM Type: Mealy

Transitions: Input/OutputChange

Ex:	Input: 000 == 0.25 LE
000/0000	Output: 00 == Nothing
	Change: 00 == No Change

Results and Snapshots

Finite state machine

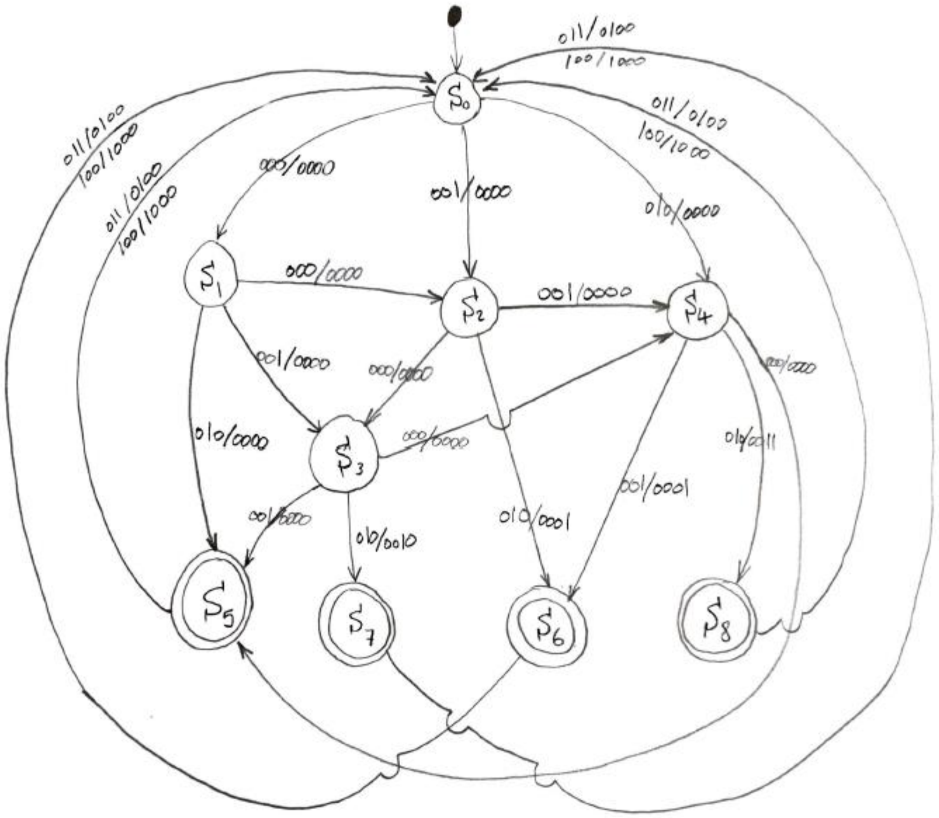


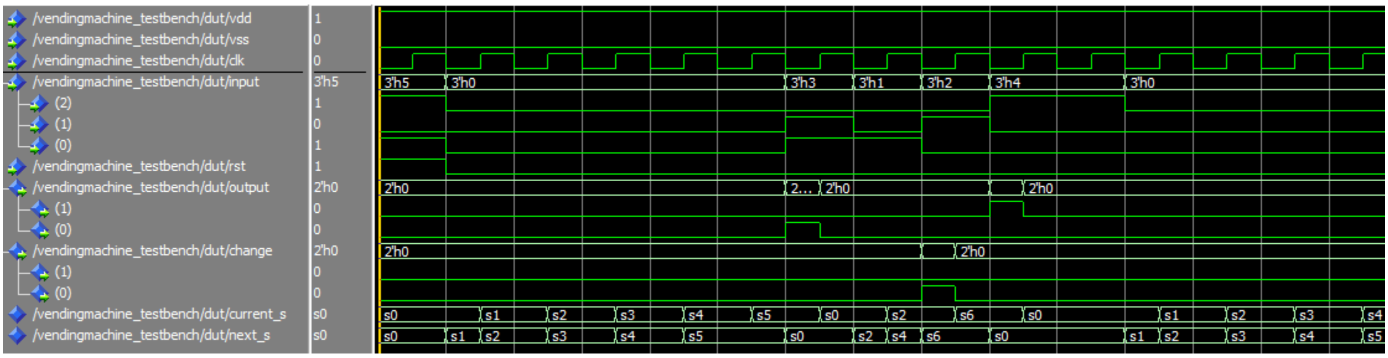
Fig (1): Finite state machine

Test bench test strategy tables

Term	Meaning
input <= "000"; ASSERT change = "00" and output = "00"	Current state: S0 Input: 0.25 LE Output: Nothing Change: No Change Next state: S1
input <= "000"; ASSERT change = "00" and output = "00"	Current state: S1 Input: 0.25 LE Output: Nothing Change: No Change Next state: S2
input <= "000"; ASSERT change = "00" and output = "00"	Current state: S2 Input: 0.25 LE Output: Nothing Change: No Change Next state: S3
input <= "000"; ASSERT change = "00" and output = "00"	Current state: S3 Input: 0.25 LE Output: Nothing Change: No Change Next state: S4
input <= "000"; ASSERT change = "00" and output = "00"	Current state: S4 Input: 0.25 LE Output: Nothing Change: No Change Next state: S5

input <= "011"; ASSERT change = "00" and output = "01"	Current state: S5 Input: Soft Drink Output: Soft Drink Change: No Change Next state: S0
input <= "001"; ASSERT change = "00" and output = "00"	Current state: S0 Input: 0.5 LE Output: Nothing Change: No Change Next state: S2
input <= "010"; ASSERT change = "00" and output = "00"	Current state: S2 Input: 1.0 LE Output: Nothing Change: No Change Next state: S6
input <= "100"; ASSERT change = "01" and output = "10"	Current state: S6 Input: Juice Output: Juice Change: 0.25 LE Next state: S0

ModelSim Simulation



```
/vendingmachine_testbench/dut/vdd           1
/vendingmachine_testbench/dut/vss           0
/vendingmachine_testbench/dut/clk           0
/vendingmachine_testbench/dut/input         3'h5
    (2)                                     1
    (1)                                     0
    (0)                                     1
/vendingmachine_testbench/dut/rst           1
/vendingmachine_testbench/dut/output        2'h0
    (1)                                     0
    (0)                                     0
/vendingmachine_testbench/dut/change        2'h0
    (1)                                     0
    (0)                                     0
/vendingmachine_testbench/dut/current_s     s0
/vendingmachine_testbench/dut/next_s        s0
```

3'h5
3'h0
3'h3
3'h1
3'h2
3'h4
3'h0
2'h0
2...
2'h0
2'h0
2'h0
2'h0
s0 s1 s2 s3 s4 s5 s0 s2 s6 s0 s1 s2 s3 s4
s0 s1 s2 s3 s4 s5 s0 s2 s4 s6 s0 s1 s2 s3 s4 s5

Fig (2): Simulation wave output

Appendix (VHDL Source Codes)

vendingMachine.vhd file

```
entity vendingMachine is
    port(
        vdd: in bit;
        vss: in bit;
        clk: in bit;
        input: in bit_vector(2 downto 0);
        rst: in bit;
        output: out bit_vector(1 downto 0);
        change: out bit_vector(1 downto 0)
    );
end vendingMachine;

architecture beh of vendingMachine is
    type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8);
    signal current_s, next_s: state_type;
    begin
    process(clk, rst)
        begin
            if (rising_edge(clk) and rst = '1') then
                current_s <= s0;
            elsif(rising_edge(clk)) then
                current_s <= next_s;
            end if;
    end process;

    process (current_s, input)
        begin
            case current_s is
                when s0 =>
                    output <= "00";
                    change <= "00";
                    if(input = "000") then
                        next_s <= s1;
                    elsif(input = "001") then
                        next_s <= s2;
                    elsif(input = "010") then
                        next_s <= s4;
                    end if;

                when s1 =>
                    output <= "00";
                    change <= "00";
                    if(input = "000") then
                        next_s <= s2;
                    elsif(input = "001") then
                        next_s <= s3;
                    elsif(input = "010") then
                        next_s <= s5;
                    end if;

                when s2 =>
                    output <= "00";
                    if(input = "000") then
                        next_s <= s3;
                        change <= "00";
                    elsif(input = "001") then
                        next_s <= s4;
                        change <= "00";
                    elsif(input = "010") then
                        next_s <= s6;
            change <= "01";
      end if;

when s3 =>
      output <= "00";
      if(input = "000") then
            next_s <= s4;
            change <= "00";
      elsif(input = "001") then
            next_s <= s5;
            change <= "00";
      elsif(input = "010") then
            next_s <= s7;
            change <= "10";
      end if;

when s4 =>
      output <= "00";
      if(input = "000") then
            next_s <= s5;
            change <= "00";
      elsif(input = "001") then
            next_s <= s6;
            change <= "01";
      elsif(input = "010") then
            next_s <= s8;
            change <= "11";
      end if;

when s5 =>
      change <= "00";
      if(input = "011") then
            next_s <= s0;
            output <= "01";
      elsif(input = "100") then
            next_s <= s0;
            output <= "10";
      end if;

when s6 =>
      change <= "00";
      if(input = "011") then
            next_s <= s0;
            output <= "01";
      elsif(input = "100") then
            next_s <= s0;
            output <= "10";
      end if;

when s7 =>
      change <= "00";
      if(input = "011") then
            next_s <= s0;
            output <= "01";
      elsif(input = "100") then
            next_s <= s0;
            output <= "10";
      end if;

when s8 =>
      change <= "00";
      if(input = "011") then
            next_s <= s0;
            output <= "01";
```

```
                             elsif(input = "100") then
                                   next_s <= s0;
                                   output <= "10";
                             end if;
                  end case;
      end process;
end beh;
```

testBench.vhd file

```
-- Entity declaration for your testbench. Don't declare any ports here
ENTITY vendingMachine_testbench IS
END ENTITY vendingMachine_testbench;

ARCHITECTURE testbench OF vendingMachine_testbench IS

-- Component Declaration for the Device Under Test (DUT)
COMPONENT vendingMachine IS
port(
            vdd: in bit;
            vss: in bit;
            clk: in bit;
            input: in bit_vector(2 downto 0);
            rst: in bit;
            output: out bit_vector(1 downto 0);
            change: out bit_vector(1 downto 0)
      );
END COMPONENT vendingMachine;

FOR dut: vendingMachine USE ENTITY WORK.vendingMachine (beh);

-- Inputs
SIGNAL clk        : bit := '0';
SIGNAL rst        : bit := '1';
SIGNAL vdd        : bit := '1';
SIGNAL vss        : bit := '0';
SIGNAL input      : bit_vector(2 Downto 0) := "101";

-- Outputs
SIGNAL change     : bit_vector(1 Downto 0);
SIGNAL output     : bit_vector(1 Downto 0);

-- Constants and Clock period definitions
constant clk_period : time := 1000 ns;
BEGIN
      dut: vendingMachine PORT MAP (vdd, vss, clk, input, rst, output, change);
      clk_process :process
   begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
   end process;
   stim_proc: PROCESS IS
BEGIN
      WAIT FOR clk_period; --For the output to be stable
      ASSERT change = "00" and output = "00"
      REPORT "Reset error"
      SEVERITY error;

      rst <= '0';

      input <= "000";
      WAIT FOR clk_period;
      ASSERT change = "00" and output = "00"
      REPORT "Outputs error"
      SEVERITY error;

      input <= "000";
      WAIT FOR clk_period;
      ASSERT change = "00" and output = "00"
      REPORT "Outputs error"
      SEVERITY error;

      input <= "000";
      WAIT FOR clk_period;
      ASSERT change = "00" and output = "00"
      REPORT "Outputs error"
      SEVERITY error;

      input <= "000";
      WAIT FOR clk_period;
      ASSERT change = "00" and output = "00"
      REPORT "Outputs error"
      SEVERITY error;

      input <= "000";
      WAIT FOR clk_period;
      ASSERT change = "00" and output = "00"
      REPORT "Outputs error"
      SEVERITY error;

      input <= "011";
      WAIT FOR clk_period;
      ASSERT change = "00" and output = "01"
      REPORT "Outputs error"
      SEVERITY error;

      input <= "001";
      WAIT FOR clk_period;
      ASSERT change = "00" and output = "00"
      REPORT "Outputs error"
      SEVERITY error;

      input <= "010";
      WAIT FOR clk_period;
      ASSERT change = "00" and output = "00"
      REPORT "Outputs error"
      SEVERITY error;

      input <= "100";
      WAIT FOR clk_period;
      ASSERT change = "01" and output = "10"
      REPORT "Outputs error"
      SEVERITY error;
end process;
end;
```