THE UNIVERSITY OF SOUTHERN MISSISSIPPI

COLLEGE OF SCIENCE AND TECHNOLOGY

The Vending Machine

Report from laboratory experiments conducted throughout Spring 2019
as part of CE 340 Digital System Design

Shankar Sharma
W975362

05/01/2019

Abstract:

This report is about small vending machine to dispatch the beverages which will be dispatched only if the user will insert 25c in any possible combination because each beverage in the machine costs the multiple of $0.25. The vending machine only takes input as quarters (25c) and dollars ($1) and it will only return quarters. Each input is a one-clock cycle wide pulse that is synchronous with the clock signal. Inputs will occur “one at a time”. You will not ever have two or more inputs asserted on the same clock cycle. To dispense change, assert each output for one clock cycle. The supply of coins for change and beverages is unlimited. The vend signal should only be asserted for one clock cycle when the user has inserted at least $0.25. This project is done by using VHDL and have been implemented by using FPGA board where switches are used as input money in binary format and LEDs and seven segments are used to show remaining amount/change and dispatched result.

Table of Contents

Abstract: i
Table of Contents ii
1. Introduction and Background 1
2. Design 2
3. Methodology 3
4. States Navigation 7
5. Discussion of Results 11
6. Conclusions 11

1. Introduction and Background

Vending machine is an automatic machine which provides soft drink, snacks etc. to a user when there is inserted money into the machine. There are also available modern vending machines which are dealing with more products and have the flexibility
to use a credit card as input instead of coins. In our project, we have implemented a vending machine for a soft drink. Where inputs are coins of 25c, 10c or 5c
and outputs are a soft drink and remaining amount/change. Before going to furthermore we should know that what finite state machine is?

A. Finite State Machine

Finite state machine (FSM) is actually a mathematical model of computation, this machine can be in one of the states from the total possible states. The present state can be changed according to input from the outside. An FSM can be defined by its states list, initial state and the condition for each state transition. There are two types of FSM

1. Mealy Machine
2. Moore Machine

These both machines have its own pros and cons, which are following:

1. Mealy Machine

In the mealy machine, the output of machine depends upon both the input and present state. The output of the designed machine will be dependent upon the change in input and present state, which definitely decrease the number of states in design. The block diagram of the mealy machine shown below in figure: 1.

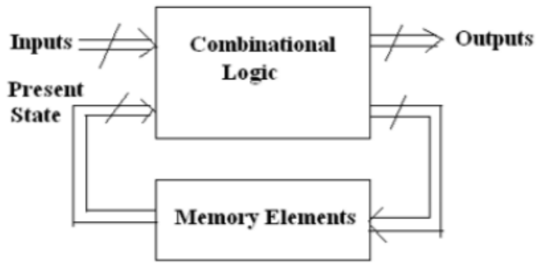


Figure 1: Mealy state machine

2. Moore Machine

In Moore machine, the output of machine depends upon only on the present state. The output of the designed machine will be dependent only upon the change in present state. The output of Moore machine is independent of the change in input, which definitely increases the number of states in design. The block diagram of Moore machine shown below

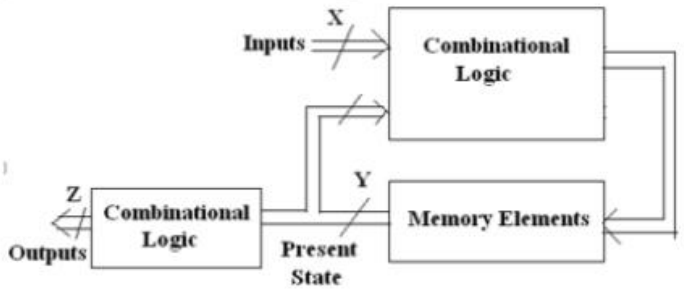
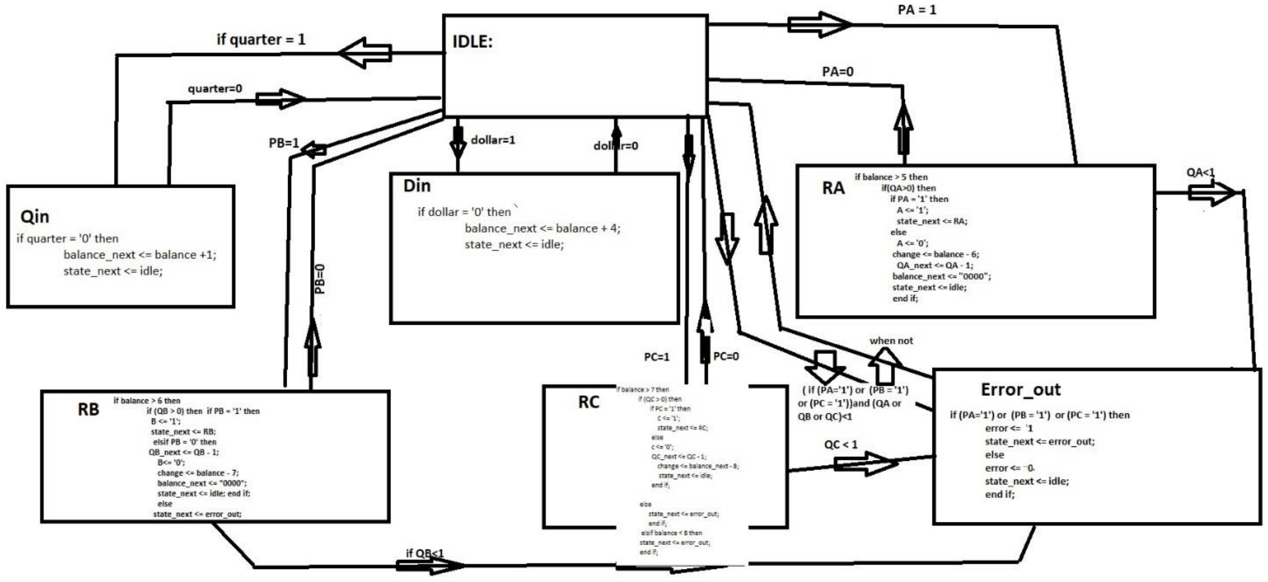


Figure 2: Moore state machine

2. Design

This vending machine takes quarters and dollars as the inputs, where a customer can select from 3 different items. The machine will have three different set of displays, for balance, for change and for messages. At the beginning of the process, the change and message displays will be off, and the balance display will display the balance that is zero at the beginning. When the customer starts inserting quarters or dollars into the machine, the balance display is updated to the most updated balance. When the customer inserts the necessary amount of money into the machine, if he tries to vend something, for example a product A, if he has inserted enough amount to vend A, led A will come up if not the message display will display “E” on the message display. Or if there is no more of the product A in the machine, the message display will display “o” in the display. Or if, the product is dispensed, the necessary change is calculated and then displayed in the change display in terms of quarter. As said previously, this machine will accept quarters and dollars but will only dispense quarters. The machine will use the following state diagram.



```
Qin
if quarter = '0' then
        balance_next <= balance +1;
        state_next <= idle;
```

```
Din
if dollar = '0' then
        balance_next <= balance + 4;
        state_next <= idle;
```

```
RA
if balance > 5 then
        if(QA>0) then
          if PA = '1' then
            A <= '1';
            state_next <= RA;
          else
            A <= '0';
          change <= balance - 6;
            QA_next <= QA - 1;
          balance_next <= "0000";
          state_next <= idle;
          end if;
```

```
RB
if balance > 6 then
        if (QB > 0) then  if PB = '1' then
          B <= '1';
          state_next <= RB;
           elsif PB = '0' then
         QB_next <= QB - 1;
            B<= '0';
            change <= balance - 7;
            balance_next <= "0000";
            state_next <= idle; end if;
            else
          state_next <= error_out;
```

```
RC
if balance > 7 then
        if (QC > 0) then
            if PC = '1' then
              C <= '1';
              state_next <= RC;
            else
            c <= '0';
            QC_next <= QC - 1;
              change <= balance_next - 8;
               state_next <= idle;
            end if;

        else
            state_next <= error_out;
            end if;
        elsif balance < 8 then
        state_next <= error_out;
        end if;
```

```
Error_out
if (PA='1') or  (PB = '1')  or (PC = '1') then
        error <=  '1
        state_next <= error_out;
        else
        error <= '0.
        state_next <= idle;
        end if;
```

3. Methodology

Implementation of the vending machine was a great challenge for us because there were many things which were needed to be integrated; although we have covered all these things in our lectures but there were still such things which were still unsettling us. To implement this machine, we have used state register and Non-clock process.

A. Synchronous Process for State Machine

This state machine is implemented through the process in VHDL and it triggers only whenever the rising edge of the clock occurs and that's why this process is called as synchronous process and it resets the whole vending machine when reset input will be high. At each rising edge of the clock present state of the machine will be changed to next state.

B. Asynchronous Process for Next State Logic

The state diagram mentioned in the figure has to be implemented in this process, in the first state the vending output will be idle and in the same way, messages output is also zero. Now when the state is idle, if the input dollar will be 1, it will change the state to Din. If the input quarter will be 1, it will change the state to Din. More can be seen in the following code:

```
if (qtr = '1') then current_s <= qin; end if;
 if (dollar = '1') then current_s <= din; end if;
 if (cancel = '1') then current_s <= refund; end if;
 if (rstn = '0') then current_s <= rst; end if;
 if (A = '1') then current_s <= reqA; end if;
 if (B = '1') then current_s <= reqB; end if;
 if (C = '1') then current_s <= reqC; end if;
```

C. Switches, LEDs, and Seven-Segment Display

In order to maximize the usefulness of the circuit, is important to make its design user friendly. A series of two switches are used to input the desired amount of coins from the user. Switches 0-1 are used to take an input from the user i.e. the 25c and $1. The switches values are only loaded into the input coin when the rising edge of the clock occurs. The reset button is wired to all process and the non-clock process whenever its value is high it resets everything. Finally, LEDs are illuminated when the vending output will be high or whenever the change will be returned to the user, meanwhile the same will appear on the seven-segment also. LEDs 1, 2 and 3 are used to dispense the products A, B and C respectively. The code I used is given below.

```
library IEEE;
use
IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity display is
   Port ( clk, reset: in
STD_LOGIC;
         hex0 : in
STD_LOGIC_VECTOR (3
downto 0);--balance
         hex1 : in
STD_LOGIC_VECTOR (3
downto 0);--change
         hex2 : in
STD_LOGIC_VECTOR (3
downto 0); -- error
         an : out
STD_LOGIC_VECTOR (7
downto 0);
         sseg : out
STD_LOGIC_VECTOR (7
downto 0));
end display;

architecture Behavioral of
display is
constant N : integer := 19;
signal q_reg, q_next :
unsigned(N-1 downto 0);
signal sel          :
std_logic_vector( 2 downto 0);
signal hex : std_logic_vector( 3
downto 0);

signal hexa :
std_logic_vector(15 downto 0);
signal hexb : std_logic_vector(7
downto 0);
signal hex0000 :
STD_LOGIC_VECTOR (3
downto 0);--balance
signal hex000 :
STD_LOGIC_VECTOR (3
downto 0);--change
signal hex00 :
STD_LOGIC_VECTOR (3
downto 0); -- error
component decoder is
port ( bin_in  : in
std_logic_vector (15 downto 0) ;
         bcd_hun : out
std_logic_vector (3 downto 0) ;
         bcd_ten : out
std_logic_vector (3 downto 0) ;
         bcd_one : out
std_logic_vector (3 downto 0) ) ;
end component;
begin
hexb <= std_logic_vector(5 *
(unsigned(hex0)));
hexa <= std_logic_vector ( 5 *
(unsigned(hexb)));
balance_one : decoder
port map(bin_in =>
std_logic_vector(hexa), bcd_hun
=> hex0000, bcd_ten =>
hex000, bcd_one => hex00);

process(clk, reset)
begin
   if reset = '0' then
      q_reg <= (others => '0');
   elsif( clk'event and clk = '1')
then
   q_reg <= q_next;
   end if;
end process;
q_next <= q_reg + 1;

sel <=
std_logic_vector(q_reg(N-1
downto N-3));
process( sel, hex1, hex2, hex000,
hex00, hex00)
begin
   case sel is
      when "000" =>
         an  <= "11111110";
         hex <= hex00;
         sseg(7) <='1';
      when "001" =>
         an  <= "11111101";
         hex <= hex000;
         sseg(7) <='1';
     when "010" =>
         an  <= "11111011";
         hex <= hex0000;
         sseg(7) <='0';

      when "011" =>
         an  <= "11110111";
         hex <= hex2;
         sseg(7) <='1';
       when "100" =>
         an  <= "11101111";
         hex <= hex2;
         sseg(7) <='1';
      when "101" =>
         an  <= "11111111";
         hex <= "1111";
         sseg(7) <='1';
     when "110" =>
        an  <= "11111111";
        hex <= "1111";
        sseg(7) <='1';
      when "111" =>
        an  <= "01111111";
        hex <= hex1;
        sseg(7) <='1';
      end case;
   end process;

   with hex select
     sseg(6 downto 0) <=
        "1000000" when "0000",
        "1111001" when "0001",
        "0100100" when "0010",
        "0110000" when "0011",
        "0011001" when "0100",
        "0010010" when "0101",
        "0000010" when "0110",
        "1111000" when "0111",
        "0000000" when "1000",
        "0010000" when "1001",
        "0001000" when "1010",
        "0000011" when "1011",
        "1000110" when "1100",
        "0100001" when "1101",
        "0000110" when "1110",
        "1111111" when "1111";

end Behavioral;
```

D. Debouncing Circuit

An important (but often overlooked) concern in digital design is switch bounce. The basic mechanisms involved are related to the mechanical design of the switch and the large electric fields that develops when the contacts are very close but not touching. The result of this situation is arcing between the contacts until they finally settle down and make permanent contact. In many design situations, this arcing (or bouncing as it is more often referred to) is of no concern to the system operation. In other situations, however, this switch bouncing can cause seriously undesirable results. Therefore, it is important that all the switches are connected to the debouncing circuit to make sure that when we make one input, the circuit takes only one input.

```
library IEEE;
use
IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity debounce is
   Port ( clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        sw : in STD_LOGIC;
        db : out STD_LOGIC);
end debounce;
architecture Behavioral of
debounce is
constant N: integer := 20;
type db_state_type is
(zero, wait1_1, wait1_2,
wait1_3, one, wait0_1, wait0_2,
wait0_3);
signal q_reg, q_next: unsigned
(N-1 downto 0);
signal m_tick      : std_logic;
signal state_reg    :
db_state_type;
signal state_next   :
db_state_type;
begin
process(clk,reset)
begin
   if(clk'event and clk = '1') then
      q_reg <= q_next;
   end if;
end process;
q_next <= q_reg + 1;
m_tick <= '1' when q_reg = 0
else '0';
process(clk, reset)
begin
   if(reset = '0') then
      state_reg <= zero;
   elsif(clk'event and clk = '1')
then
      state_reg <= state_next;
   end if;
end process;
process(state_reg, sw, m_tick)
begin
   state_next <= state_reg;
   db <= '0';
   case state_reg is
      when zero =>
         if sw = '1' then
            state_next <=
wait1_1;
         end if;
      when wait1_1 =>
         if sw = '0' then
            state_next <= zero;
         else if m_tick = '1' then
            state_next <=
wait1_2;
         end if;
      end if;
      when wait1_2 =>
         if sw = '0' then
            state_next <= zero;
         else
            if m_tick = '1' then
               state_next <=
wait1_3;
            end if;
         end if;
      when wait1_3 =>
         if sw = '0' then
            state_next <= zero;
         else
            if m_tick = '1' then
               state_next <= one;
            end if;
         end if;
      when one =>
         db <= '1';
         if sw = '0' then
            state_next <=
wait0_1;
         end if;
      when wait0_1 =>
         db <= '1';
         if sw = '1' then
            state_next <= one;
         else
            if m_tick = '1' then
               state_next <=
wait0_2;
            end if;
         end if;
      when wait0_2 =>
         db <= '1';
         if sw = '1' then
            state_next <= one;
         else
            if m_tick = '1' then
               state_next <=
wait0_3;
            end if;
         end if;
      when wait0_3 =>
         db <= '1';
         if sw = '1' then
            state_next <= one;
         else
            if m_tick = '1' then
               state_next <= zero;
            end if;
         end if;
      end case;
   end process;
end Behavioral;
```

E. Decoder Circuit

The inputs are taken as std logic and then transformed to unsigned type. The signals like balance, change and messages are all in unsigned form. To take those unsigned data and push them to the seven-segment display, we had to decode the signals into decimals and take the ones, tens, hundreds place of the signals as separate signals. This was done to push those single signals independently to the seven-segment display.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
entity decoder is
port ( bin_in  : in  std_logic_vector (15 downto
0) ;
        bcd_hun : out std_logic_vector (3 downto
0) ;
        bcd_ten : out std_logic_vector (3 downto
0) ;
        bcd_one : out std_logic_vector (3 downto
0) ) ;
end decoder;

architecture Behavioral of decoder is
signal bin_100: unsigned(15 downto 0);
signal bin_10: unsigned(15 downto 0);
signal bin_1: unsigned(15 downto 0);
signal bcd_1: unsigned(15 downto 0);
signal bcd_2: unsigned(15 downto 0);
signal bcd_3: unsigned(31 downto 0);
begin
bcd_hun <= std_logic_vector(bcd_1(3 downto
0));
bcd_ten <= std_logic_vector(bcd_2(3 downto
0));
bcd_one <= std_logic_vector(bcd_3(3 downto
0));

bin_100 <= unsigned(bin_in);
process(bin_100, bin_10, bin_1)
begin
   bin_10 <= bin_100 mod 100;
   bcd_1 <= unsigned((bin_100 - bin_10) /
100);
   bin_1<= bin_10 mod 10;
   bcd_2 <= unsigned((bin_10 - bin_1)/10);
   bcd_3 <= unsigned(bin_10 - (bcd_2 * 10));
      end process;
end Behavioral;
```

4. States Navigation

To successfully and efficiently sun my design, I used nine states. That are explained below,

a. Reset:
Whenever the reset button(active high) was pushed, it would take the design to the state Rst.

```
process(CLK,RSTn)
   begin
   if(RSTn = '0')         then  current_s <= rst;
   elsif(rising_edge(Clk)) then

      case current_s is
       when rst =>
                    Item_A      <= '0'; Item_B  <= '0'; Item_C  <= '0';
                    change_out     <= "0000";
                    current_s    <= idle;
                    QA <= "0110";
                    QB <= "0110";
                    QC <= "0110";
                    print <= "1111";
                    balance <= "0000";
```

b. Idle:

```
when idle =>
                    Item_A      <= '0'; Item_B  <= '0'; Item_C  <= '0';
                    change_out     <= "1111";
                    balance <= balance;
                    QA <= QA;
                    QB <= QB;
                    QC <= QC;
                    print <= "1111";-- Welcome
                    if (qtr = '1') then current_s <= qin; end if;
                    if (dollar = '1') then current_s <= din; end if;
                    if (cancel = '1') then current_s <= refund; end if;
                    if (rstn = '0') then current_s <= rst; end if;
                    if (A = '1') then current_s <= reqA; end if;
                    if (B = '1') then current_s <= reqB;end if;
                    if (C = '1') then current_s <= reqC; end if;
```

c. Quarter_in (qin):

```
                 when qin =>
                              Item_A      <= '0'; Item_B  <= '0'; Item_C  <= '0';
                              change_out     <= "1111";
                              QA <= QA;
                              QB <= QB;
                              QC <= QC;
                              print <= "1111";
                              if (qtr = '0') then balance <= balance + 1;
                              current_s <= idle;
                              end if;
```

d. Dollar_in(Din):

```
        when din =>
                    Item_A      <= '0'; Item_B  <= '0'; Item_C  <= '0';
                    change_out     <= "1111";
                    QA <= QA;
                    QB <= QB;
                    QC <= QC;
                    print <= "1111";
                    balance <= balance;
                    if (dollar = '0') then balance <= balance + 4;
                    current_s <= idle;
                    end if;
```

e. Product A requested(*reqA*):

```
                   when reqA =>
                        if    (A = '1' and QA >= 1 and balance >= 5) then
                        Item_A      <= '1'; Item_B  <= '0'; Item_C  <= '0';
                        change_out     <= balance - 5;
                        current_s <= reqA;
                        elsif (A = '0' and QA >= 1 and balance >= 5) then
                        Item_A      <= '0'; Item_B  <= '0'; Item_C  <= '0';
                        change_out     <= "1111";
                       change_out     <= balance - 5;
                        current_s    <= idle;
                        QA <= QA -1;
                        QB <= QB;
                        balance <= "0000";
                        QC <= QC;
                        print <= change;
                        else
                        Item_A      <= '0'; Item_B  <= '0'; Item_C  <= '0';
                        change_out     <= "1111"; -- 9$
                        balance <= balance;
                        current_s    <= error;
                        QA <= QA;
                        QB <= QB;
                        QC <= QC;
                        print <= "1110";
                        end if;
```

f. Product B requested *(reqB):*

```
        when reqB =>
            if     (B = '1' and QB >= 1 and balance >= 6) then
            Item_A       <= '0'; Item_B  <= '1'; Item_C  <= '0';
            change_out     <= balance - 6;
            current_s <= reqB;
            elsif (B = '0' and QB >= 1 and balance >= 6) then
            Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '0';
            change_out     <= "1111"; -- 9$
            current_s    <= idle;
            QA <= QA ;
            QB <= QB-1;
            balance <= "0000";
            QC <= QC;
            print <= change;
            else
            Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '0';
            change_out     <= "1111"; -- 9$
            balance <= balance;
            current_s    <= error;
            QA <= QA;
            QB <= QB;
            QC <= QC;
            print <= "1110";
            end if;
```

g. Product C requested *(reqC):*

```
        when reqC =>
            if     (C = '1' and QC >= 1 and balance >= 7) then
            Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '1';
            balance <= balance;
            change_out     <= balance - 7;
            current_s <= reqC;
            elsif (C = '0' and QB >= 1 and balance >= 7) then
            Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '0';
             -- 9$
            current_s    <= idle;
            QA <= QA ;
            QB <= QB;
            balance <= "0000";
            QC <= QC-1;
            print <= change;
            else
            Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '0';
            change_out     <= "1111"; -- 9$
```

```
                    balance <= balance;
                    current_s    <= error;
                    QA <= QA;
                    QB <= QB;
                    QC <= QC;
                    print <= "1110";
                    end if;
```

h. Refund

```
when refund =>
                 if (cancel= '1') then
                 Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '0';
                    change_out     <= balance; -- 9$
                    current_s    <= refund;
                    QA <= QA;
                    QB <= QB;
                    QC <= QC;
                    print <= std_logic_vector(balance);

                else
                   Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '0';
                    change_out     <= "1111";
                    current_s    <= idle;
                    balance <="0000";
                    QA <= QA;
                    QB <= QB;
                    QC <= QC;
                    print <= std_logic_vector(balance);
                    end if;
```

i. Error:

```
when error =>
                if     (B = '1' or A = '1' or C = '1') then
                print <= "1110";
                Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '0';
                balance <= balance;
                change_out <= "1111";
                else
                current_s <= idle;
                print <= "1111";
                Item_A       <= '0'; Item_B  <= '0'; Item_C  <= '0';
                balance <= balance;
                change_out <= "1111";
                end if;
```

5. Discussion of Results

I got hundreds of different kinds of errors throughout the debugging process of the program. However, with the help of different articles and different videos in YouTube, I was able to successfully run the program as advised by Dr. Dawoud.

6. Conclusions

With FSM state machine the task of creating a vending machine was achieved. This was no simple feat, as to include all the combinations within minimum states, proved quite the challenge. Improvements to the design that could be made include the coin sensor used in the input side which sense the right amount and the output will also be the drink and return of the amount will be the coins