Vending Machine

Tiago Piai / ASU ID: 1209830918

EEE 333 - Hardware Design and Programmable Logic

Arizona State University

November 7, 2015

Abstract

For this lab was developed a vending machine. The requirements of the project were that the vending machine receives 25 cents or more and light LEDs meaning that the product and the change are delivered. The coins accepted were nickels, dimes and quarter, if the user put more than a quarter in the machine a change will be delivered by turning on the respective LEDs for nickel and dime. To represent the user inserting coins was used the push buttons.

For this project was necessary to develop two designs, one using Moore Machine and the other using Mealy. After the implementation was possible to determinate which design was the best and most efficient.

1 Introduction

The combinational logic circuits generally are only dependent on the input, in other words the output is a function of the present input only. Sequential logic circuits usually have a memory element, so the output not only depends on the current input, but also depends on the previous inputs.

One example of sequential logic circuit is a Finite State Machine (FSM). It combines combinational and sequential logic to execute determined function. As the name refer this machine has a finite number of states and only one state is executed at a time, depending on the input the state changes. The output could change when the machine is in a state (Moore) or change in the states transitions (Mealy).

The sequential logic is represented by a memory element that stores the value of the present state and changes only when a rising edge happens on the machine clock. The combinational part of the machine is responsible for selecting the next state based on the given input and selecting the proper output based on the current state, for Moore and based on the current state and present input for Mealy.

2 Problem Statement

The objective of this project was to design two vending machines which executes similar functions. One should use the Moore's concepts and the other the Mealy's concepts. The requirements were that the machine must have inputs for nickel, dime and quarter coins and outputs the product and, if necessary, a change. The price specified for the product was 25 cents, so if the user inserts two dimes and one quarter the machine should blink the LED of the product and blink twice the LED of dime, meaning that product was provided and the two dimes as change one at a time.

Other assumptions were that the user will not insert any coins while the machine is dispensing the change and the user will never insert two coins at the same time.

3 Design Explanation

To get a complete understanding of what is needed to implement the vending machine was made a block diagram, where is possible to see all the inputs and outputs of each component of the design. In Figure 1 is present the block diagram.

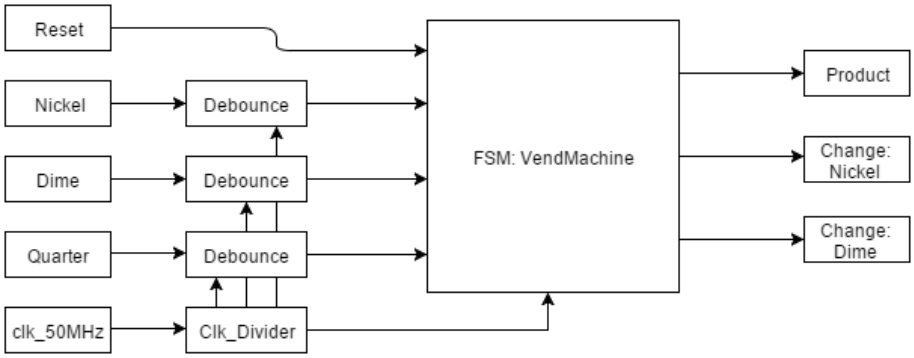

Figure 1: Block Diagram Vending Machine

In each of the coins inputs was used a debounce component, which is necessary when push

buttons are used once those does not work perfectly and every time they are pressed a bounce is generated in the signal provided by them. The debounce is responsible for not letting this bounce interfere in the machine operation generating bugs. It detects that the push button is pressed and sends only a high pulse to output preventing the following bounce to happen.

A clock divider is needed so the LEDs could blink in a speed that the human eyes could see and that the debouncer could generate a greater pulse and not work to fast. If the clock received by the debounce is too fast than the bounce could happen. The clock was slowed from 50MHz to 3Hz.

The block diagram for Moore and Mealy are the same since the only thing that changes is the logic of the VendMachine component.

3.1 First Design: Moore Machine

The first step to design a state machine is to define the states and do a state diagram. For the Moore vending machine was defined 16 states which can be seen in the Table 1 bellow.

Table 1: Moore Vending Machine States

State	Meaning	Output(PDN)
M_0	Initial State – 0 cents	000
M_5	5 cents inserted	000
M_10	10 cents inserted	000
M_15	15 cents inserted	000
M_20	20 cents inserted	000
M_25	25 cents inserted	100
M_30	30 cents inserted	100
M_35	35 cents inserted	100
M_40	40 cents inserted	100
M_45	45 cents inserted	100
C1_D	Plus one dime change	010
C1_N	Plus one nickel change	001
LED1_OFF	All LEDs off	000
C2_D	One dime change	010
C1_N	One nickel change	001
LED2_OFF	All LEDs off	000

The outputs are the Product, Dime change and Nickel change respectively. This is states relations are shown in the state diagram bellow:

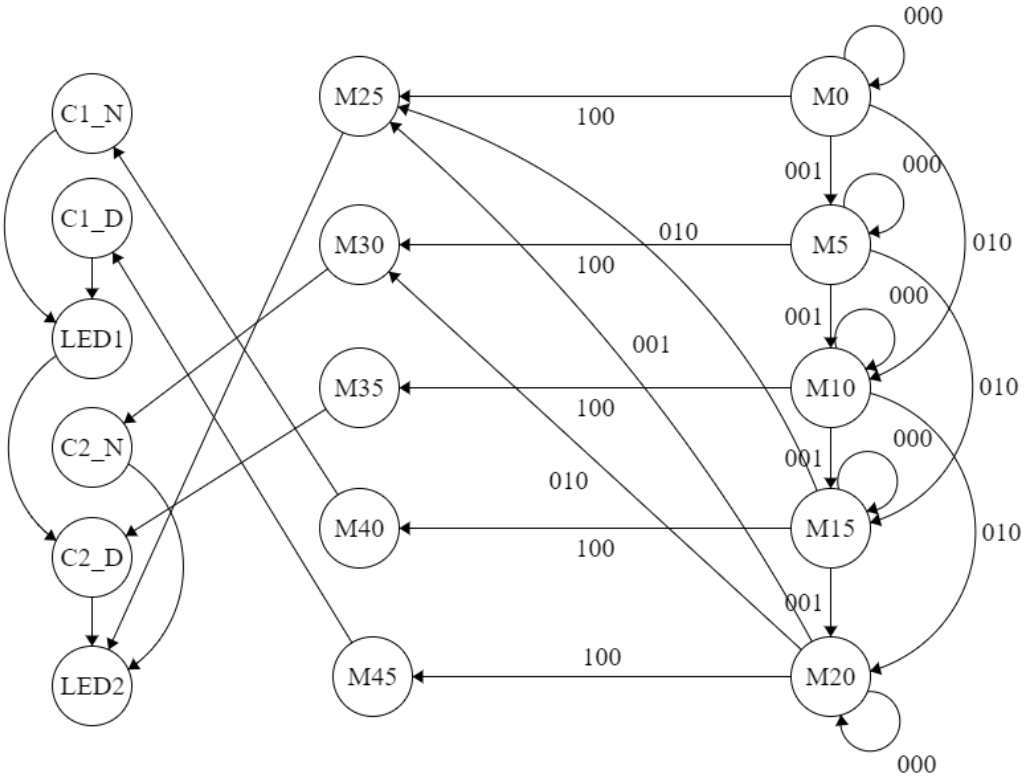

Figure 2: State Diagram - Moore Vending Machine

The number in each arrow represent the input received. When the LED2_OFF state is executed the machine resets and returns to M_0. If the reset switch is activated in any state the it will directly return to M_0. The arrows that does not have inputs on it does not need an specific input it just go to the next state.

The VHDL design of this Moore consisted in the creation of three different process: one sequential and two combinational. The sequential process was responsible for storing the current state until a rising edge on the clock happens and the current state receives the next state value. One of the combinational process was responsible to check which is the current state and then check for the input value to determinate to which state it will go or if it will stay in the same state. The second process checks for the current state and sets the proper output.

This selections are made with multiplexers, for example the combinational process responsible for the value of the next state is mutiplexer and the selector is the input value plus the value of the current state. The sequential process is a flip-flop D that stores the current state value.

3.2 Second Design: Mealy Machine

For the Mealy machine the same steps were used to the development, except that this time the outputs are changed in the state transition. The Table 2 bellow describe each of the states:

Table 2: Mealy Machine States

State	Meaning
M_0	Initial State – 0 cents
M_5	5 cents inserted
M_10	10 cents inserted
M_15	15 cents inserted
M_20	20 cents inserted
C1_D	Plus one dime change
C1_N	Plus one nickel change
LED1_OFF	All LEDs off
C2_D	One dime change
C1_N	One nickel change
LED2_OFF	All LEDs off

Using this states was possible to draw the state diagram present in the Figure 3:

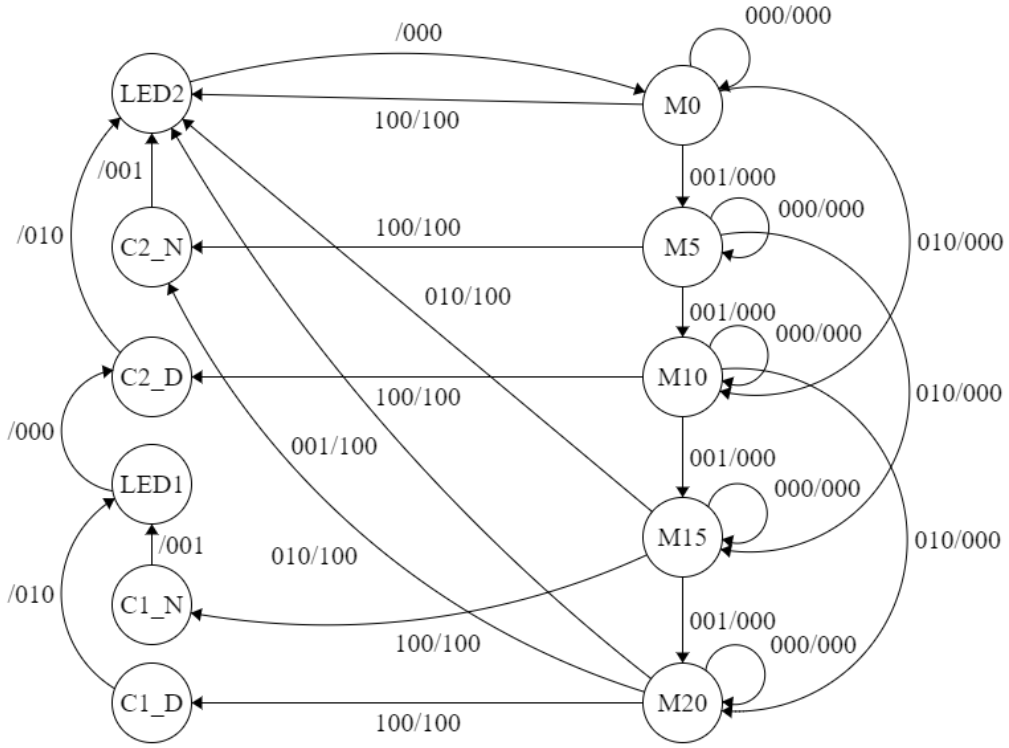

Figure 3: State Diagram - Mealy Vending Machine

The numbers in the arrows represents the inputs and outputs in every transition. The first three numbers before the slash are the Quarter, Dime and Nickel inputs respectively and the three numbers after the slash are the Product, Dime change and the Nickel change.

The implementation of this machine in VHDL required three process: one sequential and two combinational, just like the Moore machine. The sequential process was a memory element responsible for storing the current state and when happens a rising edge update the stage. The first combinational process had function of selecting which will be the next state based on the current state and the input value. This two process are the same as the ones used in the Moore machine, the difference is in the second combinational process for the output. This process checks for the current state and the values in the inputs to then change the output values.

4 Results

The debounce was written and a testbenched was typed to check if it works, the Figure 4 shows the results.

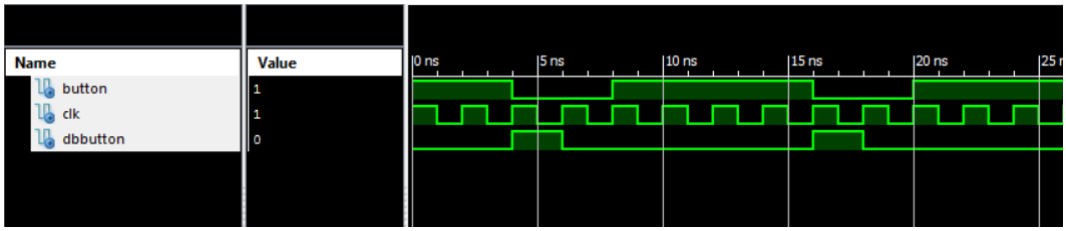

Figure 4: Debounce Testbench

The 'button' and 'clk' are inputs and the 'dbbutton' is the output.

4.1 Moore Machine

For the Moore Machine testbench were executed four different operations:

- Nickel + Dime + Dime = 25 cents
- Dime + Dime + Dime = 30 cents
- Nickel + Dime + Quarter = 40 cents
- Dime + Dime + Quarter = 45 cents

The result can be seen in the Figure 5 bellow:

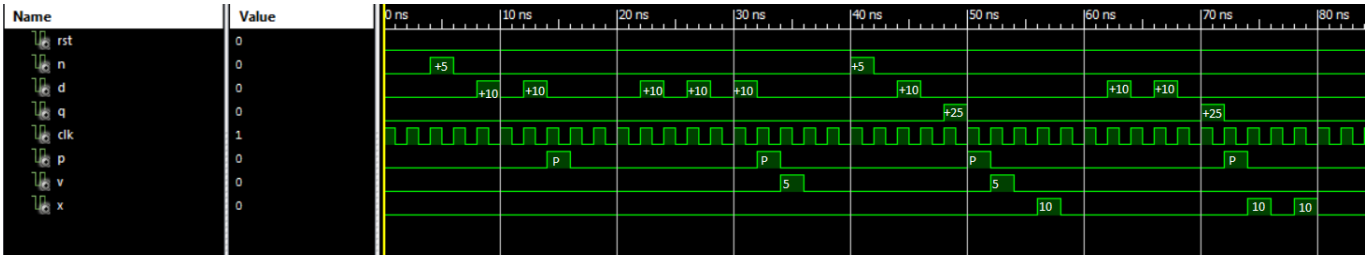

Figure 5: Moore Vending Machine Testbench

4.2 Mealy Machine

The previous four operations were also executed in the Mealy machine. The Figure 6 shows the results:

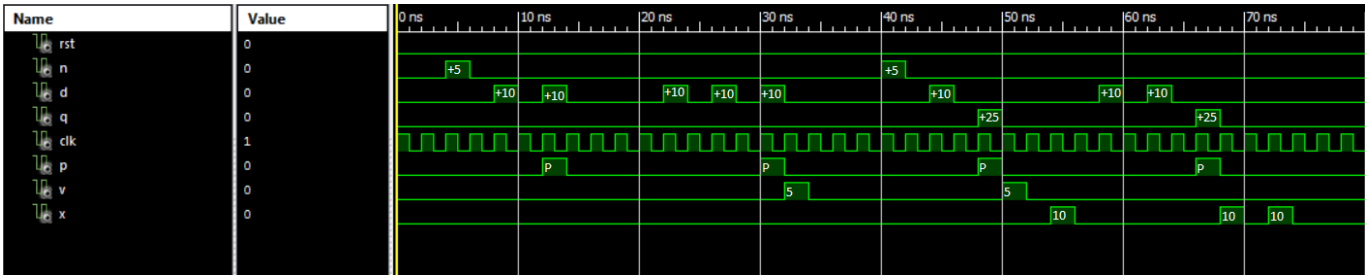

Figure 6: Mealy Vending Machine Testbench

4.3 Best Design Selection

Between the designs of the Moore and Mealy machine, the Mealy is preferred once it has fewer states and consequently reduces the size of combinational logic elements. The Mealy is more efficient than Moore in this case, however the Mealy notation can be more abstract and harder to understand.

For this specific case not only the Mealy need less hardware, but it also present quicker outputs.

5 Conclusions

In this lab was implemented in VHDL two vending machines: one using Moore logic and other using Mealy logic. Is possible to see through this project that the implementation of finite state machines are much faster in VHDL than in direct hardware. This project provided a better understanding of FSM and the differences between the Mealy and Moore designs. It is possible to conclude that this project was important to the student's educational growth.

6 Codes and Test Benches

6.1 Moore Vending Machine

6.1.1 VendingMachine_Moore.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY VendingMachine_Moore IS
        PORT(       reset, Nickel, Dime, Quarter, clk_50MHz: IN STD_LOGIC;
                       V_change, X_change, Product: OUT STD_LOGIC;
                       ledClk, ledCoin: OUT STD_LOGIC);
END VendingMachine_Moore;

ARCHITECTURE func OF VendingMachine_Moore IS

COMPONENT vendMachine IS
        PORT(       rst, N, D, Q, clk: IN STD_LOGIC;
                       P, V, X: OUT STD_LOGIC);
END COMPONENT;

COMPONENT Debounce IS
PORT(       button, clk: IN STD_LOGIC;
               dbButton: OUT STD_LOGIC);
END COMPONENT;

COMPONENT Clk_Divider IS
PORT (rst, clk_50MHz: IN STD_LOGIC;
               clk_out: OUT STD_LOGIC);
END COMPONENT;


SIGNAL dbNickel, dbDime, dbQuarter: STD_LOGIC;

SIGNAL P, V, X: STD_LOGIC;
SIGNAL clk_vMach: STD_LOGIC;

BEGIN

ledClk <= clk_vMach;
```

```
ledCoin <= dbNickel OR dbDime OR dbQuarter;

U1: vendMachine PORT MAP (reset, dbNickel, dbDime, dbQuarter, clk_vMach, Product, V_cha
D1: Debounce PORT MAP (Nickel, clk_vMach, dbNickel);
D2: Debounce PORT MAP (Dime, clk_vMach, dbDime);
D3: Debounce PORT MAP (Quarter, clk_vMach, dbQuarter);
C1: Clk_Divider PORT MAP (reset, clk_50MHz, clk_vMach);
END func;
```

6.1.2 vendMachine.vhdl

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;
USE ieee.std_logic_arith.all;

ENTITY vendMachine IS
        PORT(        rst, N, D, Q, clk: IN STD_LOGIC;
                        P, V, X: OUT STD_LOGIC);
END vendMachine;

ARCHITECTURE func OF vendMachine IS
TYPE STATE_TYPE IS (M_0, M_5, M_10, M_15, M_20, M_25, M_30, M_35, M_40,
                    M_45, C1_D, C1_N, led1_off, C2_D, C2_N, led2_off);

SIGNAL current_state, next_state: STATE_TYPE;
BEGIN

state_uptade:
PROCESS(clk, rst)
BEGIN
```

```
        IF (rst = '1') THEN
                current_state <= M_0;
        ELSIF (clk'EVENT AND clk = '1') THEN
       current_state <= next_state;
    END IF;
END PROCESS;

state_change:
PROCESS (N, D, Q, current_state)
BEGIN
        CASE current_state IS
                WHEN M_0 =>
                        IF (N = '1' AND D = '0' AND Q = '0') THEN
                                next_state <= M_5;
                        ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                                next_state <= M_10;
                        ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                                next_state <= M_25;
                        ELSE
                                next_state <= M_0;
                        END IF;

                WHEN M_5 =>
                        IF (N = '1' AND D = '0' AND Q = '0') THEN
                                next_state <= M_10;
                        ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                                next_state <= M_15;
                        ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                                next_state <= M_30;
                        ELSE
                                next_state <= M_5;
                        END IF;
```

```
WHEN M_10 =>
        IF (N = '1' AND D = '0' AND Q = '0') THEN
                next_state <= M_15;
        ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                next_state <= M_20;
        ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                next_state <= M_35;
        ELSE
                next_state <= M_10;
        END IF;

WHEN M_15 =>
        IF (N = '1' AND D = '0' AND Q = '0') THEN
                next_state <= M_20;
        ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                next_state <= M_25;
        ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                next_state <= M_40;
        ELSE
                next_state <= M_15;
        END IF;

WHEN M_20 =>
        IF (N = '1' AND D = '0' AND Q = '0') THEN
                next_state <= M_25;
        ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                next_state <= M_30;
        ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                next_state <= M_45;
        ELSE
                next_state <= M_20;
```

```
            END IF;

        WHEN M_25 =>
            next_state <= led2_off;

        WHEN M_30 =>
            next_state <= C2_N;

        WHEN M_35 =>
            next_state <= C2_D;

        WHEN M_40 =>
            next_state <= C1_N;

        WHEN M_45 =>
            next_state <= C1_D;

        WHEN C1_D =>
            next_state <= led1_off;

        WHEN C1_N =>
            next_state <= led1_off;

        WHEN led1_off =>
            next_state <= C2_D;

        WHEN C2_D =>
            next_state <= led2_off;

        WHEN C2_N =>
            next_state <= led2_off;
```

```
                WHEN led2_off =>
                        next_state <= M_0;
        END CASE;
END PROCESS;

output_process:
PROCESS (current_state)

BEGIN
        CASE current_state IS
                WHEN M_0 =>
                        P <= '0';
                        V <= '0';
                        X <= '0';

                WHEN M_5 =>
                        P <= '0';
                        V <= '0';
                        X <= '0';

                WHEN M_10 =>
                        P <= '0';
                        V <= '0';
                        X <= '0';

                WHEN M_15 =>
                        P <= '0';
                        V <= '0';
                        X <= '0';

                WHEN M_20 =>
                        P <= '0';
```

```
            V <= '0';
            X <= '0';

        WHEN M_25 =>
            P <= '1';
            V <= '0';
            X <= '0';

        WHEN M_30 =>
            P <= '1';
            V <= '0';
            X <= '0';

        WHEN M_35 =>
            P <= '1';
            V <= '0';
            X <= '0';

        WHEN M_40 =>
            P <= '1';
            V <= '0';
            X <= '0';

        WHEN M_45 =>
            P <= '1';
            V <= '0';
            X <= '0';

        WHEN C1_N =>
            V <= '1';
            P <= '0';
```

```
                WHEN C1_D =>
                        X <= '1';
                        P <= '0';

                WHEN led1_off =>
                        V <= '0';
                        X <= '0';
                        P <= '0';

                WHEN C2_N =>
                        V <= '1';
                        P <= '0';

                WHEN C2_D =>
                        X <= '1';
                        P <= '0';

                WHEN led2_off =>
                        V <= '0';
                        X <= '0';
                        P <= '0';


        END CASE;
END PROCESS;


END func;
```

6.1.3 Debounce.vhdl

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
```

```
ENTITY Debounce IS
PORT(        button, clk: IN STD_LOGIC;
                dbButton: OUT STD_LOGIC);
END Debounce;

ARCHITECTURE func OF Debounce IS
SIGNAL count: STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
PROCESS(clk, button, count)
BEGIN
        IF (button = '1') THEN
                count <= "00";
        ELSIF (clk'EVENT AND clk = '1') THEN
                IF(count /= "11") THEN
                        count <= count + 1;
                END IF;
        END IF;
        IF (count = "01" AND button = '0') THEN
                dbButton <= '1';
        ELSE
                dbButton <= '0';
        END IF;
END PROCESS;
END func;
```

6.1.4 Clk_divider.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY Clk_Divider IS
```

```
PORT (rst, clk_50MHz: IN STD_LOGIC;
                clk_out: OUT STD_LOGIC);
END Clk_Divider;

ARCHITECTURE func OF Clk_Divider IS
BEGIN
clk_div:
PROCESS (clk_50MHz, rst)
VARIABLE count: STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN
        IF (rst = '1') THEN
                count := x"00000000";
        ELSIF (clk_50MHz'EVENT AND clk_50MHz = '1') THEN
                        count := count + 1;
        END IF;
        clk_out <= count(23); --3Hz
END PROCESS;

END func;
```

6.1.5 VendingMachine_Moore.ucf

```
## 50 MHz clock
NET "clk_50MHz"   LOC = "B8" ;

## Inputs
NET "reset"       LOC = "P11";
NET "Nickel"      LOC = "A7";
NET "Dime"        LOC = "M4";
NET "Quarter"     LOC = "C11";
```

```
## Outputs
NET "Product"     LOC = "P7";
NET "X_change"    LOC = "M11";
NET "V_change"    LOC = "M5";

NET "ledClk"      LOC = "G1";
NET "ledCoin"     LOC = "P4";
```

6.2 Mealy Vending Machine

6.2.1 VendindMachine_Mealy.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY VendingMachine_Mealy IS
        PORT (reset, Nickel, Dime, Quarter, clk_50MHz: IN STD_LOGIC;
                        V_change, X_change, Product: OUT STD_LOGIC;
                        ledClk, ledCoin: OUT STD_LOGIC);
END VendingMachine_Mealy;

ARCHITECTURE func OF VendingMachine_Mealy IS

COMPONENT vendMachine IS
        PORT(        rst, N, D, Q, clk: IN STD_LOGIC;
                        P, V, X: OUT STD_LOGIC);
END COMPONENT;

COMPONENT Debounce IS
PORT(        button, clk: IN STD_LOGIC;
                dbButton: OUT STD_LOGIC);
END COMPONENT;
```

```
COMPONENT Clk_Divider IS
PORT (rst, clk_50MHz: IN STD_LOGIC;
                clk_out: OUT STD_LOGIC);
END COMPONENT;

SIGNAL dbNickel, dbDime, dbQuarter: STD_LOGIC;

SIGNAL clk_vMach: STD_LOGIC;

BEGIN

ledClk <= clk_vMach;
ledCoin <= dbNickel OR dbDime OR dbQuarter;

U1: vendMachine PORT MAP (reset, dbNickel, dbDime, dbQuarter, clk_vMach, Product, V_cha
D1: Debounce PORT MAP (Nickel, clk_vMach, dbNickel);
D2: Debounce PORT MAP (Dime, clk_vMach, dbDime);
D3: Debounce PORT MAP (Quarter, clk_vMach, dbQuarter);
C1: Clk_Divider PORT MAP (reset, clk_50MHz, clk_vMach);

END func;
```

6.2.2 vendMachine.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;
USE ieee.std_logic_arith.all;

ENTITY vendMachine IS
        PORT(        rst, N, D, Q, clk: IN STD_LOGIC;
```

```
                        P, V, X: OUT STD_LOGIC);
END vendMachine;

ARCHITECTURE func OF vendMachine IS
TYPE STATE_TYPE IS (M_0, M_5, M_10, M_15, M_20, C1_D, C1_N, led1_off, C2_D, C2_N, led2_

SIGNAL current_state, next_state: STATE_TYPE;
BEGIN

state_uptade:
PROCESS(clk, rst)
BEGIN
        IF (rst = '1') THEN
                current_state <= M_0;
        ELSIF (clk'EVENT AND clk = '1') THEN
       current_state <= next_state;
    END IF;
END PROCESS;

state_change:
PROCESS (N, D, Q, current_state)
BEGIN
        CASE current_state IS
                WHEN M_0 =>
                        IF (N = '1' AND D = '0' AND Q = '0') THEN
                                next_state <= M_5;
                        ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                                next_state <= M_10;
                        ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                                next_state <= led2_off;
                        ELSE
                                next_state <= M_0;
```

```
            END IF;

        WHEN M_5 =>
            IF (N = '1' AND D = '0' AND Q = '0') THEN
                next_state <= M_10;
            ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                next_state <= M_15;
            ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                next_state <= C2_N;
            ELSE
                next_state <= M_5;
            END IF;

        WHEN M_10 =>
            IF (N = '1' AND D = '0' AND Q = '0') THEN
                next_state <= M_15;
            ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                next_state <= M_20;
            ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                next_state <= C2_D;
            ELSE
                next_state <= M_10;
            END IF;

        WHEN M_15 =>
            IF (N = '1' AND D = '0' AND Q = '0') THEN
                next_state <= M_20;
            ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                next_state <= led2_off;
            ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                next_state <= C1_N;
            ELSE
```

```
                next_state <= M_15;
        END IF;

WHEN M_20 =>
        IF (N = '1' AND D = '0' AND Q = '0') THEN
                next_state <= led2_off;
        ELSIF (N = '0' AND D = '1' AND Q = '0') THEN
                next_state <= C2_N;
        ELSIF (N = '0' AND D = '0' AND Q = '1') THEN
                next_state <= C1_D;
        ELSE
                next_state <= M_20;
        END IF;

WHEN C1_D =>
        next_state <= led1_off;

WHEN C1_N =>
        next_state <= led1_off;

WHEN led1_off =>
        next_state <= C2_D;

WHEN C2_D =>
        next_state <= led2_off;

WHEN C2_N =>
        next_state <= led2_off;

WHEN led2_off =>
        next_state <= M_0;
```

```
        END CASE;
END PROCESS;

output_process:
PROCESS (current_state, N, D, Q, rst)
BEGIN
        IF (rst <= '1') THEN
                P <= '0';
                V <= '0';
                X <= '0';
        END IF;
        IF (current_state = M_0 AND Q = '1') THEN
                P <= '1';
                V <= '0';
                X <= '0';
        ELSIF (current_state = M_5 AND Q = '1') THEN
                P <= '1';
                V <= '0';
                X <= '0';
        ELSIF (current_state = M_10 AND Q = '1') THEN
                P <= '1';
                V <= '0';
                X <= '0';
        ELSIF (current_state = M_15 AND (D = '1' OR Q = '1')) THEN
                P <= '1';
                V <= '0';
                X <= '0';
        ELSIF (current_state = M_20 AND (N = '1' OR D = '1' OR Q = '1')) THEN
                P <= '1';
                V <= '0';
                X <= '0';
        ELSIF (current_state = C1_D) THEN
```

```
                P <= '0';
                V <= '0';
                X <= '1';
        ELSIF (current_state = C1_N) THEN
                P <= '0';
                V <= '1';
                X <= '0';
        ELSIF (current_state = led1_off) THEN
                P <= '0';
                V <= '0';
                X <= '0';
        ELSIF (current_state = C2_D) THEN
                P <= '0';
                V <= '0';
                X <= '1';
        ELSIF (current_state = C2_N) THEN
                P <= '0';
                V <= '1';
                X <= '0';
        ELSIF (current_state = led2_off) THEN
                P <= '0';
                V <= '0';
                X <= '0';
        END IF;
END PROCESS;
END func;
```

6.2.3 Debounce.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
```

```
ENTITY Debounce IS
PORT(        button, clk: IN STD_LOGIC;
                dbButton: OUT STD_LOGIC);
END Debounce;

ARCHITECTURE func OF Debounce IS
SIGNAL count: STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
PROCESS(clk, button, count)
BEGIN
        IF (button = '1') THEN
                count <= "00";
        ELSIF (clk'EVENT AND clk = '1') THEN
                IF(count /= "11") THEN
                        count <= count + 1;
                END IF;
        END IF;
        IF (count = "01" AND button = '0') THEN
                dbButton <= '1';
        ELSE
                dbButton <= '0';
        END IF;
END PROCESS;
END func;
```

6.2.4 Clk_divider,vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY Clk_Divider IS
```

```
PORT (rst, clk_50MHz: IN STD_LOGIC;
                clk_out: OUT STD_LOGIC);
END Clk_Divider;

ARCHITECTURE func OF Clk_Divider IS
BEGIN
clk_div:
PROCESS (clk_50MHz, rst)
VARIABLE count: STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN
        IF (rst = '1') THEN
                count := x"00000000";
        ELSIF (clk_50MHz'EVENT AND clk_50MHz = '1') THEN
                        count := count + 1;
        END IF;
        clk_out <= count(23); --3Hz
END PROCESS;

END func;
```

6.2.5 VendingMachine_Mealy.ucf

```
## 50 MHz clock
NET "clk_50MHz"   LOC = "B8" ;

## Inputs
NET "reset"       LOC = "P11";
NET "Nickel"      LOC = "A7";
NET "Dime"        LOC = "M4";
NET "Quarter"     LOC = "C11";
```

```
## Outputs
NET "Product"     LOC = "P7";
NET "X_change"    LOC = "M11";
NET "V_change"    LOC = "M5";

NET "ledClk"      LOC = "G1";
NET "ledCoin"     LOC = "P4";
```

6.3 Mealy/Moore Test Bench

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

ENTITY vendMachine_tb IS
END vendMachine_tb;

ARCHITECTURE behavior OF vendMachine_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT vendMachine
    PORT(
         rst : IN  std_logic;
         N : IN  std_logic;
         D : IN  std_logic;
         Q : IN  std_logic;
         clk : IN  std_logic;
         P : OUT  std_logic;
         V : OUT  std_logic;
         X : OUT  std_logic
        );
    END COMPONENT;
```

```
   --Inputs
   signal rst : std_logic := '0';
   signal N : std_logic := '0';
   signal D : std_logic := '0';
   signal Q : std_logic := '0';
   signal clk : std_logic := '0';

         --Outputs
   signal P : std_logic;
   signal V : std_logic;
   signal X : std_logic;


BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: vendMachine PORT MAP (
          rst => rst,
          N => N,
          D => D,
          Q => Q,
          clk => clk,
          P => P,
          V => V,
          X => X
        );

         clock:
        PROCESS
        BEGIN
```

```
                clk <= NOT clk;
                WAIT FOR 1 NS;
        END PROCESS;

        PROCESS
        BEGIN
                D <= '0';
                N <= '0';
                Q <= '0';

                WAIT FOR 4 NS;

-- N + D + D = 25 cents no change
                N <= '1';

                WAIT FOR 2 NS;

                N <= '0';

                WAIT FOR 2 NS;

                D <= '1';

                WAIT FOR 2 NS;

                D <= '0';

                WAIT FOR 2 NS;

                D <= '1';

                WAIT FOR 2 NS;
```

```
                D <= '0';

                WAIT FOR 8 NS;

-- D + D + D = 30 cents 5 cents of change

                D <= '1';

                WAIT FOR 2 NS;

                D <= '0';

                WAIT FOR 2 NS;

                D <= '1';

                WAIT FOR 2 NS;

                D <= '0';

                WAIT FOR 2 NS;

                D <= '1';

                WAIT FOR 2 NS;

                D <= '0';

                WAIT FOR 8 NS;

-- N + D + Q = 40 cents 15 cents of change
```

```
                N <= '1';

                WAIT FOR 2 NS;

                N <= '0';

                WAIT FOR 2 NS;

                D <= '1';

                WAIT FOR 2 NS;

                D <= '0';

                WAIT FOR 2 NS;

                Q <= '1';

                WAIT FOR 2 NS;

                Q <= '0';

                WAIT FOR 8 NS;

-- D + D + Q = 45 cents 20 cents of change

                D <= '1';

                WAIT FOR 2 NS;

                D <= '0';
```

```
                WAIT FOR 2 NS;

                D <= '1';

                WAIT FOR 2 NS;

                D <= '0';

                WAIT FOR 2 NS;

                Q <= '1';

                WAIT FOR 2 NS;

                Q <= '0';

                WAIT;
        END PROCESS;
END;
```

6.4 Extra: Debounce Testbench

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

ENTITY Debounce_tb IS
END Debounce_tb;

ARCHITECTURE behavior OF Debounce_tb IS

    -- Component Declaration for the Unit Under Test (UUT)
```

```
    COMPONENT Debounce
    PORT(
         button : IN  std_logic;
         clk : IN  std_logic;
         dbButton : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal button : std_logic := '0';
   signal clk : std_logic := '0';

         --Outputs
   signal dbButton : std_logic;


BEGIN

         -- Instantiate the Unit Under Test (UUT)
   uut: Debounce PORT MAP (
          button => button,
          clk => clk,
          dbButton => dbButton
        );

        PROCESS
        BEGIN
        clk <= NOT clk;
        WAIT FOR 1 NS;
        END PROCESS;
```

```
        PROCESS
        BEGIN
                button <= '1';

                WAIT FOR 4 NS;

                button <= '0';

                WAIT FOR 4 NS;

                button <= '1';

                WAIT FOR 8 NS;

                button <= '0';

                WAIT FOR 4 NS;

                button <= '1';

                WAIT;
        END PROCESS;

END;
```