Vending Machine

DLD_4301

Final project

BASU_UNIVERSITY

Author : Mehrdad Varmaziyar

Content :

Introduction

In this project we have designed a Vending Machine using VHDL .

In the following we will discuss state machine and digital design of this hard were and then we will discuss the code and test cases and at the final part you can see the user guide of the project and we have explained components and main code to help you understand it comprehensively .

Design

In this part we will check and analyze the design of circuit using FSM and then functionality of module will be explained .

Finite state machine

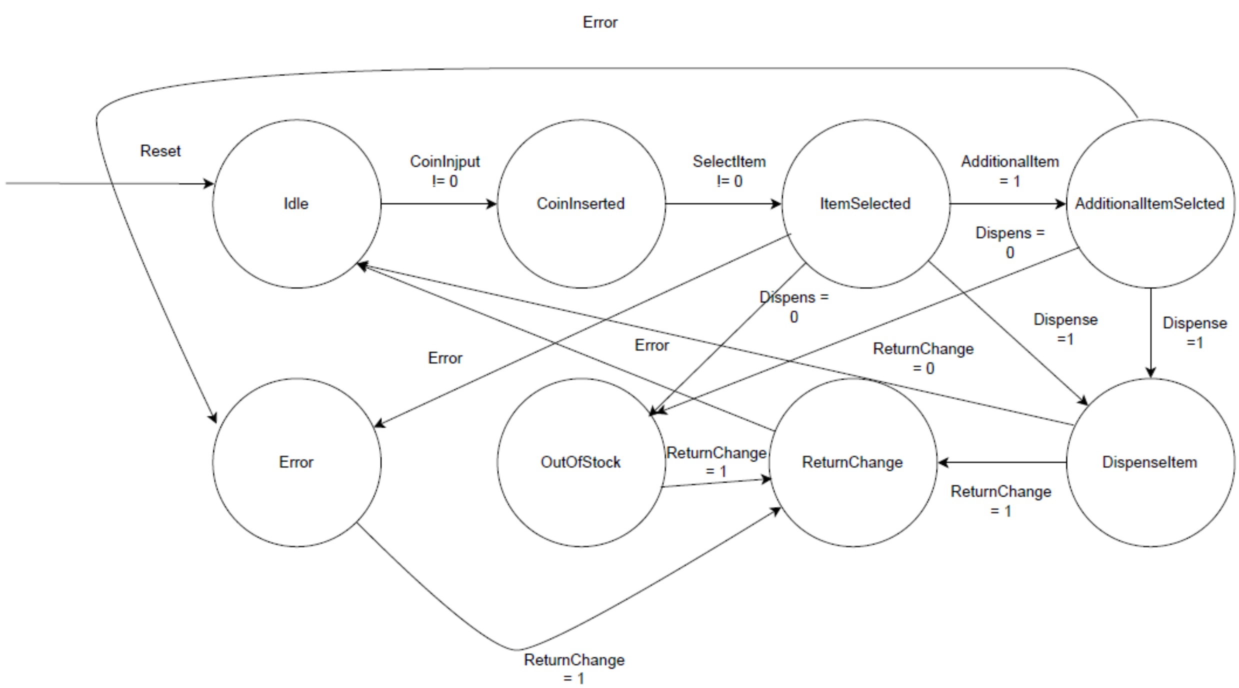
First of all we should define states for our Vending machine then we will connect states to each other based on different inputs and signals.

STATES

1. **Idle**: waiting for coin input.
2. **CoinInserted**: A coin has been inserted,
3. **ItemSelected**: An item has been selected
4. **AdditionalItemSelected**: An additional item has been selected.
5. **DispenseItem**: Dispensing the selected item.
6. **ReturnChange**: Returning change if necessary.
7. **OutOfStock**: An item is out of stock.
8. **Error**: Not enough balance to select an item.

SIGNALS

1.CoinInput : 1,5,10
2.SelectItem : A,B,C
3.AdditionalItem : 0 , 1
4.Dispense : 0,1
5.ReturnChange : 0 ,1
6.Error : 1,0
7.Reset : 0,1



Module Functionality

Our first state is idle and it means the Machine is expecting a coin (signal) to go to our next state Coin Inserted then after inserting coin the machine expect a signal for selecting item and if an item is selected it will go to next state but here we face a dilemma choosing an extra item or dispensing the item and also another dilemma error or out of stock and they will happen based on the signal that you can see on their edges and if we go to out of stock or error or dispense Item we will go to return change and and after that we will get back to idle .

Reset is connected to idle so we somehow reset the system by getting back to Idle.

Testing report

All actions are obvious if you take a look at the comments so I prefer not to explain them again one by one but as a summery when we enter a coin our balance will be increased and then we select item now we have several scenarios consist of error for not having enough balance or out of stock for not having an item or user may choose an extra item and each of this scenarios has their own signal and pass you can see step by step explanation on comments of the code .

```
stim_proc: process
begin
    -- Hold reset state for a while
    reset <= '1';
    wait for 20 ns;
    reset <= '0';

    -- Scenario 1: Insert coins and buy item A (price 5)
    coin_input <= "01";  -- Insert 1 unit
    wait for clk_period;
    coin_input <= "10";  -- Insert 5 units
    wait for clk_period;

    -- Request item A (available)
    item_available <= '1';
    item_selection <= "00";  -- Select item A
    wait for clk_period;

    -- Check if item A was dispensed
    wait for clk_period;  -- Wait for dispensing

    -- Scenario 2: Not enough balance for item B (price 7)
    new_order_request <= '1';  -- Request new order
    wait for clk_period;
    new_order_request <= '0';   -- Reset new order request

    coin_input <= "00";  -- Insert 1 unit (balance = 1)
    wait for clk_period;
    item_selection <= "01";  -- Select item B
    wait for clk_period;  -- Should trigger error signal

    -- Scenario 3: Insert enough coins and buy item B
    coin_input <= "01";  -- Insert 1 unit (balance = 2)
    wait for clk_period;
    coin_input <= "10";  -- Insert 5 units (balance = 7)
    wait for clk_period;

    item_selection <= "01";  -- Select item B
    wait for clk_period;  -- Should dispense item B

    -- Scenario 4: Out of stock for item C
    new_order_request <= '1';  -- Request new order
    wait for clk_period;
    new_order_request <= '0';   -- Reset new order request

    item_available <= '0';  -- Item C is not available
    coin_input <= "01";  -- Insert 1 unit
    wait for clk_period;
    coin_input <= "10";  -- Insert 5 units
    wait for clk_period;

    item_selection <= "10";  -- Request item C (which is out of stock)
    wait for clk_period;  -- Should trigger error signal

    -- Scenario 5: Buy item C after it becomes available
    item_available <= '1';  -- Now item C is available
    wait for clk_period;

    item_selection <= "10";  -- Select item C
    wait for clk_period;  -- Should dispense item C

    -- Scenario 6: Repeat new order for item A and have sufficient balance
    new_order_request <= '1';  -- Request new order
    wait for clk_period;
    new_order_request <= '0';   -- Reset new order request

    coin_input <= "10";  -- Insert 5 units
    wait for clk_period;
    coin_input <= "01";  -- Insert 1 unit
    wait for clk_period;

    item_selection <= "00";  -- Select item A again
    wait for clk_period;  -- Should dispense item A

    -- Finish the simulation
    wait;
end process;
```

User Guide

In this part we will analyze the code and explain usage of each component and explain their inputs , outputs , entities and architectures.

Files and Components

```
Coin_Input_Controller.vhd
DIspense_Controller.vhd
Item_selector.vhd
Vending_Machine_TB.vhd
Vending_Machine.vhd
```

As you can see we have 3 components and main file for vending machine which is made of other 3 components and we have test _bench file for our project.

Instruments

Let's check them all one by one :

- **Coin_Input _Controller : this component's duty is to get coin as input and give us a balance as output as you can see on the photo of its entity. (all components have clk and reset as their input so I would not kepp mentioning them for all sections.**

```
ENTITY Coin_Input_Controller IS
    PORT (
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        coin_input : IN STD_LOGIC_VECTOR (1 DOWNTO 0); --(1, 5, 10)
        balance : OUT INTEGER
    );
END Coin_Input_Controller;
```

Now take a look at its architecture :

```vhdl
ARCHITECTURE Behavioral OF Coin_Input_Controller IS
    SIGNAL internal_balance : INTEGER := 0;
BEGIN
    PROCESS (clk, reset)
    BEGIN
        IF reset = '1' THEN
            internal_balance <= 0;
        ELSIF rising_edge(clk) THEN
            CASE coin_input IS
                WHEN "00" =>
                    IF (internal_balance + 1) < 128 THEN
                        internal_balance <= internal_balance + 1;
                    END IF;
                WHEN "01" =>
                    IF (internal_balance + 5) < 128 THEN
                        internal_balance <= internal_balance + 5;
                    END IF;
                WHEN "11" =>
                    IF (internal_balance + 10) < 128 THEN
                        internal_balance <= internal_balance + 10;
                    END IF;
                WHEN OTHERS => NULL;
            END CASE;
        END IF;
    END PROCESS;

    -- Assign internal_balance to balance output
    balance <= internal_balance;

END Behavioral;
```

Here we got a behavioral architecture for our component so let's start to explain it:

We have an internal signal here name internal_balance and here I will brief the reason of using internal signals and in other part I won't repeat it again :

1. it's the main difference between combinational logic and sequential logic if we assign the value in clk edges to balance directly it would lead to combinational logic.
2. Storage : signals in vhdl acts like a storage element and in digital design it corresponds to using Flip_Flops.
3. Using an internal signal allows you to control when the balance updates happen. In our design, the balance is only updated when there is a rising edge on the clock. If we were to use the balance output directly, we would have a situation where it could change on every input signal change, which could lead to unpredictable behavior in a synchronous circuit.

The other part in this component is clear and easy to understand we just have a progress which increase the balance based on the coin input and you may notice the condition that controls balance not to exceed 128 , we use it to have an limitation for our system and avoid potential errors from wrong inputs but we could've not to use it .

- Dispense_Controller : so here we get balance , item price and extra item request as inputs and dispense signal and change (of balance) and error as output.

```vhdl
ENTITY Dispense_Controller IS
    PORT (
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        balance : IN INTEGER;
        item_price : IN INTEGER;
        new_order_request : IN STD_LOGIC; --for extra item
        dispense_signal : OUT STD_LOGIC;
        error_signal : OUT STD_LOGIC;
        change_return : OUT INTEGER
    );
END Dispense_Controller;
```

Take a look at its architecture :

```vhdl
ARCHITECTURE Behavioral OF Dispense_Controller IS
BEGIN
    PROCESS (reset, clk)
    BEGIN
        IF reset = '1' THEN
            dispense_signal <= '0';
            change_return <= 0;
            error_signal <= '0';
        ELSIF rising_edge(clk) THEN
            IF new_order_request = '1' THEN
                dispense_signal <= '0';

                IF balance < item_price THEN
                    error_signal <= '1';
                ELSE
                    error_signal <= '0';
                END IF;
            ELSIF balance >= item_price AND item_price > 0 THEN
                dispense_signal <= '1';
                change_return <= balance - item_price;
                error_signal <= '0';
            ELSE
                dispense_signal <= '0';
                IF balance < item_price THEN
                    error_signal <= '1';
                ELSE
                    error_signal <= '0';
                END IF;
            END IF;
        END IF;
    END PROCESS;
END Behavioral;
```

It is only an obvious progress I forgot to talk about reset in previous section here I tell you if reset is 1 everything will be 0 so now about other parts in rising edge of clk if user has an extra request dispense signal would be 0 it means we won't dispense item until user choose next item but now for dispensing we're gonna have an error if user has not enough balance and if he has we will dispense item and return change to him from balance.

- **Item_Selector :** item available and item selection are our inputs and item price is our output.

```
ENTITY Item_Selector IS
    PORT (
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        item_selection : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        item_available : IN STD_LOGIC;
        item_price : OUT INTEGER
    );
END Item_Selector;
```

take a look at its architecture :

```
ARCHITECTURE Behavioral OF Item_Selector IS
BEGIN
    PROCESS (reset, clk)
    BEGIN
        IF reset = '1' THEN
            item_price <= 0;
        ELSIF rising_edge(clk) THEN
            IF item_available = '1' THEN
                CASE item_selection IS
                    WHEN "00" => item_price <= 5; -- item A
                    WHEN "01" => item_price <= 7; -- item B
                    WHEN "10" => item_price <= 10; -- item C
                    WHEN OTHERS => item_price <= 0; -- invalid selection
                END CASE;
            ELSE
                item_price <= 0;
            END IF;
        END IF;
    END PROCESS;
END Behavioral;
```

So simple we just assign item price based the item that is selected and we get it from input and also if it's available otherwise nothing will happen .

- **Vending_Machine :** so now we explain our main part , the Vending Machine here we have used structural style of coding and we have the other tree elements as components here so all inputs and output are clear just take a look at the entity here :

```
ENTITY Vending_Machine IS
    PORT (
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        coin_input : IN STD_LOGIC_VECTOR (1 DOWNTO 0); -- 1,5,10
        item_selection : IN STD_LOGIC_VECTOR (1 DOWNTO 0); -- A, B, C
        item_available : IN STD_LOGIC; -- assume as control elsewhere
        new_order_request : IN STD_LOGIC; -- extra order
        balance_display : OUT STD_LOGIC_VECTOR (6 DOWNTO 0); -- two seven segments
        change_return : OUT INTEGER; -- amount of the change
        dispense_signal : OUT STD_LOGIC; -- signal for dispense
        error_signal : OUT STD_LOGIC -- signal for error
    );
    SIGNAL balance : INTEGER;
    SIGNAL item_price : INTEGER;
    SIGNAL dispense_sig : STD_LOGIC;
    SIGNAL error_sig : STD_LOGIC;
END Vending_Machine;
```

And we have 4 internal signals here to make our system sequential (they are storage elements)

So let's analyze the architecture :

We got 3 component of the mentioned ones on previous pages and we have connected the inputs and outputs of our machine to them and o won't bother to explain all the connections one by one because you can see them and they also have same names .

And at the end we assign the internal values to output signals .

That's it no need to explain anymore.

```
ARCHITECTURE Structural OF Vending_Machine IS

BEGIN
    Coin_input_Controller : ENTITY work.Coin_Input_Controller
        PORT MAP(
            clk => clk,
            reset => reset,
            coin_input => coin_input,
            balance => balance
        );

    Item_Selector : ENTITY work.Item_Selector
        PORT MAP(
            clk => clk,
            reset => reset,
            item_selection => item_selection,
            item_available => item_available,
            item_price => item_price
        );

    Dispense_Controller : ENTITY work.Dispense_Controller
        PORT MAP(
            clk => clk,
            reset => reset,
            balance => balance,
            item_price => item_price,
            new_order_request => new_order_request,
            dispense_signal => dispense_sig,
            error_signal => error_sig,
            change_return => change_return
        );
    dispense_signal <= dispense_sig;
    error_signal <= error_sig;
    balance_display <= STD_LOGIC_VECTOR(to_unsigned(balance, balance_display'length));

END Structural;
```

The end.