# CS M152A Lab 3

## Finite State Machine Design: Vending Machine

*In this lab, you will design and implement a finite state machine (FSM) for a vending machine.*

## Introduction

For this lab, you will design a finite state machine that matches the specified behavior

This lab will be based on simulation only; no FPGA use will be involved. you are going to implement your design in Verilog HDL. At the end of the lab, you are expected to present a design project with source code and testbench.

## Overview and Example

Finite State Machines (FSMs) are a powerful tool that can be used to model many real-world systems and are particularly useful for the behavioral modeling of sequential circuits. An FSM has a finite number of 'states' and can be in any one of these states at a given time. The machine transitions from one state to another based on the inputs it receives and the state that it is currently in. The machine must begin operation in an initial state. Given below is a simple example of an FSM for a turnstile, and the Verilog code to implement it. This FSM is a Moore machine because the output (*is_locked*) depends only on the present state. A Mealy machine is an FSM whose outputs depend on both the present state and input.

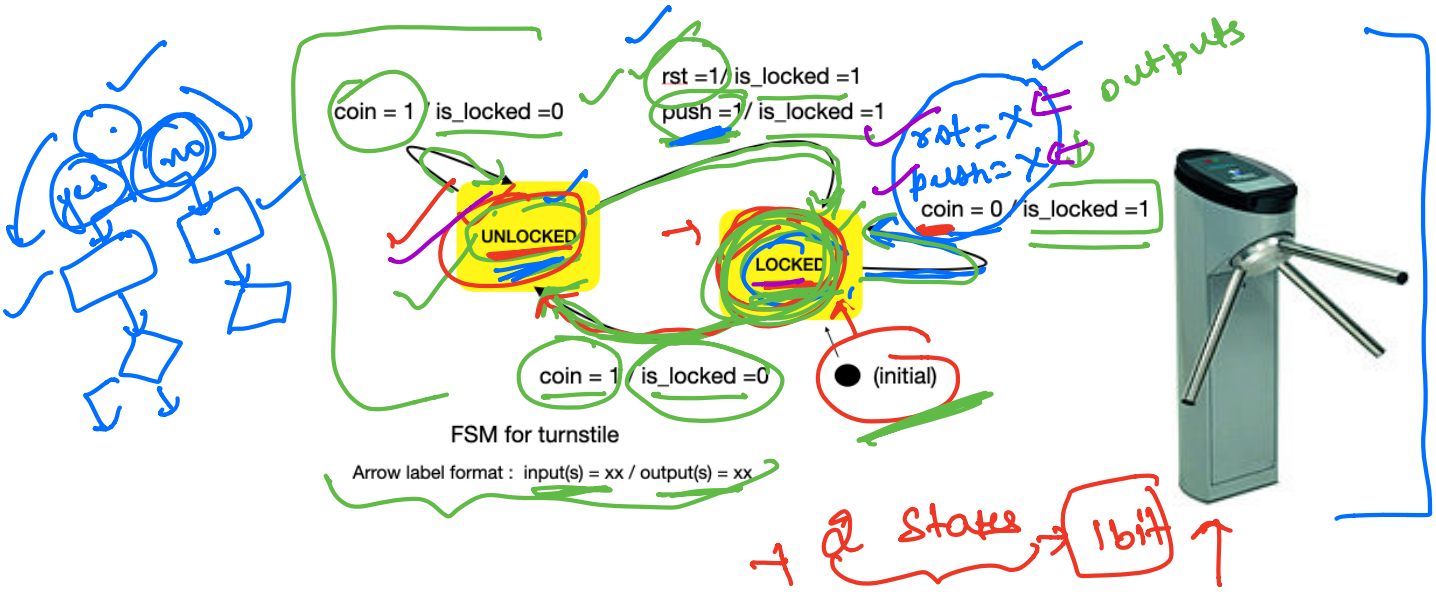
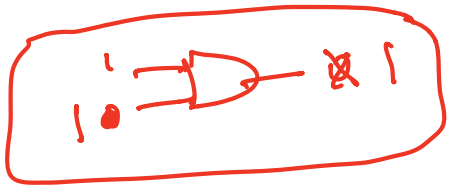
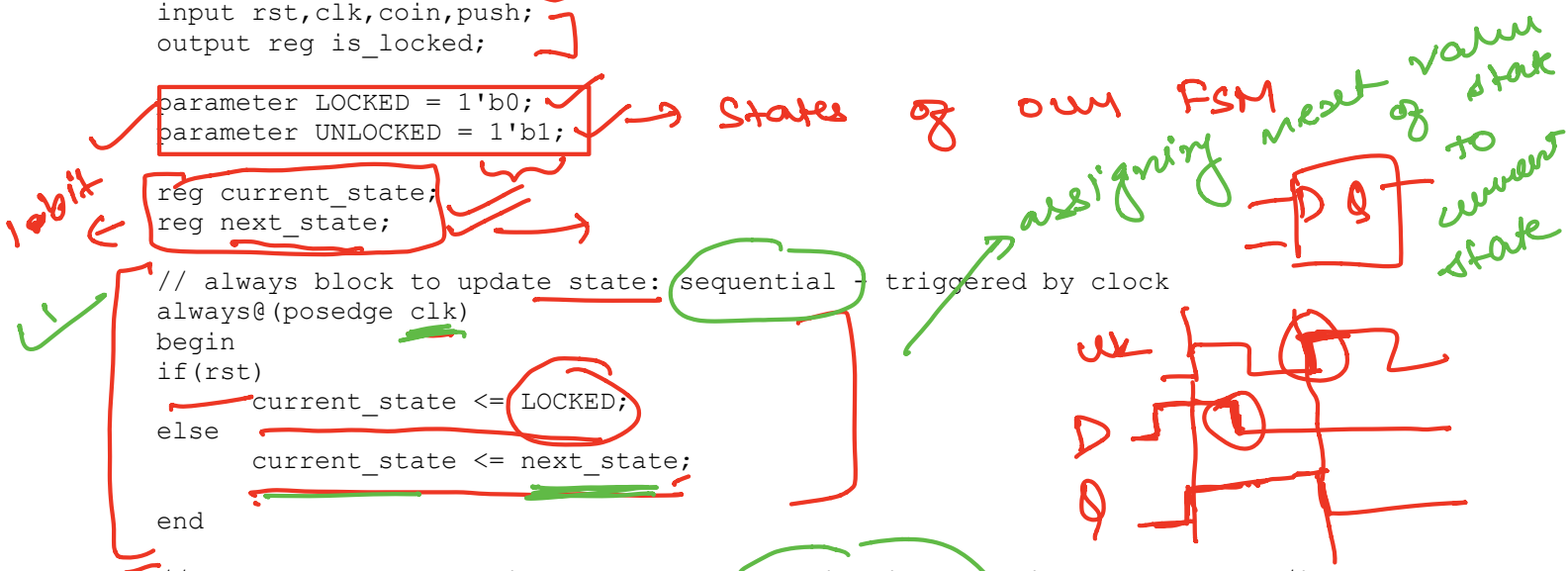



Fig 1: FSM for a Turnstile (allows a user to push through when a coin is inserted)



```
module turnstile(rst,clk,coin,push,is_locked);

input rst,clk,coin,push;
output reg is_locked;

parameter LOCKED = 1'b0;
parameter UNLOCKED = 1'b1;

reg current_state;
reg next_state;

// always block to update state: sequential - triggered by clock
always@(posedge clk)
begin
if(rst)
      current_state <= LOCKED;
else
      current_state <= next_state;

end

// always block to decide next_state: combinational- triggered by state/input
always @(*)
case(current_state)
LOCKED : begin
            if(coin)
                  next_state = UNLOCKED;
            else
                  next_state = LOCKED;
             end
UNLOCKED:
         begin
            if(coin)
                  next_state = UNLOCKED;
            else if(push)
                  next_state = LOCKED;
            else
                  next_state = UNLOCKED;  //stay unlocked until push
             end

endcase

//always block to decide outputs: triggered by state/inputs; can be comb/seq.
always @(*)
case(current_state)
LOCKED : begin
            is_locked = 1'b1;
            end
UNLOCKED:
         begin
            is_locked = 1'b0;
             end

endcase

endmodule
```



## System Specifications:

This section describes the behavior of the FSM that you are expected to design.

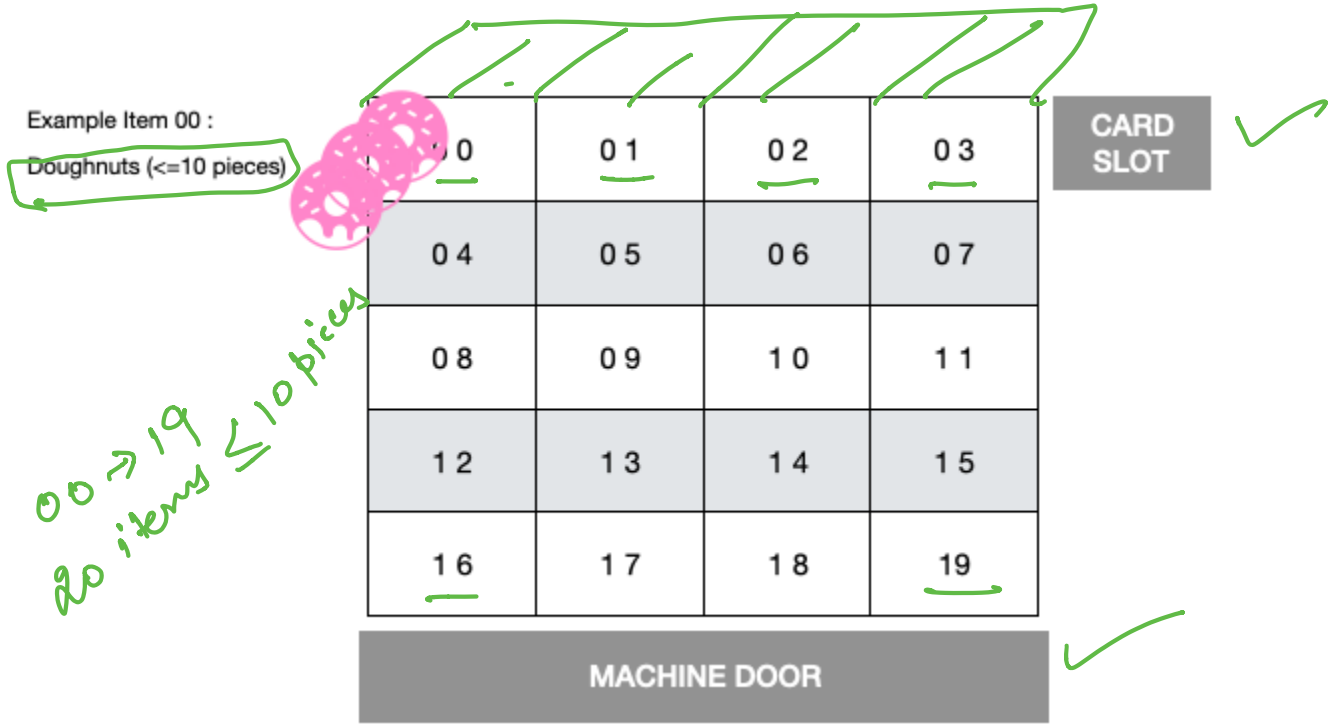



**Fig 2: Vending Machine**

Design a vending machine with the following characteristics:

1. The vending machine has 20 different snacks for sale. Each snack has a two-digit code (00 to 19).
2. Each snack is stored in a separate slot like the vending machines on campus (see Fig. 2). There can be up to 10 units of the snack stored in 1 slot. A counter for every slot keeps track of the number of units of the snack remaining in that slot.
3. A buyer can purchase only 1 item at a time.
4. The machine only accepts payment by card.

**Actions/States**:

1. **Reset**: When reset = 1, all item counters and outputs are set to 0. The machine goes to the idle state when the reset becomes 0.
2. **Idle**: This is a state where the machine waits for a new transaction to begin. In this state, all outputs are set to 0, and the machine waits in this state until CARD_IN goes high to signal the beginning of a new transaction. The idle state is the initial state of this FSM.

3. **Re-load:** All snack counters are set to 10, i.e. the machine is fully re-loaded. A re-load can only be done when the machine is idle. A new transaction cannot begin when the machine is re-loading.
4. **Transact:** When card is inserted (CARD_IN = 1) - wait for item selection (see action : get code)
    a. If selection is valid (i.e. the code is a number between 00 and 19 and there are a non-zero number of items corresponding to that code left in the machine), display the $ amount of the selection on COST<2:0> as per Table 1, and wait for the VALID_TRAN signal (represents a valid/invalid transaction signal from the bank).
        i. If VALID_TRAN = 1 : VEND selected item (see action: vend)
        ii. If the VALID_TRAN signal does not go high within 5 clock cycles, the transaction failed. Set the 'FAILED_TRAN' bit to high, and go to the idle state.
    b. If the selection is invalid, set the 'INVALID_SEL' bit to high and go to the idle state.

5. **Vend**:
    a. decrement counter of the corresponding item by one (i.e., the item is dispensed)
    b. set VEND to 1
    c. Wait for the door-open signal to go HIGH (open) and then (LOW) (i.e. the door opened, the item was collected and the door was closed) to begin a new transaction (idle). If the door does not open for 5 clock cycles, go to the idle state.

6. **Get Code:**

1. The same bus (ITEM_CODE<3:0>) is used to enter the 2 digit item code sequentially, the way you would press numbers on a keypad. The press is modeled by the input 'key_press'. Thus, the item_code is read-only when key_press = 1.

2. Wait up to 5 clock cycles for each digit. If no digit is entered, or if a digit is entered and there is no second digit for 5 clock cycles, go to the idle state.

Example: For item '16' entered over 3 clock cycles:

clk cycle 1: ITEM_CODE = 0001 KEY_PRESS = 1 // read 1.
clk cycle 2: ITEM_CODE = 0001 KEY_PRESS = 0 // not read. max 4 cycles left
clk cycle 3: ITEM_CODE= 0110 KEY_PRESS = 1 // read 6.

| ITEM CODE | COST ($) | ITEM CODE | COST ($) |
|---|---|---|---|
| 00, 01,02,03 | 1 | 12,13,14,15 | 4 |
| 04,05,06,07 | 2 | 16,17 | 5 |
| 08,09,10,11 | 3 | 18,19 | 6 |

**Table 1: ITEM CODE vs COST**

**Inputs and Outputs:**

The I/O of the module vending_machine should be exactly as shown in Figure 3.

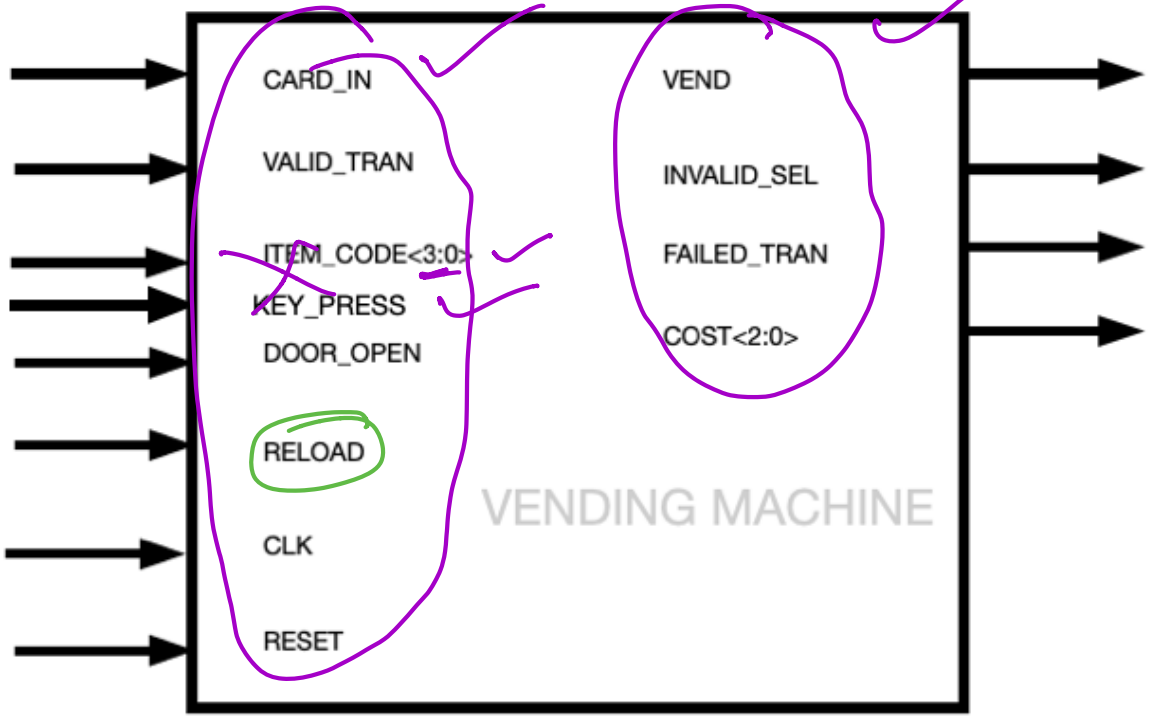



**Figure 3: VENDING MACHINE I/O**

**INPUTS:**

| INPUT | SIZE/ BEHAVIOR |
|---|---|
| CLK | 1 bit. System clock ( T= 10 ns) |
| RESET | 1 bit synchronous reset. When high, set all item counters, outputs to 0 and go to the idle state. |
| RELOAD | 1 bit. Reload the machine (set all item counters to 10) |
| CARD_IN | 1 bit. Stays high as long as the card remains inserted. |
| ITEM_CODE <3:0> | 4-bit signal to input item code. The 2 digit item code is entered one digit at a time. |
| KEY_PRESS | 1 bit. ITEM_CODE is valid for reading when this signal is high. |
| VALID_TRAN | 1 bit. HIGH = transaction using the card is valid (can go high any time after item selection is determined to be valid) (card does not need to be inserted when this occurs) |
| DOOR_OPEN | 1 bit. HIGH = The vending machine door is open<br>(This can occur any time after the 'VEND' goes high.) |

**OUTPUTS:**

| OUTPUT | SIZE/BEHAVIOR |
|---|---|
| VEND | 1 bit. Set to HIGH once the transaction is deemed to be valid. Set to LOW once DOOR_OPEN goes high and then low/ or if the door does not open in 5 clock cycles. |
| INVALID_SEL | 1 bit. Set to HIGH if:<br>1. Only 1 digit of ITEM_CODE is entered and there is no 2nd digit after 5 clock cycles or if no digit is entered for 5 clock cycles<br>2. The 2 digit ITEM_CODE is invalid (Ex. 23)<br>3. The counter for one of the items is 0. |
| COST<2:0> | 3 bits. Set to 000 by default. Set to the cost of an item once item code is entered, and remains at this value until a new transaction begins. (Ex. $5 = 101) |
| FAILED_TRAN | 1 bit. Set to 1 if VALID_TRAN signal does not go high within 5 clock cycles of determining the ITEM_CODE |

**Things to think about/Hints :**

1. Draw your FSM diagram before you write any code. What are your states? What inputs cause a state transition? Once your FSM is designed, use the example provided as a template to convert it to Verilog.
2. You can use the 'actions' as a starting point for your state definitions, but feel free to combine actions into one state or split up actions into multiple states if required. Aim for a clean, logical design with as few states as possible. (As a reference - one possible solution has around 10 states. Your design can have more/fewer.)
3. What happens if any input signal occurs when it is not supposed to? Does your FSM handle this correctly? Your vending machine cannot crash or take an action that violates the design specifications.
4. Additional signals, counters, etc. may be defined/instantiated within the module (invisible outside the module) to help synchronize operations and reduce the number of states.

## Deliverables

When you finish, the following should be submitted for this lab:

1. **Verilog source code** for the "vending_machine" module. The file should be named exactly as "vending_machine.v" and the module and port names should exactly match names defined in Figure 3. It is very important as your code is automatically run. Also, note that this code should be completely synthesizable. ***There is no restriction on the naming of the submodules (if any) but make sure to place all the submodules in the vending_machine.v file.
2. **Verilog testbench** you used to evaluate your design. Note that your testbench is graded based on the correctness of the waveforms generated in your report. Please name the file "testbench_UID.v" where UID is your UCLA ID. It is too time-consuming to exhaustively test all possible input/state combinations for this design. Make sure you test **at least** one successful transaction and 4 special cases to get 80% of the marks associated with simulation. A testbench that identifies and tests all/most 'interesting' cases will get full marks for simulation.
3. **Lab Report**: Please refer to the syllabus for the basic components of your lab report. Note:
    1. Include the FSM diagram and explain the states and transitions of your FSM. Bullet-points/tables are preferred over lengthy paragraphs. Draw a diagram similar to Figure 1 to represent the working of your FSM. This diagram can be drawn by hand or using the software. You can label the arrows (1,2,3, a,b,c, etc.) and put the text associated with each arrow in a table if you feel your diagram is getting very crowded. Include this diagram in your report.
    2. Explain how you test your design and include simulation waveforms
    3. Schematics can be generated from ISE but please explain how your Verilog code results in the RTL generated. Include the 'Design Summary' section of the synthesis report and the summary of your implementation (map) report and write 1-2 lines on the conclusions you draw from these reports.
    4. Please name your report "report_UID.pdf" where UID is your UCLA ID.

4. **Video Demo**: Please refer to 'Syllabus' for details. The ideal video length is 5-10 minutes. Please name your video "video_UID.xxx" where UID is your UCLA ID and xxx is the video format extension.

**Submission Checklist:**

- There is no late submission for this project.
- It is recommended that you have your submission ready in advance so that you do not face problems due to long upload times etc.

**FAQ:**

1) What should the behavior be when the Door never goes low?
Example: Someone picks up an item but the door gets stuck.

If the door never goes low - the machine should not do anything until the door is closed. It should just remain in that state. The only thing that can remove it from this state is a door close or a reset.

2) When an invalid transaction occurs the output goes high and the machine goes to the idle state. Is the invalid transaction signal also set to 0 upon transitioning to idle state?

In the idle state, all outputs are set to 0. There could be an intermediate 'failed tran' state for one cycle if required.

4) When happens after a successful transaction? How is this different from reset?
After 'vend', the machine goes to the idle state. It does not reset. On transition to the idle state, all outputs are set to 0, but the item counters are not affected. On reset, both item counters and outputs are set to 0.

5) What happens when CARD_IN goes low in the middle of the transaction? Can you begin a transaction while reloading?

CARD_IN can go low any time after the transaction begins (assume the card's info. is stored by the machine as soon as it is inserted). The 'RELOAD' input is valid only when the machine is idle. Additionally, card_in is ignored if it goes high after reload becomes high. A new transaction can begin only once reload goes low.

6) What if CARD_IN stays as 1 throughout the transaction and through it finishing? Then should another transaction start immediately once the current transaction finishes?
Yes.

7) Changing inputs at the same clock edge as the state update is causing unpredictable behavior.
You can change the inputs (in the testbench) at the negative edge of the clock (or sometime before the clock edge) so that they are stable at the clock edge.