CS M152A Lab 3: Finite State Machine - Vending Machine

Arnold Pfahnl

TA: Mohit Garg, Winter 2021

Contents

1 Introduction

In this lab, we implement a finite state machine used to operate a vending machine. The vending machine has 20 unique two-decimal digit codes for different snacks. Each code represents a slot within which there can be up to 10 units of a snack. This necessitates a counter for each slot. Other requirements include a buyer only being able to purchase 1 item at a time, and card-only transactions.

The finite state machine is implemented in module *vending_machine*. The inputs and outputs of the module are based on the following specifications in Table 1 and Table 2, respectively.

Input	**Size/Behavior**
CLK	100 MHz system clock.
$RESET$	Synchronous reset signal. If high, item counters and outputs are set to 0, and the machine enters the idle state.
$RELOAD$	Reloads the machine by setting all counters to 10.
$CARD\_IN$	Signal that is high while the card is inserted into the machine.
$ITEM\_CODE[3:0]$	The two-decimal digit item code input where each digit is entered one at a time.
KEY_PRESS	Indicates that $ITEM\_CODE$ is valid for reading when high.
$VALID\_TRAN$	Indicates that the transaction using the card is valid when high.
$DOOR\_OPEN$	Indicates that the vending machine door is open when high.

Table 1: Inputs to *vending_machine* module.

Output	Size/Behavior
$VEND$	High when transaction is valid. Low when $DOOR\_OPEN$ goes high and then low or if the door doesn't open in 5 clock cycles.
$INVALID\_SEL$	High if… 1. only 1 digit of $ITEM\_CODE$ entered and there is no second digit after 5 clock cycles or if no digit is entered for 5 clock cycles. 2. the two-decimal digit $ITEM\_CODE$. 3. the counter for on of the items is 0.
$COST[2:0]$	Once an item code is entered, this is set to the cost of that item and remains at this value until a new transaction begins. 000 by default.
$FAILED\_TRAN$	High when signal doesn't go high within 5 clock cycles of determining $ITEM\_CODE$.

Table 2: Outputs to *vending_machine* module.

The cost of each item based on the item code is depicted in Table 3 below.

Item Code	Cost ($)
00, 01, 02, 03	1
04, 05, 06, 07	2
08, 09, 10, 11	3
12, 13, 14, 15	4
16, 17	5
18, 19	6

Table 3: Item costs in the *vending_machine* module.

2 Vending Machine Design

The *vending_machine* module was created based on the finite state machine depicted in Figure 1. There are seven main states: RESETTING, IDLE, RELOADING, CODE1, CODE2, TRANSACT, and VENDING.

1. The **RESETTING** state can be transitioned to from any other state as long as RESET is high. In this state, all item counters and outputs are set to zero. When RESET becomes low, the machine transitions to the IDLE state.

2. The **IDLE** state is the default state. When transitioning from another state to IDLE, all outputs are set to zero. The machine waits here until a new transaction is initiated with the CARD_IN signal going high, transitioning to the CODE1 state. Alternatively, with the RELOAD signal, the machine can transition to the RELOADING state.

3. The **RELOADING** state can only be accessed from the IDLE state with a RELOAD signal. In this state, all counters are set to 10, and the machine transitions back to IDLE when the RELOAD signal goes low.

4. The **CODE1** state can only be accessed from the IDLE state with a CARD_IN signal. In this state, we store the value of ITEM_CODE as the first digit of the two-digit code when KEY_PRESS goes high. In my implementation, the combination of KEY_PRESS high and the code being stored is indicated by the CODE1_STORED internal signal. Additionally, an internal INVALID_SEL_DETECT bit is set if this first digit is not valid. If a KEY_PRESS is not detected in 5 cycles, INVALID_SEL is set to high, and the machine transitions to the IDLE state.

5. The **CODE2** state can only be accessed from the CODE1 state after the first digit has been stored. In this state, we store the value of ITEM_CODE as the second digit of the two-digit code when KEY_PRESS goes high. In my implementation, the combination of KEY_PRESS high and the code being stored is indicated by the CODE2_STORED internal signal. Additionally, the INVALID_SEL_DETECT bit is set if this second digit is not valid or if there are no items left for the item selected. If a KEY_PRESS is not detected in 5 cycles, INVALID_SEL is set to high, and the machine transitions to the IDLE state.

6. The **TRANSACT** state can only be accessed from the CODE2 state after the second digit has been stored.

   If the selection is determined to be invalid based on INVALID_SEL_DETECT being high, INVALID_SEL is set to high and the machine transitions to the IDLE state.

   For a valid selection, the machine waits for the VALID_TRAN signal to go high. If this doesn't happen within 5 clock cycles, the machine transitions to the IDLE state and FAILED_TRAN is set to high. With VALID_TRAN set to high, the machine transitions to the final state: VENDING.

   A valid selection also means that COST will be set to the dollar amount as specified by Table 3. To determine the cost of an item, an internal function called *find_cost* takes the stored first digit and second digit as inputs, and determines the cost with a series of if-else statements. In pseudocode:

   ```
   if the first digit is 0:
       if the second digit is in [0, 3]:
           return 1
       if the second digit is in [4, 7]:
           return 2
       otherwise return 3
   if the first digit is 1:
       if the second digit is in [0, 1]:
           return 3
       if the second digit is in [2, 5]:
           return 4
       if the second digit is in [6, 7]:
           return 5
       otherwise return 6
   ```

7. The **VENDING** state can only be accessed from the TRANSACT state. In this state, the item that was selected has its counter decremented by one and VEND is set to high. Then the machine waits for the DOOR_OPEN signal to go high and then low before transitioning to the IDLE state. The machine will transition to the IDLE state if the door doesn't open for 5 clock cycles. It's also worth noting that a little peculiarity with this state is that if the door stay open without closing, the machine will stay in this state until a door close or a reset.

To take care of states that have a timeout condition, my implementation has a timer that starts when transitioning between different states. Regardless of whether or not a state will take advantage of the timer, the timer will increment by one every cycle until the state is in its fifth cycle at which point an internal TIMEOUT signal is set high. States that use a timer will recognize the TIMEOUT signal and transition as needed.

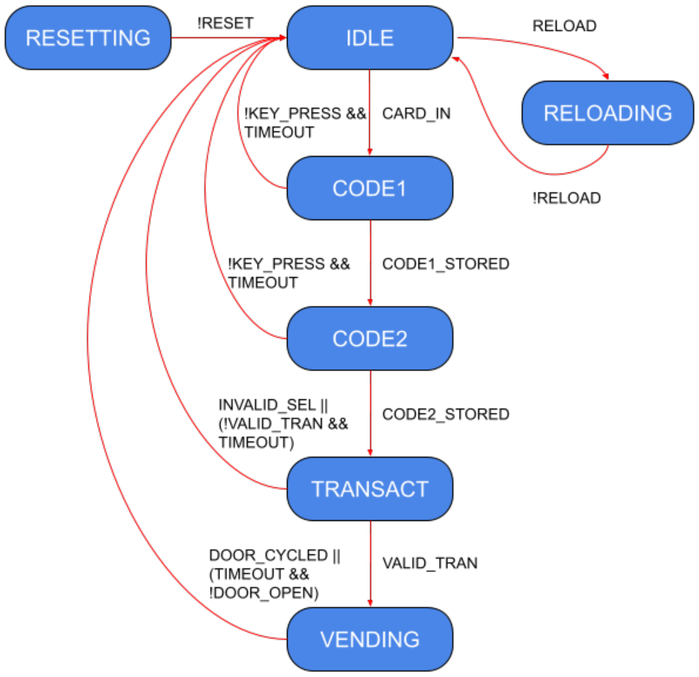

Figure 1: Finite state machine flow chart for the *vending_machine* module.

2.1 Schematics

The top level schematic in Figure 2 hides the considerable complexity of the RTL schematic in Figure 3. Due to the combination of both combinational and sequential logic used to determine states and output, the RTL features a mess of registers, muxes, and many other gates. Notably, the long vertical line of registers on the left are for the counters. There are also large muxes used to determine item cost and state transitions.

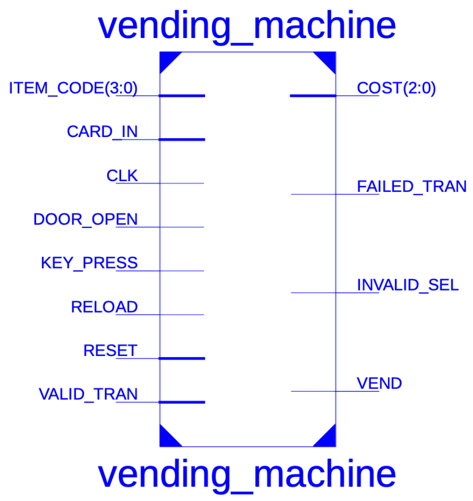

Figure 2: Top level schematic for the *vending_machine* module.

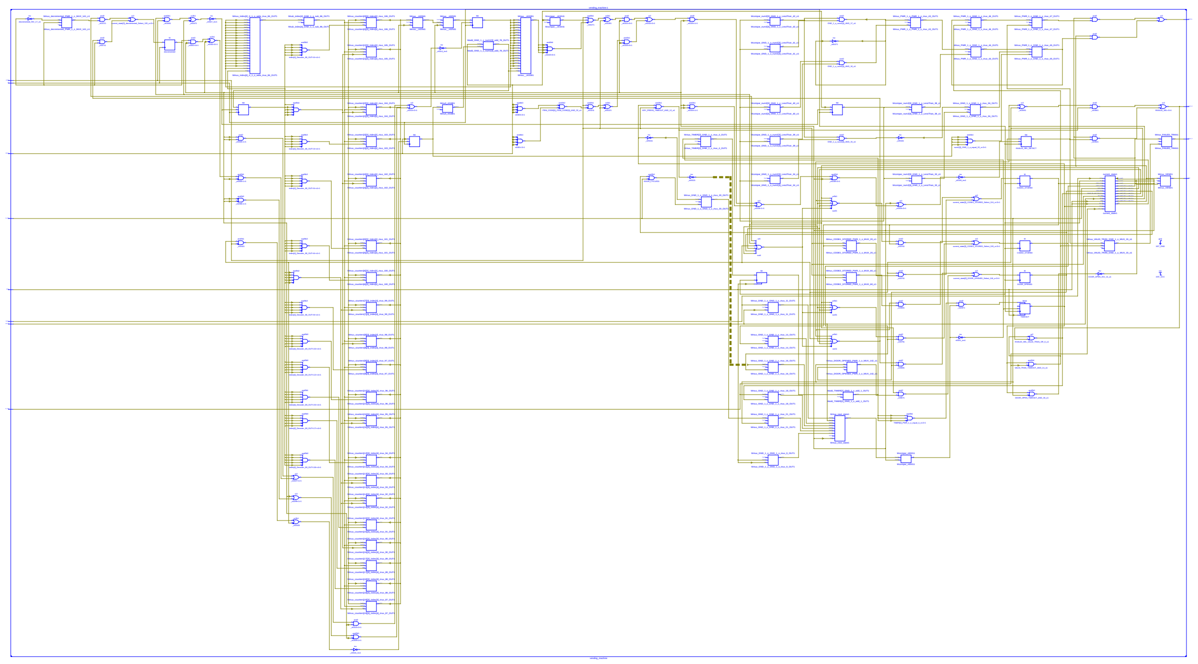

Figure 3: Detailed RTL schematic for the *vending_machine* module.

2.2 Design Summary Report

vending_machine Project Status (01/23/2021 - 23:05:50)			
Project File:	vending_machine.xise	**Parser Errors:**	No Errors
Module Name:	vending_machine	**Implementation State:**	Programming File Generated
Target Device:	xc6slx16-3csg324	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	1 Warning (0 new)
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				**[-]**
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	108	18,224	1%	
Number used as Flip Flops	108			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	180	9,112	1%	
Number used as logic	180	9,112	1%	
Number using O6 output only	162			
Number using O5 output only	0			
Number using O5 and O6	18			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number of occupied Slices	60	2,278	2%	
Number of MUXCYs used	4	4,556	1%	
Number of LUT Flip Flop pairs used	189			
Number with an unused Flip Flop	81	189	42%	
Number with an unused LUT	9	189	4%	
Number of fully used LUT-FF pairs	99	189	52%	
Number of unique control sets	6			
Number of slice register sites lost to control set restrictions	20	18,224	1%	
Number of bonded IOBs	17	232	7%	
Number of RAMB16BWERs	0	32	0%	
Number of RAMB8BWERs	0	64	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			

Number of DCM/DCM_CLKGENs	0	4	0%	
Number of ILOGIC2/ISERDES2s	0	248	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	248	0%	
Number of OLOGIC2/OSERDES2s	0	248	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	128	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	32	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	2	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	5.06			

Performance Summary			[-]
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sat Jan 23 23:05:05 2021	0	1 Warning (0 new)	1 Info (0 new)
Translation Report	Current	Sat Jan 23 23:05:23 2021	0	0	0
Map Report	Current	Sat Jan 23 23:05:31 2021	0	0	6 Infos (0 new)
Place and Route Report	Current	Sat Jan 23 23:05:37 2021	0	0	3 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Sat Jan 23 23:05:41 2021	0	0	4 Infos (0 new)
Bitgen Report	Current	Sat Jan 23 23:05:48 2021	0	0	0

Secondary Reports		[-]
Report Name	**Status**	**Generated**
ISIM Simulator Log	Out of Date	Sat Jan 23 22:51:53 2021
WebTalk Report	Current	Sat Jan 23 23:05:49 2021
WebTalk Log File	Current	Sat Jan 23 23:05:49 2021

Date Generated: 01/23/2021 - 23:05:50

3 Simulation

To test the *vending_machine* module, I created a few helper tasks. Task *reset_in*() resets all the inputs to the module to zero. Task *reset_n_load*() calls *reset_in* and also transitions the machine from IDLE to RELOADING and back. Task *vend_cycle*() takes two digits representing a two-digit item code and runs through a full cycle. These three tasks help streamline the initialization and setup of multiple tests.

My tests also include a 100 MHz clock, CLK. To aid in testing, an always block offset by half a cycle prints out the errors for INVALID_SEL and FAILED_TRAN when the corresponding outputs go high. In general, inputs to the module are offset by half a cycle to prevent strange behavior happening.

The following is a description of the tests:

1. **No reload.** The purpose of this test is to ensure that INVALID_SEL is set when trying to vend an item that has a counter at 0, and that with no reload, the counters are indeed at 0. As expected INVALID_SEL was set appropriately, and the machine transitioned back to the IDLE state. See Figure 4.

2. **Valid run.** This test does a full run of a valid vending cycle (no errors). This test establishes a baseline for making sure that the module can function at its most basic level. As expected, the test passed with no errors. See Figure 4.

3. **Full to empty slot.** This test makes sure that the vending machine vends properly, counters reach 0, and that an item will only be able to vend 10 times with only one reload. This test reloads the machine and vends the same item 12 times. As expected, the module vends 10 items and then sets INVALID_SEL the last two times. See Figure 5.

4. **Invalid selection number.** This is a simple test that checks if an invalid selection will trigger INVALID_SEL. Specifically 20 was chosen, and this test passed as expected. See Figure 6.

5. **Key press timeout.** This test actually contains two tests in one. First, a vending cycle is started but then KEY_PRESS is never set high for the first digit. This resulted in the machine returning to IDLE after 5 cycles as expected. Following the failed vend, another vending cycle is started, the first digit is entered successfully, but then the second digit times out. This resulted in the machine returning to IDLE as expected. See Figure 6.

6. **Door timeout.** This test checks to see if the machine will handle a door timeout properly. Once the VENDING state is reached, DOOR_OPEN is never set high. In testing, the machine went back to the IDLE state after 5 cycles as expected. See Figure 7.

7. **Door open.** This is an edge case where the door stays open during the VENDING state. In this case, the machine ignored TIMEOUT and stayed in the VENDING state as required by the specifications. See Figure 7.

8. **Failed transaction.** This test simulates a failed transaction with the expectation that the machine sets FAILED_TRAN and transitions back to IDLE. In testing, the machine passed this test as expected. See Figure 7.

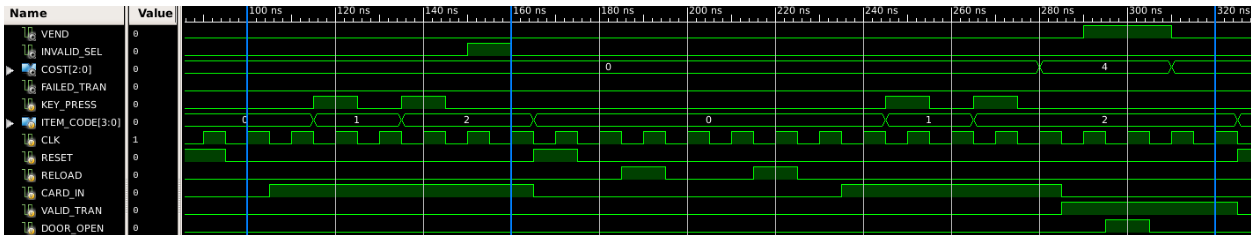

Figure 4: No reload (100 to 160 ns) and valid run (160 to 320 ns) simulation waveforms. Item 12 is selected for the no reload test after which INVALID_SEL is selected since the machine hasn't been loaded yet. The valid run ends with a successful vend and all outputs reset once the machine enters the IDLE state.

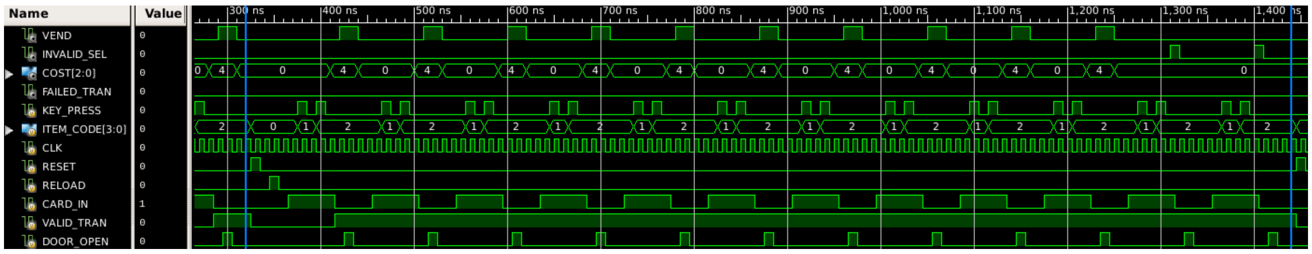

Figure 5: Full to empty on item 12 simulation waveform. Note the 10 successful vends and the two invalid selections due to the slot being empty.

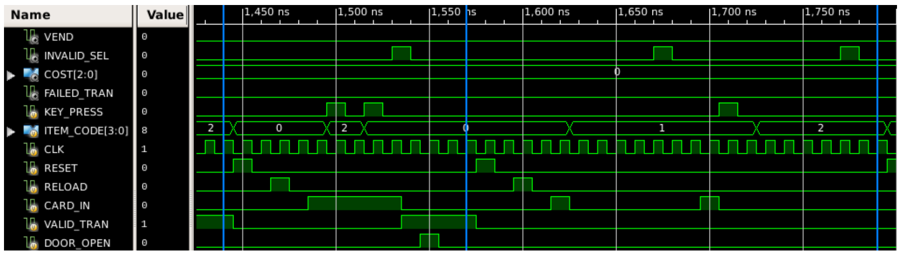

Figure 6: Invalid selection number (1440 to 1570 ns) and key press timeout (1570 to 1790 ns) simulation waveforms. The invalid selection number used is 20. Note that for the key press timeout the first time INVALID_SEL is high is 5 cycles after CARD_IN is detected, and the second time INVALID_SEL is high is 5 cycles after the first digit is processed (6 cycles after the first digit's KEY_PRESS).

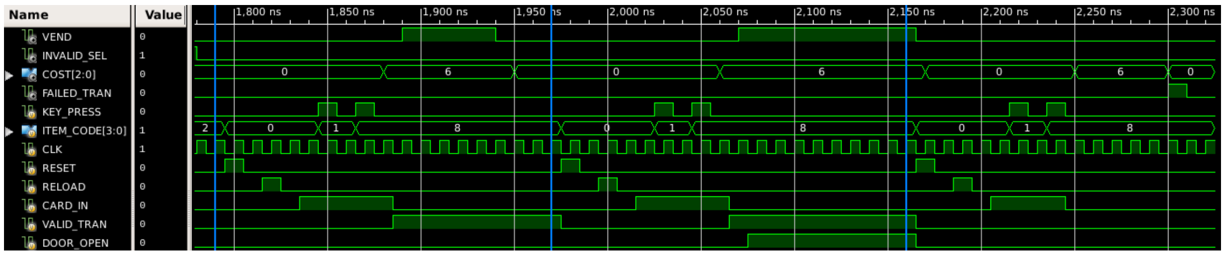

Figure 7: Door timeout (1790 to 1970 ns), door open (1970 to 2160 ns), and failed transaction (2160 to 2320 ns). For door timeout, note how VEND is high for 5 cycles before being set to low. For door open, note how VEND is high even after 5 cycles since DOOR_OPEN is stuck open. For failed transaction, note how FAILED_TRAN is set high after 5 cycles without the VALID_TRAN signal (6 cycles after saving the second digit in the CODE2 state).

4 Conclusion

In the lab, I was able to successfully create a finite state machine (FSM) that works as a vending machine. My FSM also handles some tougher edge cases confirmed by my testing.

This lab was much more involved than previous labs; however, I found the material engaging and interesting, especially in terms of complex Verilog coding and FSMs as a concept.

I did find that some aspects of the vending machine specifications weren't entirely realistic, but given the complexity of the project as is, it makes sense to not have us worry about those conditions. In general, I thought the project was well put together, and I didn't run into too many issues.