CSM152A-LAB 4

Lab 5

Yichen Lyu

UID: 004940413

May 24, 2020

1. INTRODUCTION

This lab will explore how to design, implement and test using Xilinx a finite state machine (FSM). The FSM this lab will use is a mealy machine, whose output depends on both its input and state. The final project should be a vending machine with 20 snacks. Each snack has a price and a number left. If a card is detected, the transection process starts and only one item can be purchased at the same time.

2. DESIGN DESCRIPTION

The overall FSM will have two internal arrays: COSTS[] and counts[]. COSTS[] is a constant array that stores the price of an item based on its item code. counts[] contains the number of items left based on item code. There is also a 5-bit counter, cycle_count, which counts the number of cycles that the FSM has been waiting in one state. The current state and the next state are recorded in vend_state and nx_vend_state. Every time there is a posedge clk, vend_state will be given the value of nx_vend_state. The input and output signals are shown below.

The overall design of the FSM is given in the graph below.

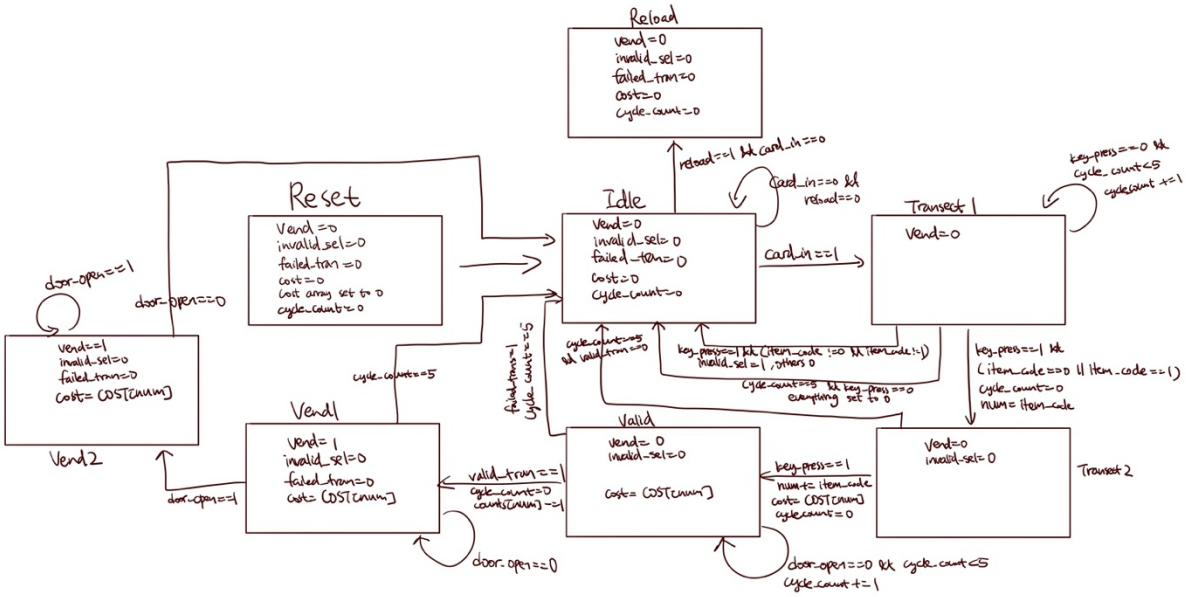

Figure 1: Overall Combinational Circuit Design

The FSM is implemented using a case statement using vend_state. Within each case, if statements are used to assign outputs based on inputs.

2.1 RESET

1. When rst is 1, vend_state will be set to RESET.
2. The program enters case RESET and sets every output to 0 and counts[] to be 0 with a for loop.
3. If rst == 0, nx_vend_state is set to IDLE. Otherwise, it loops in RESET.

2.2 IDLE

1. All outputs are set to 0.
2. If card_in is 1, IDLE's next state is set to TRANSECT1.
3. If card_in is 0, but reload is 1, it sets nx_vend_state to RELOAD.
4. Otherwise, it keeps looping back to IDLE, as indicated in the design.

2.3 RELOAD

1. All outputs are 0, but it does not have to intentionally set any output because it can only come after IDLE, who has already set every output to 0.
2. It sets all product counts to 10. (It is implemented by setting counts[] to 10 using a for loop).
3. It always sets its next state to IDLE.

2.4 TRANSECT1

1. This state waits for the first digit of item code.
2. If key_press and card_in are both 1, it takes in the value in item_code and check if it is valid as a first digit of the overall item code. For the first digit, the value in item_code can only be 0 or 1. If it is not valid, invalid_sel is set high and nx_vend_state is IDLE.
3. If item_code is valid, nx_vend_state is TRANSECT2.
4. If key_press is not high after 5 clock cycles, we set nx_vend_state to IDLE. (The cycle count is stored in cycle_count, which counts up to 9. This is because the always loop the case statement is in is triggered both on posedge and negedge. cycle_count is increased by 2 each cycle. Thus, we double the cycle count from 5 to 9)

2.5 TRANSECT2

1. This state waits for the second digit.
2. Every value will be valid as a second digit, so invalid_sel is always 0.
3. If key_press becomes high, it calculates the overall item code. It sets cost using the corresponding price of the item code. nx_vend_state is VALID.
4. As long as key_press is low, we loop in TRANSECT2.
5. If key_press does not become high in 5 cycles, nx_vend_state is set to IDLE.

(it adds the second item_code to 10*first_item_code, whose value is stored in a register num. cost will be set to the value of COST[num]). nx_vend_state will be set to VALID. As long as key_press is low, we loop in TRANSECT2. If key_press does not become high in 5 cycles (using the same counting way in TRANSECT1), nx_vend_state is set to IDLE.

2.6 VALID

1. This state waits for valid_tran to become high.
2. If valid_tran is high, this transection is validated and we can start vending: it sets nx_vend_state to VEND1.
3. If valid_tran is low, it keeps looping to itself up to 5 cycles.
4. If valid_tran does not go high in 5 cycles, the transection fails. Failed_tran is set high and nx_vend_state becomes IDLE.

After the 2 digits are received, num is set to the correct value of the overall item code. So in VALID, cost is still COST[num].

2.6 VEND1

1. The machine starts vending. Vend is set high, and VEND1 waits for door_open to be 1.
2. If door_open is high, its nx_vend_state becomes VEND2.
3. If door_open is low, it loops back to itself for up to 5 cycles.
4. After waiting for 5 cycles, nx_vend_state is set to IDLE.

2.6 VEND2

1. The machine waits for door to close. Once door_open becomes low again, it goes back to IDLE.
2. If door_open does not become low, it keeps looping to VEND2.

3. SCHEMATICS

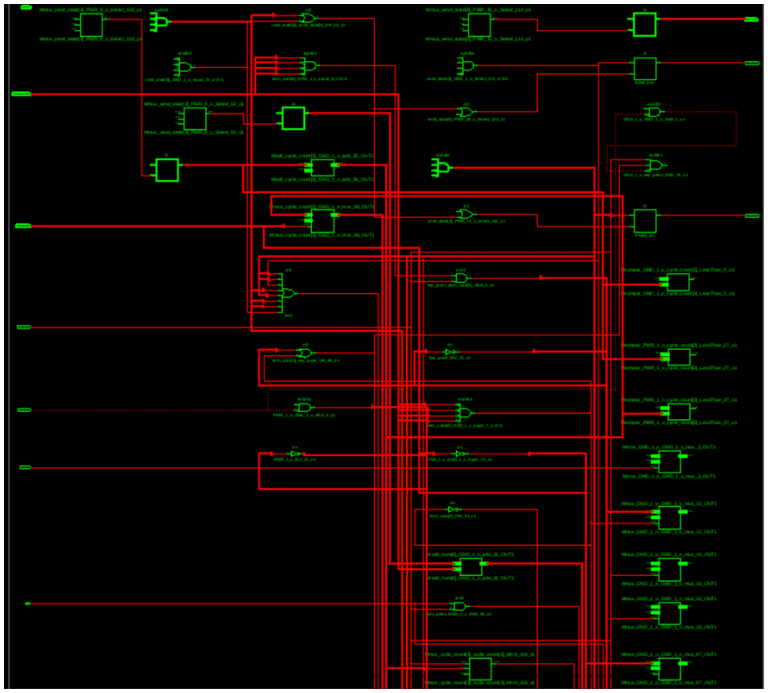

Figure 2: Auto Generated Schematics

The generated schematics is very complex. The next state is constantly influenced by the input, so we need flip-flops to combine inputs and the clock signal to determine the next state, which is fed back into the circuit to become the current state for the next round. We also need many MUXes because the outputs are based on many if-statements, depending on the state and the input.

4. SIMULATION

4.1 Complete Transection

In a complete transection, the FSM first detects reload and enters RELOAD(0010). There, it changes all item stocks to be 10, and all outputs remain 0. It then falls back to IDLE(0001). Then, it detects card_in and enters TRANSECT1(0011) from IDLE. It sees that key_press is high, so it takes in item_code, which is 1. Since 1 is valid as a first digit, it goes to TRANSECT2(0100). There, it waits for a cycle for key_press is high and takes in item_code, which is 4. It calculates the overall item code (1*10+4 = 14) and accordingly sets cost to 4. It then enters the next state, VALID(0111) where it sees that valid_tran is high. Thus, it goes to VEND1(0101) and sets vend to 1. Other outputs remain the same as the previous state. It detects that door_open is high, so it goes to VEND2(0110), where it sees that door_open becomes low again. It then completes the whole cycle by going back to IDLE.

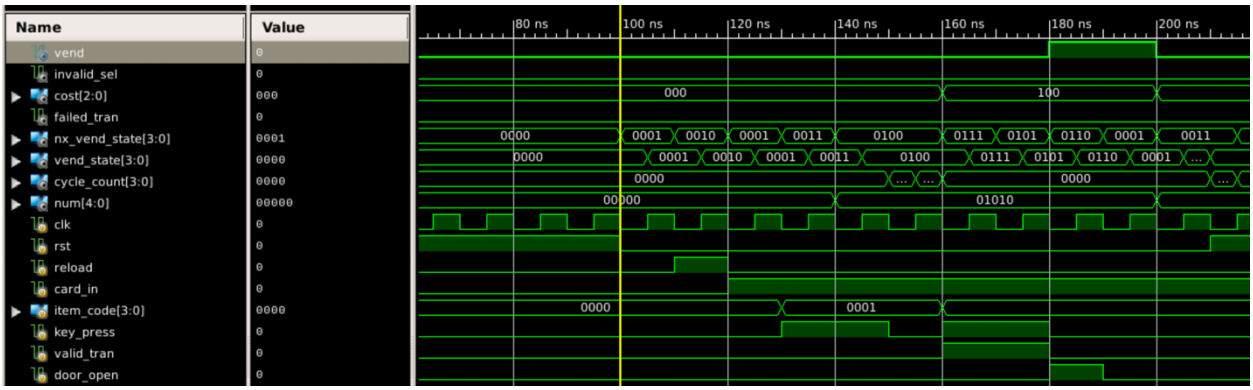

Figure 3: Successful Transection (need to change vend!)

4.2 Timeout on First Item Code

In this case, key_press is not set high after the FSM detects card_in and enters TRANSECT1 (0011) from IDLE (0000). Vend_state loops in TRANSECT1 for 5 cycles and then goes to IDLE. During this process, all outputs remain 0.

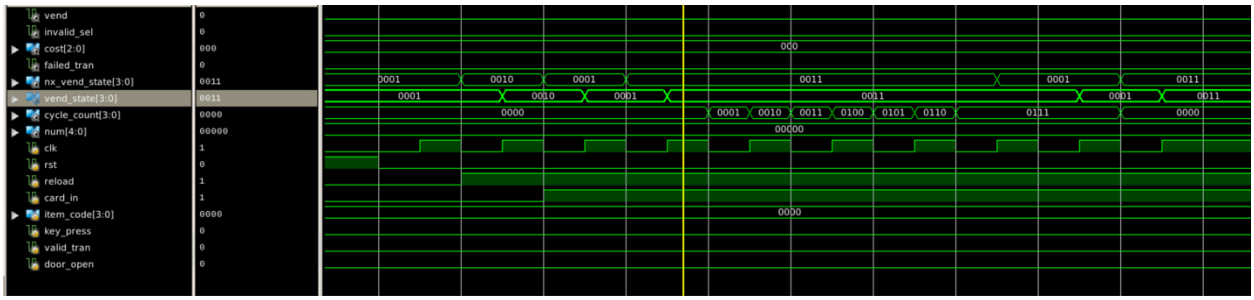

Figure 4: TRANSECT1 Timeout

4.3 Invalid Item Code

In this case, the FSM enters TRANSECT1(0011) and receives a high key_press. It then takes in the value in item_code, which is 4. The first digit can only be 0 or 1, so it knows that this digit is invalid. It then sets invalid_sel to be high and go back to IDLE(0001). In the simulation below, card_in and key_press are not set low, so after it enters IDLE, it keeps going back to TRANSECT1 and takes in the invalid code 4, so invalid_sel is constantly set high.

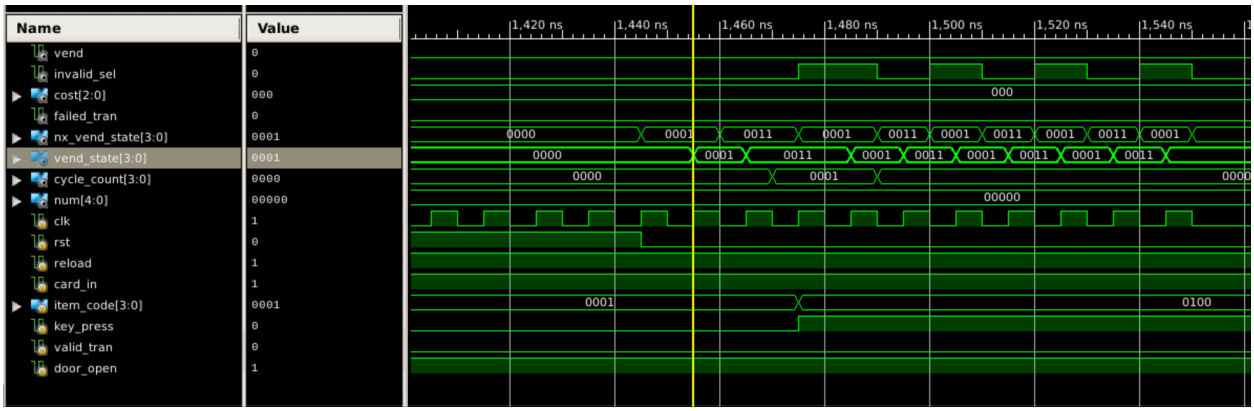

Figure 5: invalid_sel

4.4 Timeout on Second Item Code

In this case, the FSM enters TRANSECT1(0011) from IDLE(0001). It detects key_press to be high and reads in the item code (0001). It determines that 1 is a valid first digit and goes to TRANSECT2(0100). Now, key_press is low, so it waits for it to become high. After 5 cycles, it times out and goes back to IDLE.

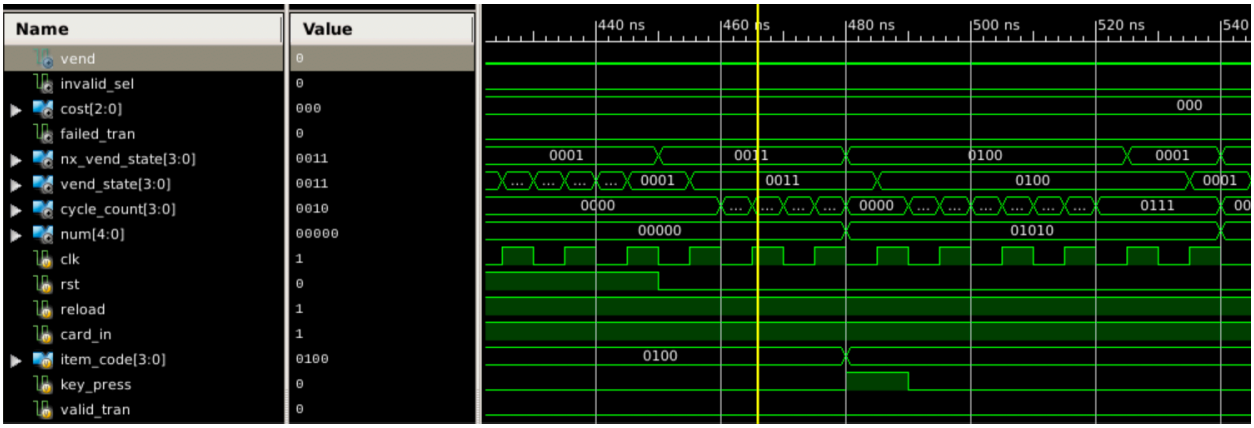

Figure 6: TRANSECT2 Timeout

4.5 Timeout on VALID_TRAN

In this case, the FSM receives high key_press within 5 cycles in TRANSECT2(0100) and enters VALID(0111). Both item codes are 1, so the overall code is 11, and the cost is 3. It waits for valid_tran to be high until timeout. In the fifth cycle, when it detects valid_tran is still low, it sets failed_tran to be 1. Then it enters IDLE(0001), where every output is set to 0 again.

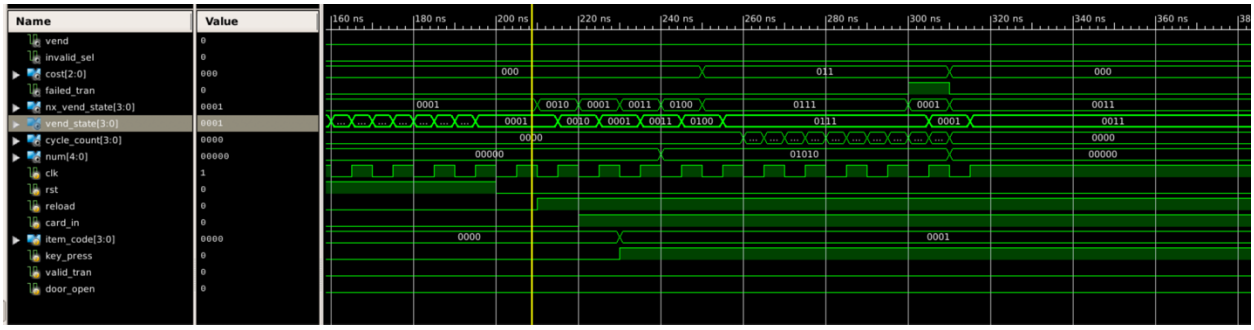

Figure 7: VALID Timeout

4.6 Timeout on DOOR_OPEN

In this case, after receiving high valid_trans in VALID (0111), the FSM enters VEND1 (0101). It waits for door_open to be 1. After 5 cycles, it times out and goes back to IDLE(0001). The cost remains the same ever since TRANSECT2(0100), which inputs item code 14 and gets the output as 4. Vend is set. Other outputs are low.

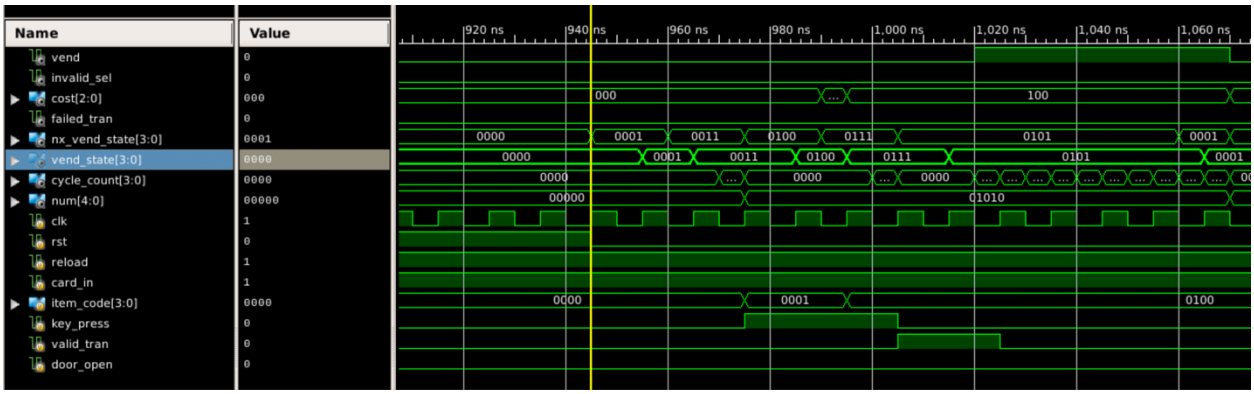

Figure 8: VEND1 Timeout (need change)

4.7 Door Does Not Close

In this case, the door opens in VEND1(0101), so the FSM enters VEND2(0110). Here, it waits for door_open to become 0. As door_open is never set low again, it loops in VEND2 forever. The cost remains the same ever since TRANSECT2(0100), which inputs item code 11 and gets the output as 3. Vend is set high in VEND1 and remains high here. Other outputs are low.

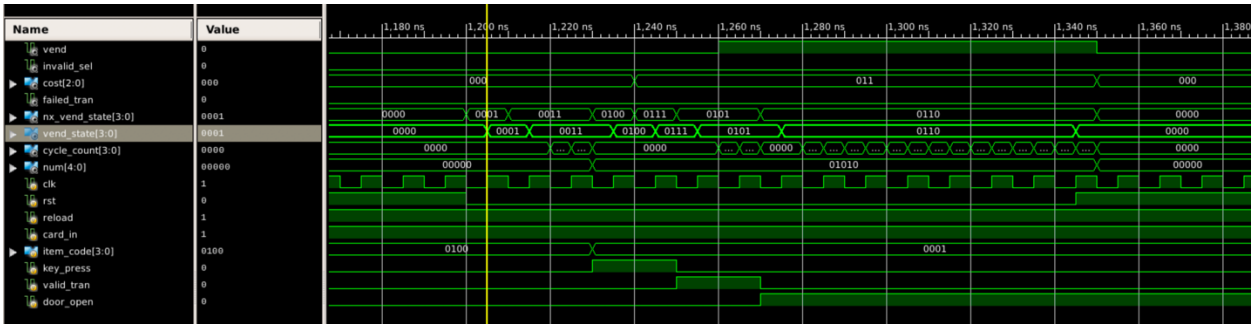

Figure 9: VEND2 Timeout (need change!)

5. CONCLUSION

This lab allows us to explore how to build a finite state machine using case statement in Xilinx ISE. It helps us learn how to divide a real-life problem into different states of the FSM and implement it with if statements to determine the outputs and the following state. Cooperating between input and state assignment is the key for the implementation of a mealy machine.

6. APPENDIX: DESIGN SUMMARY REPORT

6.1 Design Summary

vending_machine Project Status (05/21/2020 - 21:19:34)			
Project File:	lab4.xise	**Parser Errors:**	No Errors
Module Name:	vending_machine	**Implementation State:**	Placed and Routed
Target Device:	xc6slx16-3csg324	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	55 Warnings (13 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	9	18,224	1%	
Number used as Flip Flops	4			
Number used as Latches	5			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	27	9,112	1%	
Number used as logic	27	9,112	1%	
Number using O6 output only	18			
Number using O5 output only	0			
Number using O5 and O6	9			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number of occupied Slices	11	2,278	1%	
Number of MUXCYs used	0	4,556	0%	
Number of LUT Flip Flop pairs used	28			
Number with an unused Flip Flop	19	28	67%	
Number with an unused LUT	1	28	3%	
Number of fully used LUT-FF pairs	8	28	28%	
Number of unique control sets	3			
Number of slice register sites lost to control set restrictions	15	18,224	1%	
Number of bonded IOBs	34	232	14%	
IOB Latches	8			
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	8	248	3%	
Number used as OLOGIC2s	8			
Number used as OSERDES2s	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 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	3.42			

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	Thu May 21 21:29:34 2020	0	49 Warnings (7 new)	3 Infos (1 new)
Translation Report	Current	Thu May 21 21:29:42 2020	0	0	0
Map Report	Current	Thu May 21 21:29:54 2020	0	6 Warnings (6 new)	6 Infos (0 new)
Place and Route Report	Current	Thu May 21 21:30:03 2020	0	0	3 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Thu May 21 21:30:09 2020	0	0	4 Infos (0 new)
Bitgen Report					

Secondary Reports			[-]
Report Name	**Status**	**Generated**	
ISIM Simulator Log	Out of Date	Thu May 21 21:27:41 2020	

Date Generated: 05/21/2020 - 21:19:34

6.2 Implementation Design Summary

```
================================================================================
*                            Design Summary                                    *
================================================================================

Top Level Output File Name         : vending_machine.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                              : 51
#      LUT2                        : 4
#      LUT3                        : 7
#      LUT4                        : 5
#      LUT5                        : 13
#      LUT6                        : 20
#      MUXF7                       : 2
# FlipFlops/Latches                : 18
#      FDC                         : 3
#      LD                          : 15
# Clock Buffers                    : 1
#      BUFG                        : 1
# IO Buffers                       : 17
#      IBUF                        : 11
#      OBUF                        : 6

Device utilization summary:
---------------------------

Selected Device : 6slx16csg324-3


Slice Logic Utilization:
 Number of Slice Registers:              12  out of  18224     0%
 Number of Slice LUTs:                   49  out of   9112     0%
    Number used as Logic:                49  out of   9112     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     49
   Number with an unused Flip Flop:      37  out of     49    75%
   Number with an unused LUT:             0  out of     49     0%
   Number of fully used LUT-FF pairs:    12  out of     49    24%
   Number of unique control sets:         7

IO Utilization:
 Number of IOs:                          17
 Number of IOs:                          17
 Number of bonded IOBs:                  17  out of    232     7%
    IOB Flip Flops/Latches:              6

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               1  out of     16     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


================================================================================
```

6.3 Map Summary

```
Design Summary
--------------
Number of errors:      0
Number of warnings:    6
Slice Logic Utilization:
  Number of Slice Registers:                    12 out of  18,224    1%
    Number used as Flip Flops:                   3
    Number used as Latches:                      9
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                         36 out of   9,112    1%
    Number used as logic:                       36 out of   9,112    1%
      Number using O6 output only:              23
      Number using O5 output only:               0
      Number using O5 and O6:                   13
      Number used as ROM:                        0
    Number used as Memory:                       0 out of   2,176    0%

Slice Logic Distribution:
  Number of occupied Slices:                    12 out of   2,278    1%
  Number of MUXCYs used:                         0 out of   4,556    0%
  Number of LUT Flip Flop pairs used:           37
    Number with an unused Flip Flop:            25 out of      37   67%
    Number with an unused LUT:                   1 out of      37    2%
    Number of fully used LUT-FF pairs:          11 out of      37   29%
    Number of unique control sets:               3
    Number of slice register sites lost
      to control set restrictions:              12 out of  18,224    1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
  one Flip Flop within a slice.  A control set is a unique combination of
  clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.

IO Utilization:
  Number of bonded IOBs:                        17 out of     232    7%
    IOB Latches:                                6

Specific Feature Utilization:
  Number of RAMB16BWERs:                         0 out of      32    0%
  Number of RAMB8BWERs:                          0 out of      64    0%
  Number of BUFIO2/BUFIO2_2CLKs:                 0 out of      32    0%
  Number of BUFIO2FB/BUFIO2FB_2CLKs:             0 out of      32    0%
  Number of RAMB16BWERs:                         0 out of      32    0%
  Number of RAMB8BWERs:                          0 out of      64    0%
  Number of BUFIO2/BUFIO2_2CLKs:                 0 out of      32    0%
  Number of BUFIO2FB/BUFIO2FB_2CLKs:             0 out of      32    0%
  Number of BUFG/BUFGMUXs:                       1 out of      16    6%
    Number used as BUFGs:                        1
    Number used as BUFGMUX:                      0
  Number of DCM/DCM_CLKGENs:                     0 out of       4    0%
  Number of ILOGIC2/ISERDES2s:                   0 out of     248    0%
  Number of IODELAY2/IODRP2/IODRP2_MCBs:         0 out of     248    0%
  Number of OLOGIC2/OSERDES2s:                   6 out of     248    2%
    Number used as OLOGIC2s:                     6
    Number used as OSERDES2s:                    0
  Number of BSCANs:                              0 out of       4    0%
  Number of BUFHs:                               0 out of     128    0%
  Number of BUFPLLs:                             0 out of       8    0%
  Number of BUFPLL_MCBs:                         0 out of       4    0%
  Number of DSP48A1s:                            0 out of      32    0%
  Number of ICAPs:                               0 out of       1    0%
  Number of MCBs:                                0 out of       2    0%
  Number of PCILOGICSEs:                         0 out of       2    0%
  Number of PLL_ADVs:                            0 out of       2    0%
  Number of PMVs:                                0 out of       1    0%
  Number of STARTUPs:                            0 out of       1    0%
  Number of SUSPEND_SYNCs:                       0 out of       1    0%
```

Comment: The implementation requires a lot of flip-flops because they are important for the realization of state assignment for each clock cycle.