**Lab 3 Project Report**

Ethan Keshishian

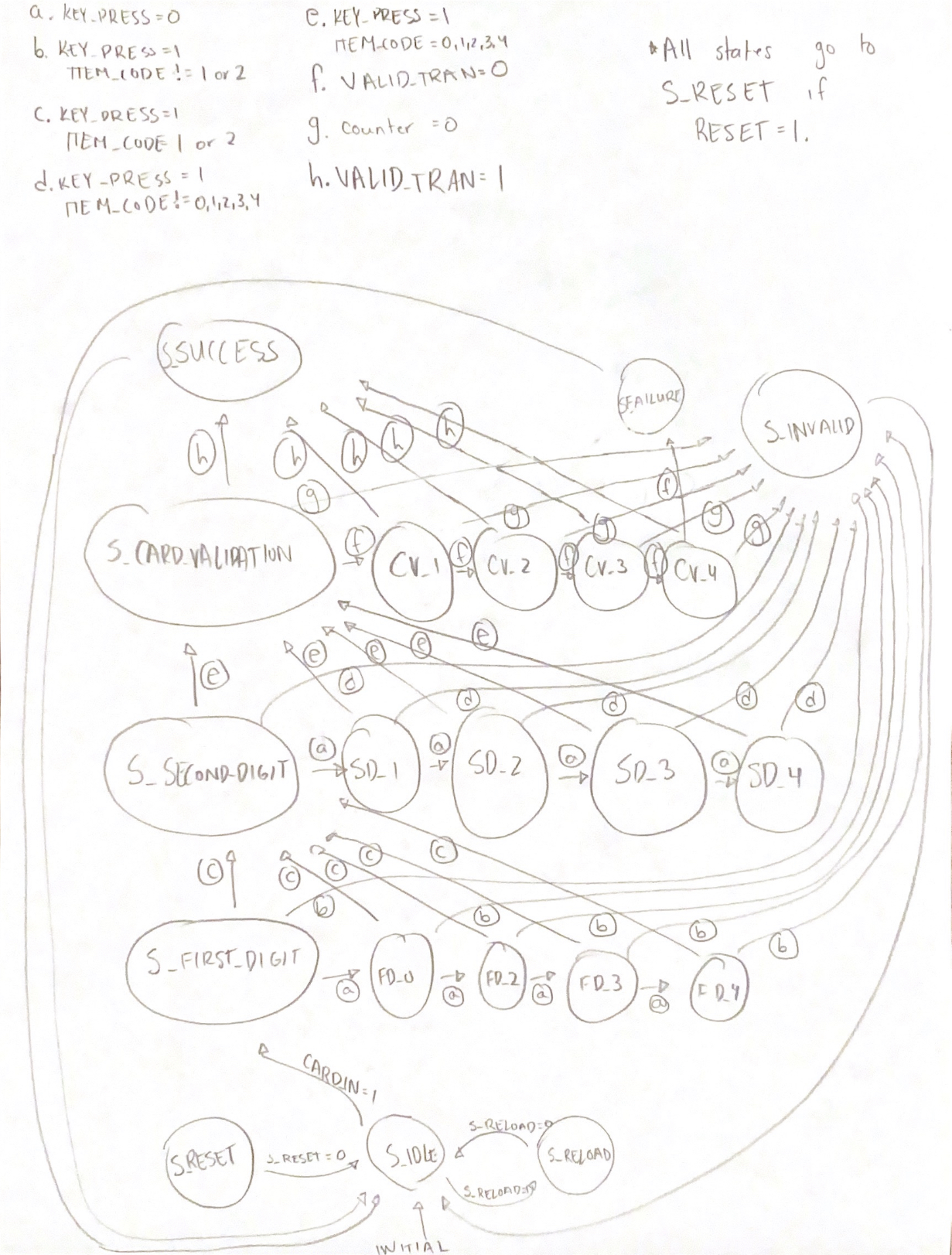
CS M152A Lab1

Introduction:

The goal of Lab 3 is to create a vending machine by designing a finite state machine. This is done with Verilog and the Xilinx software, which helps write the code as well as generate the waveforms. A finite state machine is a machine that is always in a particular state. There are two types of finite state machines: Moore machines, where the next state only depends on the current state, and Mealy machines, where the next state depends on the current state and the input. This design has a module called vending\_machine at the top level, and multiple parameters to define states. These states are handled in an always block with a case. This allows logic for each state to remain separate. Some states include S\_IDLE, S\_RESET, S\_RELOAD, S\_FIRST\_DIGIT, and more. In total, this implementation uses 21 states. This is in line with the example design given in the lab document. The included testbench, testbench\_105422235.v, handles multiple potential use cases for the vending machine, preventing the implementation from being buggy, crashing, or otherwise incorrectly handling input. My final implementation includes waveforms from these test cases.

Module design:

1. States:
2. S\_RESET:
   * Resets all values to their defaults.
   * Refills the vending machine.
   * Remains in reset as long as RESET == 1.
3. S\_RELOAD:
   * Refills the vending machine.
   * Returns to S\_IDLE if RELOAD is no longer 1.
4. S\_IDLE:
   * If RESET ==1, goes to S\_RESET.
   * If RELOAD ==1, goes to S\_RELOAD .
   * Otherwise, waits until CARD\_IN is 1 to move to S\_FIRST\_DIGIT.
5. S\_FIRST\_DIGIT:
   * Checks if KEY\_PRESS ==1. If it is.
     + Reads from ITEM\_CODE and check validity. If valid, move onto S\_SECOND\_DIGIT. Set internal cost accordingly.
     + If not valid, go to S\_INVALID.
   * If KEY\_PRESS isn’t 0, proceed to FD\_0.
6. FD\_0, FD\_2-4:
   * 4 states for handling time between inputs.
   * Implements the same checks as S\_FIRST\_DIGIT. Reads from ITEM\_CODE and check validity.
   * When KEY\_PRESS is 1:
     + If valid, move to S\_SECOND\_DIGIT. Set internal cost accordingly.
     + If invalid, go to S\_INVALID.
   * Otherwise, proceed to the next state in the line.
   * If at the last state, FD\_4, and KEY\_PRESS still isn’t 1, go to S\_INVALID.
7. S\_SECOND\_DIGIT:
   * Checks if KEY\_PRESS ==1. If it is:
     + Reads from ITEM\_CODE and check validity. If valid, move onto S\_CARD\_VALIDATION and set COST.
     + If not valid, go to S\_INVALID.
   * If KEY\_PRESS isn’t 0, proceed to SD\_0.
8. SD\_1-4:
   * 4 states for handling time between inputs
   * Implements the same checks as S\_SECOND \_DIGIT. Reads from ITEM\_CODE and check validity.
   * If KEY\_PRESS is 1:
     + If ITEM\_CODE is valid, move to S\_CARD\_VALIDATION and set COST.
     + If invalid, go to S\_INVALID.
   * Otherwise, proceed to the next state in the line.
   * If at the last state, SD\_4, and KEY\_PRESS still isn’t 1, go to S\_INVALID.
9. S\_CARD\_VALIDATION:
   * Checks for the VALID\_TRAN signal.
     + If VALID\_TRAN is 1, decrements the appropriate counter if possible and moves to S\_SUCCESS.
     + If the counter is already empty, go to S\_INVALID.
   * If VALID\_TRAN isn’t 1, proceed to CV\_1.
10. CV\_1-4:
    * 4 states for handling time until card validation.
    * Implements the same checks as S\_CARD\_VALIDATION. Checks for the VALID\_TRAN signal.
      + If VALID\_TRAN is 1, move to decrements the appropriate counter if possible and moves to S\_SUCCESS.
        - If the counter is already 0, then proceed to S\_INVALID.
      + Otherwise, proceed to the next state in the line.
    * If at the last state, CV\_4, and VALID still isn’t 1, go to S\_FAILURE.
11. S\_SUCCESS:
    * If here, the transaction succeeded.
    * VEND is set to 1, then return to S\_IDLE.
12. S\_INVALID:
    * If here, there was an error, according to the spec.
    * Set INVALID\_SEL to 1, then return to S\_IDLE.
13. S\_FAILURE:
    * If here, card validation took too long, as described by the spec.
    * Set FAILED\_TRAN to 1, then return to S\_IDLE.
14. FSM Diagram (States and transitions):

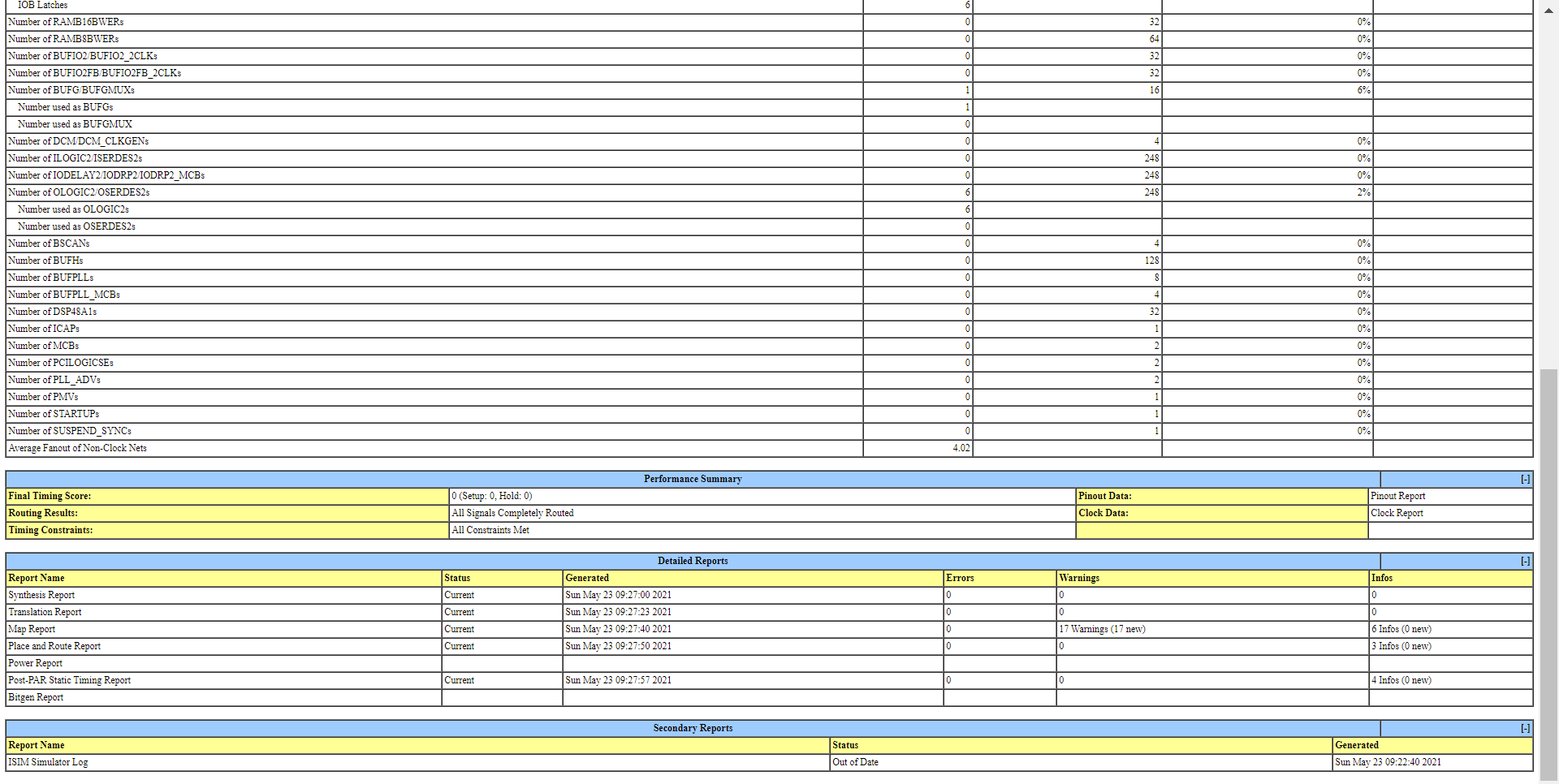
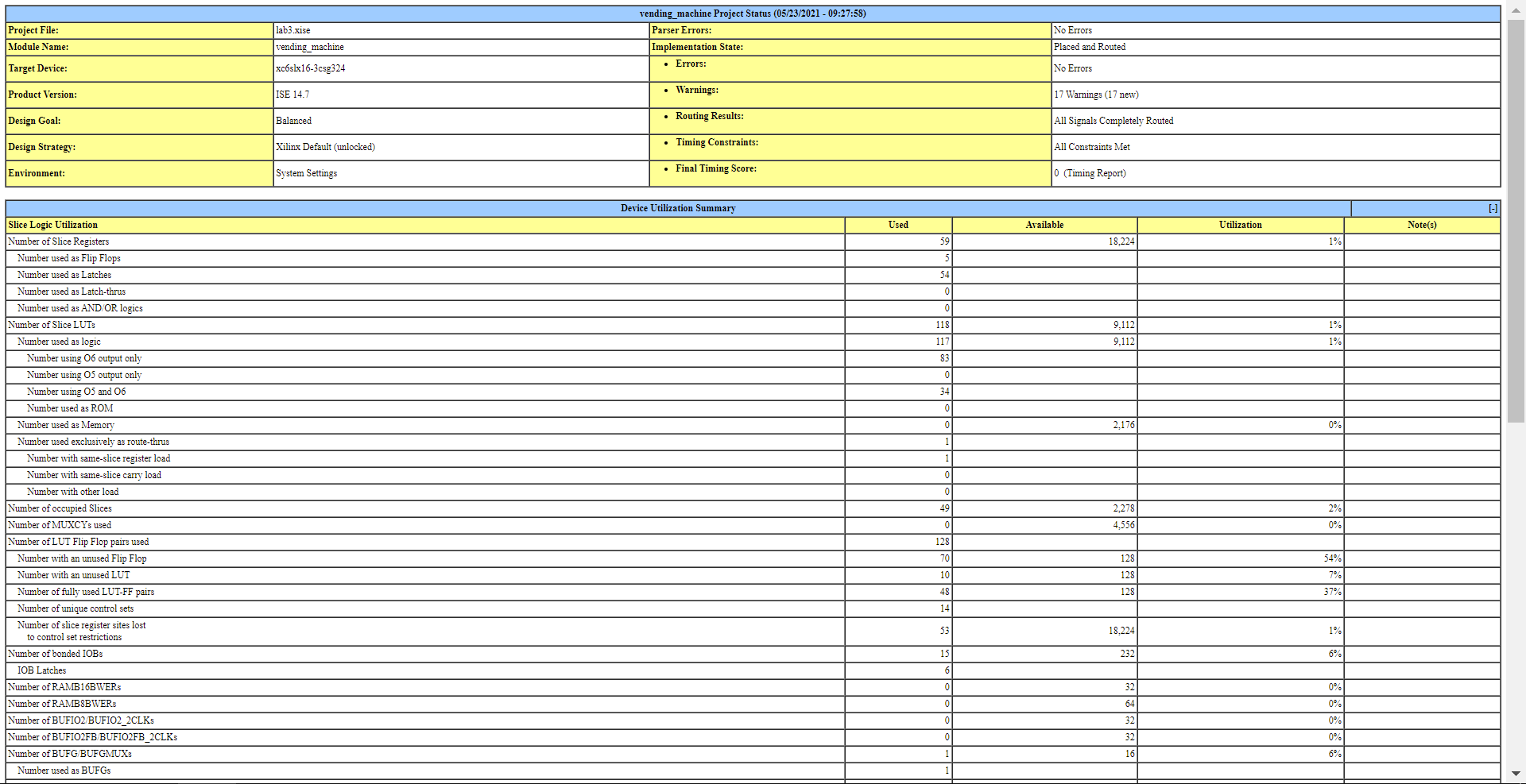


1. Testbench Design:

My testbench handles multiple cases. Here are the implemented cases:

* 1. Successful transaction that begins with the digit 1.
     + This is the case where a cheaper purchase is made, and COST is set to $2.
  2. Successful transaction that begins with the digit 2.
     + This is the case where a pricier purchase is made, and COST is set to $5.
  3. It took too long to press the first digit.
     + Occurs when the FD\_4 state is reached without KEY\_PRESS beings set to 1. The INVALID\_SEL flag should be set.
  4. It took too long to press the second digit.
     + Occurs when the SD\_4 state is reached without KEY\_PRESS beings set to 1. The INVALID\_SEL flag should be set.
  5. Bad input for first digit.
     + In this case, a digit is inputted that is not equal to 1 or 2. The INVALID\_SEL output should be set to 1.
  6. Bad input for second digit.
     + In this case, a digit is inputted that is not between 0 and 4. The INVALID\_SEL output should be set to 1.
  7. Card is declined.
     + In this case, the input is correct, but VALID\_TRAN is never set. The timer should run out, and the FAILED\_TRAN should be set to 1.
  8. Finishing an item from a vending machine.
     + In this case, the same item is purchased 10 times. It accurately reflects this in the counter.
     + An eleventh purchase is made, but in this case, there is no inventory left, and the INVALID\_SEL output should be set to 1. VEND should not be set to 1.
  9. Attempting to reload when not in S\_IDLE.
     + In this case, a reload is attempted when not in idle.
     + RELOAD is set to 1 while the current state is S\_FIRST\_DIGIT, and nothing should happen.
  10. Successful reload.
      + In this case, RELOAD is set to 1 while the current state is S\_IDLE. The reload should reset the counter for the empty item from the previous case.
  11. Resetting halfway through a transaction.
      + In this case, the machine is reset in the middle of the transaction, and the transaction does not complete.
  12. Multiple purchases without removing card.
      + In this case, after each transaction completes, S\_IDLE is quickly exited and the same card is used for another transaction. Expected behavior is to have VEND be set to 1 for each transaction.

1. Waveforms:Graphical user interface

   Description automatically generated with medium confidence
2. Design Summary
3. Map Summary:

Map Report

Sun May 23 10:55:32 2021

Release 14.7 Map P.20131013 (lin64)

Xilinx Mapping Report File for Design 'vending\_machine'

Design Information

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

Command Line : map -intstyle ise -p xc6slx16-csg324-3 -w -logic\_opt off -ol

high -t 1 -xt 0 -register\_duplication off -r 4 -global\_opt off -mt off -ir off

-pr off -lc off -power off -o vending\_machine\_map.ncd vending\_machine.ngd

vending\_machine.pcf

Target Device : xc6slx16

Target Package : csg324

Target Speed : -3

Mapper Version : spartan6 -- $Revision: 1.55 $

Mapped Date : Sun May 23 10:53:43 2021

Design Summary

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

Number of errors: 0

Number of warnings: 17

Slice Logic Utilization:

Number of Slice Registers: 59 out of 18,224 1%

Number used as Flip Flops: 5

Number used as Latches: 54

Number used as Latch-thrus: 0

Number used as AND/OR logics: 0

Number of Slice LUTs: 118 out of 9,112 1%

Number used as logic: 118 out of 9,112 1%

Number using O6 output only: 84

Number using O5 output only: 0

Number using O5 and O6: 34

Number used as ROM: 0

Number used as Memory: 0 out of 2,176 0%

Slice Logic Distribution:

Number of occupied Slices: 45 out of 2,278 1%

Number of MUXCYs used: 0 out of 4,556 0%

Number of LUT Flip Flop pairs used: 128

Number with an unused Flip Flop: 69 out of 128 53%

Number with an unused LUT: 10 out of 128 7%

Number of fully used LUT-FF pairs: 49 out of 128 38%

Number of unique control sets: 15

Number of slice register sites lost

to control set restrictions: 61 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: 15 out of 232 6%

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 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%

Average Fanout of Non-Clock Nets: 4.02

Peak Memory Usage: 762 MB

Total REAL time to MAP completion: 5 secs

Total CPU time to MAP completion: 5 secs

Table of Contents

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

Section 1 - Errors

Section 2 - Warnings

Section 3 - Informational

Section 4 - Removed Logic Summary

Section 5 - Removed Logic

Section 6 - IOB Properties

Section 7 - RPMs

Section 8 - Guide Report

Section 9 - Area Group and Partition Summary

Section 10 - Timing Report

Section 11 - Configuration String Information

Section 12 - Control Set Information

Section 13 - Utilization by Hierarchy

Section 1 - Errors

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

Section 2 - Warnings

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

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_94\_o\_Select\_712\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_46\_o\_Select\_616\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net \_n2384 is sourced by a

combinatorial pin. This is not good design practice. Use the CE pin to

control the loading of data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_22\_o\_Select\_568\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_142\_o\_Select\_808\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_190\_o\_Select\_904\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_118\_o\_Select\_760\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_70\_o\_Select\_664\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_6\_o\_Select\_536\_o is sourced by a combinatorial pin. This

is not good design practice. Use the CE pin to control the loading of data

into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net \_n2385 is sourced by a

combinatorial pin. This is not good design practice. Use the CE pin to

control the loading of data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net \_n2383 is sourced by a

combinatorial pin. This is not good design practice. Use the CE pin to

control the loading of data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_238\_o\_Select\_1000\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_214\_o\_Select\_952\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_166\_o\_Select\_856\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_13\_o\_Select\_550\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_3\_o\_Select\_530\_o is sourced by a combinatorial pin. This

is not good design practice. Use the CE pin to control the loading of data

into the flip-flop.

WARNING:PhysDesignRules:372 - Gated clock. Clock net

current\_state[5]\_PWR\_257\_o\_Select\_1038\_o is sourced by a combinatorial pin.

This is not good design practice. Use the CE pin to control the loading of

data into the flip-flop.

Section 3 - Informational

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

INFO:MapLib:562 - No environment variables are currently set.

INFO:LIT:244 - All of the single ended outputs in this design are using slew

rate limited output drivers. The delay on speed critical single ended outputs

can be dramatically reduced by designating them as fast outputs.

INFO:Pack:1716 - Initializing temperature to 85.000 Celsius. (default - Range:

0.000 to 85.000 Celsius)

INFO:Pack:1720 - Initializing voltage to 1.140 Volts. (default - Range: 1.140 to

1.260 Volts)

INFO:Map:215 - The Interim Design Summary has been generated in the MAP Report

(.mrp).

INFO:Pack:1650 - Map created a placed design.

Section 4 - Removed Logic Summary

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

Section 5 - Removed Logic

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

Section 6 - IOB Properties

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

+---------------------------------------------------------------------------------------------------------------------------------------------------------+

| IOB Name | Type | Direction | IO Standard | Diff | Drive | Slew | Reg (s) | Resistor | IOB |

| | | | | Term | Strength | Rate | | | Delay |

+---------------------------------------------------------------------------------------------------------------------------------------------------------+

| CARD\_IN | IOB | INPUT | LVCMOS25 | | | | | | |

| CLK | IOB | INPUT | LVCMOS25 | | | | | | |

| COST<0> | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | OLATCH | | |

| COST<1> | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | OLATCH | | |

| COST<2> | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | OLATCH | | |

| FAILED\_TRAN | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | OLATCH | | |

| INVALID\_SEL | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | OLATCH | | |

| ITEM\_CODE<0> | IOB | INPUT | LVCMOS25 | | | | | | |

| ITEM\_CODE<1> | IOB | INPUT | LVCMOS25 | | | | | | |

| ITEM\_CODE<2> | IOB | INPUT | LVCMOS25 | | | | | | |

| KEY\_PRESS | IOB | INPUT | LVCMOS25 | | | | | | |

| RELOAD | IOB | INPUT | LVCMOS25 | | | | | | |

| RESET | IOB | INPUT | LVCMOS25 | | | | | | |

| VALID\_TRAN | IOB | INPUT | LVCMOS25 | | | | | | |

| VEND | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | OLATCH | | |

+---------------------------------------------------------------------------------------------------------------------------------------------------------+

Section 7 - RPMs

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

Section 8 - Guide Report

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

Guide not run on this design.

Section 9 - Area Group and Partition Summary

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

Partition Implementation Status

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

No Partitions were found in this design.

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

Area Group Information

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

No area groups were found in this design.

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

Section 10 - Timing Report

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

A logic-level (pre-route) timing report can be generated by using Xilinx static

timing analysis tools, Timing Analyzer (GUI) or TRCE (command line), with the

mapped NCD and PCF files. Please note that this timing report will be generated

using estimated delay information. For accurate numbers, please generate a

timing report with the post Place and Route NCD file.

For more information about the Timing Analyzer, consult the Xilinx Timing

Analyzer Reference Manual; for more information about TRCE, consult the Xilinx

Command Line Tools User Guide "TRACE" chapter.

Section 11 - Configuration String Details

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

Use the "-detail" map option to print out Configuration Strings

Section 12 - Control Set Information

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

Use the "-detail" map option to print out Control Set Information.

Section 13 - Utilization by Hierarchy

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

Use the "-detail" map option to print out the Utilization by Hierarchy section.

Conclusion:

Overall, my implementation uses a top module and states to build the vending machine. It outputs four variables, as described above. The design details indicate that there isn’t too many resources being used, but there are a couple warning that might warrant looking into, though they have no effect on the waveforms, which are displaying accurately. There were multiple difficulties I ran into with the project. I initially wanted to implement a counter instead of using multiple states, but this ran into undefined behavior. Using multiple states resulted in simpler and more readable code. Another thing I learned to use were tasks, since they simplified the code tremendously. Maybe learning about tasks can be added to the course, since I enjoyed using them and found them extremely helpful. Overall, I really enjoyed the project, and found myself learning a lot from overcoming challenges like these.