Final Project: Vending Machine Chip

Fabricio Zuniga

VLSI Design

EE 4513

Table of Contents

Abstract

Vending Machines have served as convenient machines that have been dated back to ancient Greek times. These machines have evolved since its inception and continues to be a staple in convenience. Found all over the world, these automated machines can dispense various products, which includes snacks, beverages, office supplies, and so on. Their position in spontaneous locations and automation make it more likely a person will buy a product out of it.

This project will focus on creating a chip design based on a vending machine. The machine will dispense either a water bottle, sold at 50 cents, or a soda bottle, sold at 100 cents or 1 dollar, based on the amount of coins inserted onto the machine. The machine will also accept different coin denominations in order to reach an amount that is equal to the cost of the water bottle or soda bottle. After dispensing either the water bottle or the soda bottle, if there is any leftover money inputted, then it will return as change. Finite state machines will be used as a design approach in order to count the input of coins and determine when to dispense the water bottle or soda when the amount of coins inserted reaches the selling price of either product.

The project was completed using Xilinx Vivado and Cadence Encounter in order to develop the chip. Most of the work was spent in Vivado as it involved writing the description of the hardware using Verilog, a hardware description language.

Project Overview

In creating a vending machine, the design needed the following: a machine that accepts several coin denominations mostly based on the United States coin denomination, the machine can dispense either a water bottle or a soda bottle (two different products) so as long as the user inserts enough coins to buy the product, properly releasing change, a clock that updates the sequential logic of the machine and a reset. The vending machine can accept the following coin denominations: nickels, dimes, quarters, half dollars and dollars. The water bottle will be sold at fifty cents and the soda bottle will be sold as one hundred cents. This machine will only accept a maximum of two dollars' worth of coins inserted into the machine

Generally, if a user inputs a coin, there will be a count that keeps up with the amount of coins inserted. Once the amount of the inserted coins reaches fifty cents, then the person will be able to dispense the water bottle and if the inserted amount is one hundred cents, then the person will be able to dispense the soda bottle. However, what if instead the user puts in one hundred cents and wants a water bottle? How will the machine know which to dispense for because the machine has enough coins to dispense either one? In this case, there will be two 'enables' specifically to let the user decide what to dispense. The user would be able to select the water bottle by selecting the enable for the water bottle. The water bottle would be dispensed to the user and the machine would also return change of fifty cents back to the user. Essentially, this enable is practically when a person in a vending machine manually selects the item, such as when they dial in the keyword for a product they would want or push a button that corresponds to that item.

Project Design

Taking in consideration for the requirements mentioned in the project overview, I considered developing the vending machine using finite state machines to keep track of the amount of coins being inserted onto the machine. In order to implement this design approach, I had to take it to Xilinx Vivado, where a source code was written. The following was written in Verilog, a hardware description language using Xilinx Vivado. First, The inputs for the machine are the clock that updates the sequential logic, a reset that clears several input/output values, a water_en and a soda_en which allows the user to select either the water bottle or the soda to dispense and a three bit coin_in, which is the coins that are being inserted onto the machine. The outputs are soda, which enables 1 when soda is dispensed, waterbottle enables one when the water bottle is dispensed, a six bit c_return, which is the change returned to the user and a six bit current_state which displays the current state of the finite state machine.

Various local parameters were defined to represent the possible coins for the user to insert onto the machine. There was also another group of local parameters that represented the current state of the machine or the amount of change being returned to the user. All the local parameters were defined to equal a binary value. The local parameters for the coins were in three bits and the ones for the change/current state were in 6 bits. A reg was declared for a six bit next_ state, which determines the next state for the machine.

Before starting to implement the machine, c_return, soda and waterbottle are set to zero. current_state and next_state are set to 'Wait', a local parameter which represents the initial state. Afterwards, an always block is made, which is responsible for determining the next state of the state machine. The sensitivity list contains the current_state and coin_in, meaning that the always block will update whenever those inputs change. The always block begins with a case statement. The case will depend on what the current state is. Within each state, there is another case statement, which determines the next_state based on what coin_in is. Example would be when the current_state is 'Five' for nickel. If coin_in is 'Ten_C' or a dime, then the next_state will update to 'Fifteen.' Because the most expensive product sold in the machine is one hundred cents, our state will only go from our 'Wait' case, to 'Five', all the way to 'Onehundred.'

In the next always block, this block contains the sequential logic that updates the machine at every clock cycle or reset enabled. Using an if statement, if the reset is one, the current_status updates to wait, c_return (change) updates to zero, and both soda and waterbottle return to zero. Otherwise, if reset is zero, then the current_state will update from the next_state.

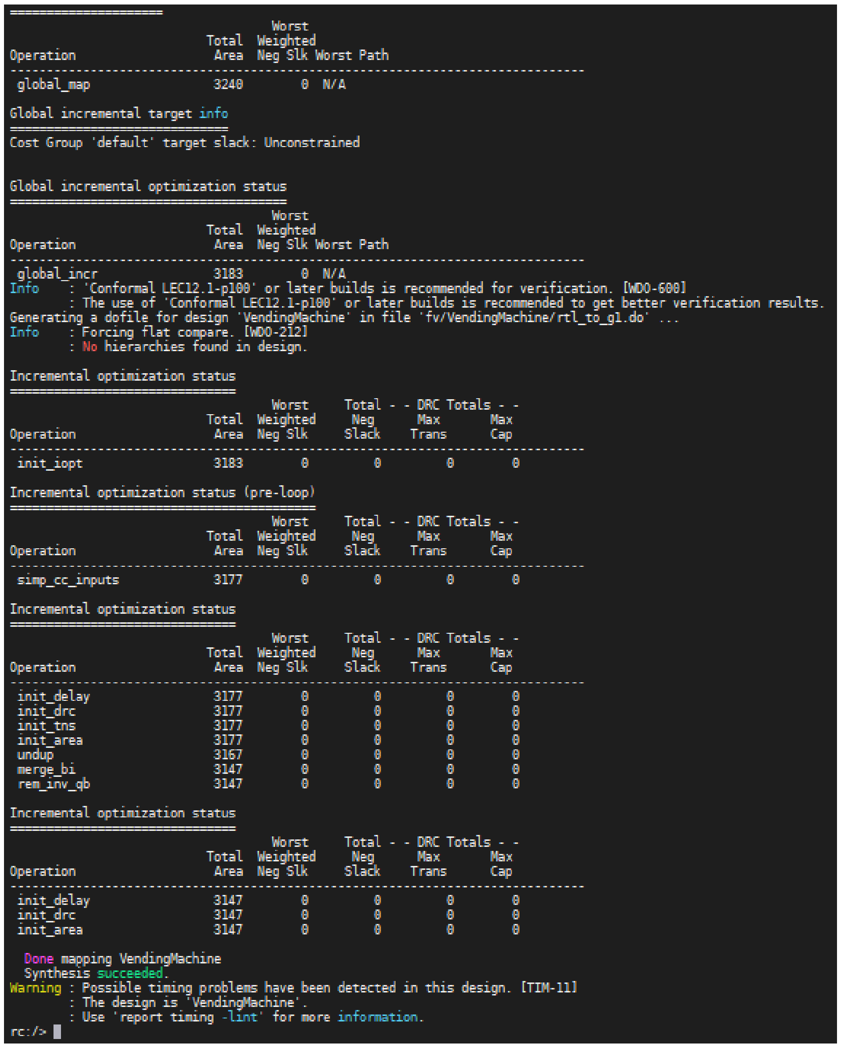
In this last always block, it is purely combinational logic therefore there is nothing in the sensitivity list. This always block is responsible for determining when the water bottle and soda can be dispensed and the amount of changed returned. It begins by using 2 different if statements. On the first if statement, it asks if the current_state is greater than or equal to fifty and water_en is one and soda_en is zero, then it will check another series of if statements that is responsible for both dispensing the water bottle and determining c_return depending on what the current_state is. The first if statement checks if your current_state is at least fifty to have enough coins inserted to dispense the water. To make sure the user wants water, the water_en has to be one and soda_en to be zero. This same logic applies to the next if statement for the soda. If the current_state is at least one hundred and the soda_en is one and water_en is zero, then it will dispense the soda and will determine the change depending on what the current_state is.

That is essentially what my design approach is when writing the source code in Xilinx Vivado. In order to verify its functionality, a test bench was written to test the source code. The test bench will add values for coin_in that equal to the coin denomination values. This will update the state machine at every rising edge of the clock. The reset will be tested to check if the state, soda, waterbottle and both enables reset to zero or 'Wait' for state. After verifying its functionality, the project is operational and can proceed into the next step, where a chip is mapped and routed based on the source code that was created.

Tools and Methodology

The program used to write up the hardware is Xilinx Vivado. With Vivado, the program was written in Verilog, a hardware description language. This is where the actual design begins. The necessary process discussed in prior sections are taken in writing the hardware. Once the source code has been completed, the test bench is written in order to test the validity of the hardware. In order to view the results of the test bench, a simulation waveform must be generated in order to view the values of the inputs and outputs coming from the source file. Once verified, the project will move to Cadence Encounter.

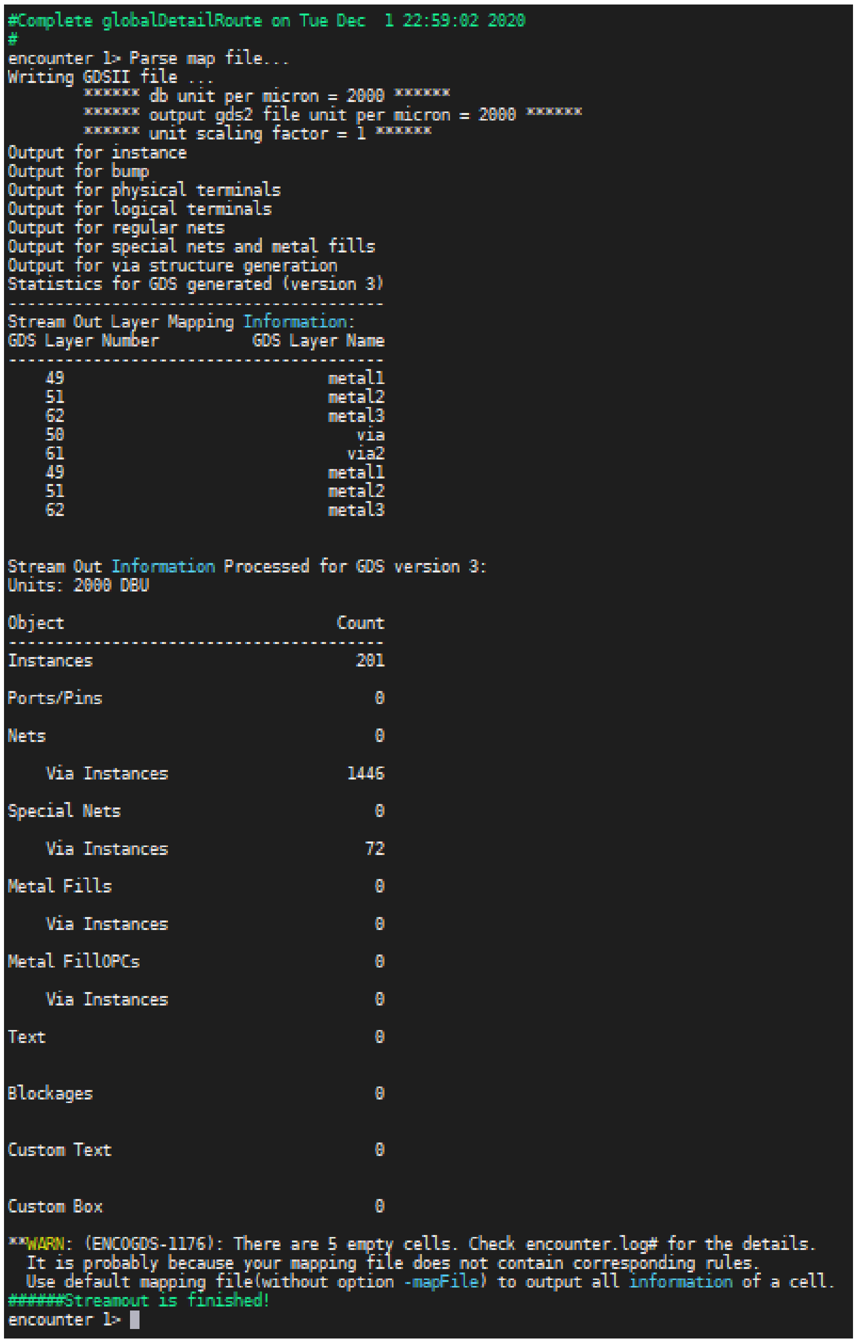
Cadence Encounter is a program that can design an integrated chip by automatically placing and routing various IC components. This program, however, will need to be accessed using MobaXterm, an SSH client or secure shell client that has access to Cadence Encounter from The University of Texas at San Antonio's servers. Several files will be needed in order to map and route the design of the IC. One of these files needed will be the source file that was used to design the vending machine in Vivado. These files need to move into a new folder/directory, which was named 'Finalproject' in UTSA's server, where Encounter will be able to access these files. Once these files have been moved, configure the contents of some of the files, which will let Encounter know where the files are stored in the new directory that was made and where extra design files are stored. Before accessing encounter, you will need to access the terminal and enter 'rc -f design2.g' and 'write_sdc>VendingMachine.sdc' which will create a set of files that Encounter will need in mapping and routing the design.



```
=====================
                                 Worst
                          Total  Weighted
Operation                  Area  Neg Slk Worst Path
-------------------------------------------------------------------------------
 global_map                3240        0  N/A

Global incremental target info
=============================
Cost Group 'default' target slack: Unconstrained


Global incremental optimization status
======================================
                                 Worst
                          Total  Weighted
Operation                  Area  Neg Slk Worst Path
-------------------------------------------------------------------------------
 global_incr               3183        0  N/A
Info    : 'Conformal LEC12.1-p100' or later builds is recommended for verification. [WDO-600]
        : The use of 'Conformal LEC12.1-p100' or later builds is recommended to get better verification results.
Generating a dofile for design 'VendingMachine' in file 'fv/VendingMachine/rtl_to_g1.do' ...
Info    : Forcing flat compare. [WDO-212]
        : No hierarchies found in design.

Incremental optimization status
===============================
                                 Worst     Total - - DRC Totals - -
                          Total  Weighted   Neg      Max       Max
Operation                  Area  Neg Slk    Slack    Trans     Cap
-------------------------------------------------------------------------------
 init_iopt                 3183        0         0         0        0

Incremental optimization status (pre-loop)
==========================================
                                 Worst     Total - - DRC Totals - -
                          Total  Weighted   Neg      Max       Max
Operation                  Area  Neg Slk    Slack    Trans     Cap
-------------------------------------------------------------------------------
 simp_cc_inputs            3177        0         0         0        0

Incremental optimization status
===============================
                                 Worst     Total - - DRC Totals - -
                          Total  Weighted   Neg      Max       Max
Operation                  Area  Neg Slk    Slack    Trans     Cap
-------------------------------------------------------------------------------
 init_delay                3177        0         0         0        0
 init_drc                  3177        0         0         0        0
 init_tns                  3177        0         0         0        0
 init_area                 3177        0         0         0        0
 undup                     3167        0         0         0        0
 merge_bi                  3147        0         0         0        0
 rem_inv_qb                3147        0         0         0        0

Incremental optimization status
===============================
                                 Worst     Total - - DRC Totals - -
                          Total  Weighted   Neg      Max       Max
Operation                  Area  Neg Slk    Slack    Trans     Cap
-------------------------------------------------------------------------------
 init_delay                3147        0         0         0        0
 init_drc                  3147        0         0         0        0
 init_area                 3147        0         0         0        0

  Done mapping VendingMachine
  Synthesis succeeded.
Warning : Possible timing problems have been detected in this design. [TIM-11]
        : The design is 'VendingMachine'.
        : Use 'report timing -lint' for more information.
rc:/>
```

Figure 1. Global Mapping results after executing design2.g command

Encounter can be launched and will be responsible in mapping, routing, placing the ground and power and placing transistors to create the logic behind the Vivado source code. Once the process is complete, the design can be tested to make sure it is able to function properly. The tests that were performed were timing, connectivity, geometry and power tests. If all tests display satisfactory results, then the final design result can be exported using a GDS2/OASIS layout.



```
#Complete globalDetailRoute on Tue Dec  1 22:59:02 2020
#
encounter 1> Parse map file...
Writing GDSII file ...
        ****** db unit per micron = 2000 ******
        ****** output gds2 file unit per micron = 2000 ******
        ****** unit scaling factor = 1 ******
Output for instance
Output for bump
Output for physical terminals
Output for logical terminals
Output for regular nets
Output for special nets and metal fills
Output for via structure generation
Statistics for GDS generated (version 3)
----------------------------------------
Stream Out Layer Mapping Information:
GDS Layer Number          GDS Layer Name
----------------------------------------
    49                              metal1
    51                              metal2
    62                              metal3
    50                                 via
    61                                via2
    49                              metal1
    51                              metal2
    62                              metal3


Stream Out Information Processed for GDS version 3:
Units: 2000 DBU

Object                             Count
----------------------------------------
Instances                            201

Ports/Pins                             0

Nets                                   0

    Via Instances                   1446

Special Nets                           0

    Via Instances                     72

Metal Fills                            0

    Via Instances                      0

Metal FillOPCs                         0

    Via Instances                      0

Text                                   0


Blockages                              0


Custom Text                            0


Custom Box                             0

**WARN: (ENCOGDS-1176): There are 5 empty cells. Check encounter.log# for the details.
  It is probably because your mapping file does not contain corresponding rules.
  Use default mapping file(without option -mapFile) to output all information of a cell.
######Streamout is finished!
encounter 1>
```

Figure 2.GDS2/OASIS Streamout results

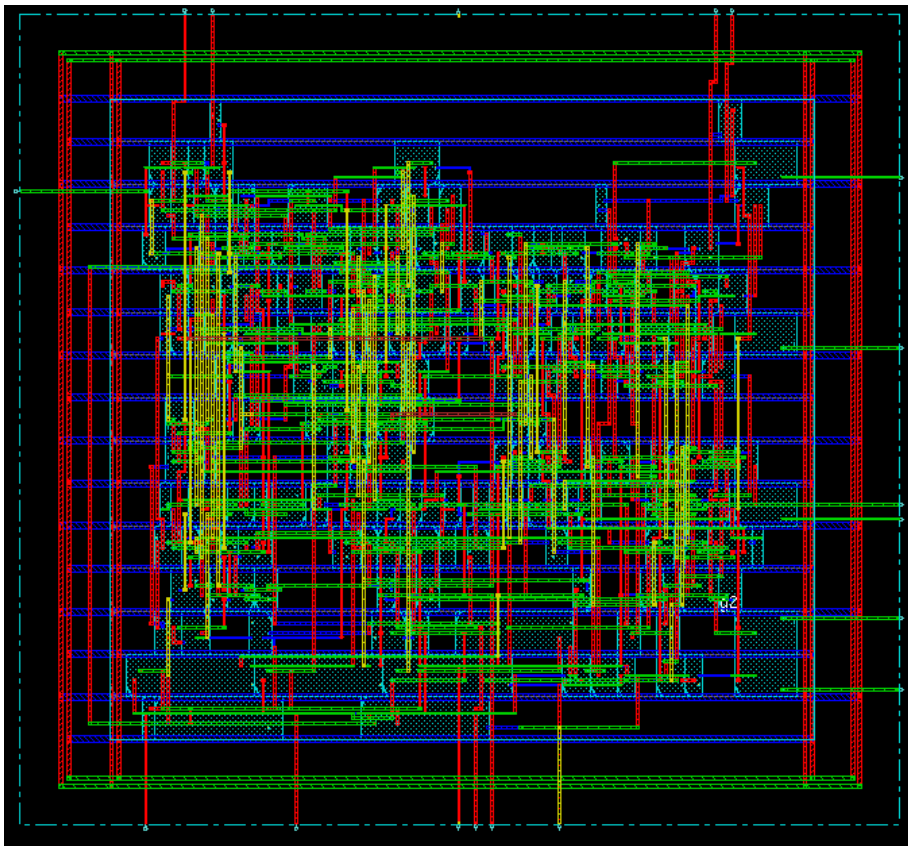


Figure 3. Chip design layout from Encounter

Additionally, schematics were generated from both software. Both schematics look very different from each other

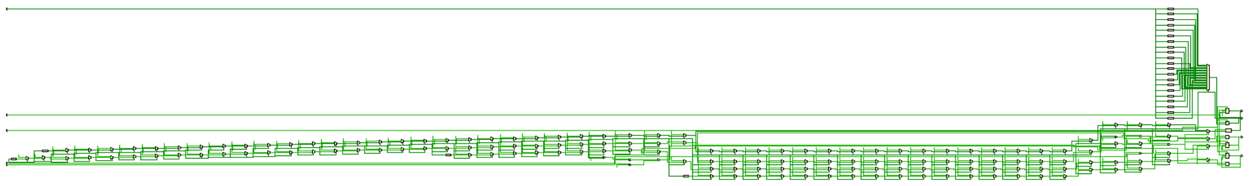


Figure 4. Xilinx Vivado Schematic

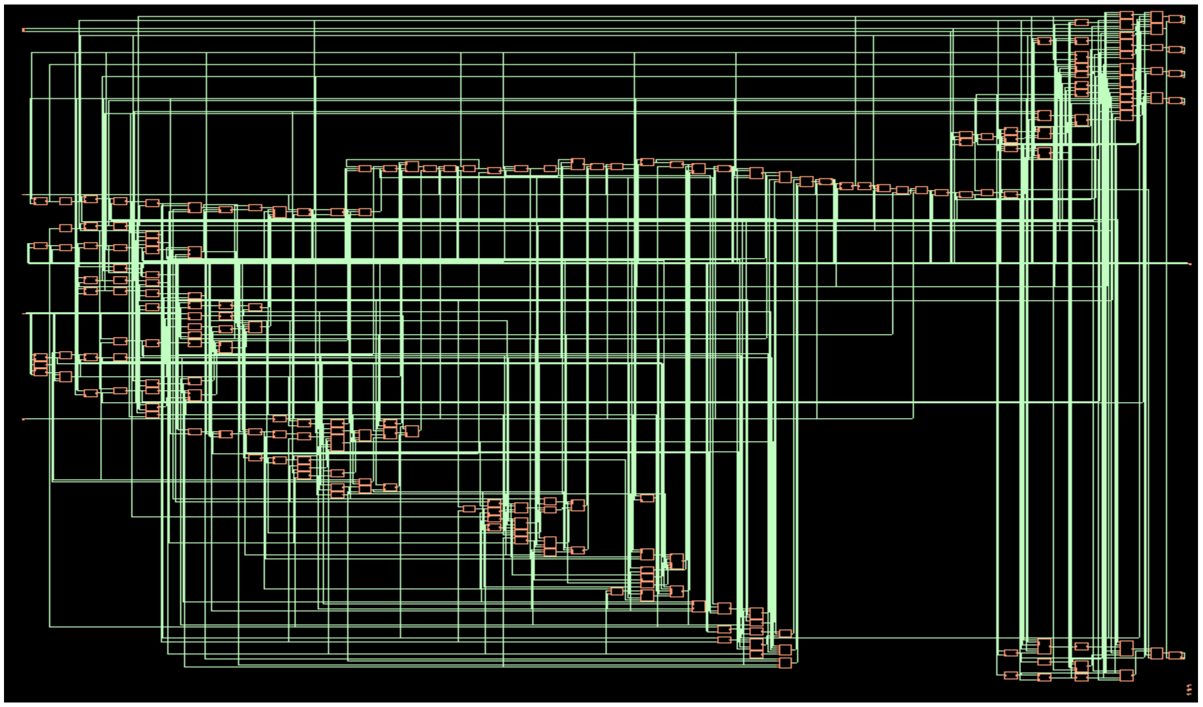


Figure 5. Schematic from Encounter

Results and Discussion

Starting in Xilinx Vivado, a simulation waveform was produced that tests the functionality of the source code. It will test several situations: First, the user inputs the half dollar and dispenses a water bottle. Second, the user inputs a dollar coin and dispenses a soda. Third, the user dispenses a dollar and a quarter and dispenses a water bottle and change of seventy-five cents. Fourth, the user inputs two dollars and dispenses a soda. Fifth, the user inputs two nickels and tries to dispense soda, but it should output a one for soda. Sixth, the user inputs four quarters and dispenses a soda. Finally, for the seventh situation, the user inputs five dimes and dispenses a water bottle. Based on my results the waveform shows that the output values for my present state, soda, water bottle and change are all correct for each situation. Their values match for what it is supposed to be, and their values properly update at every rising edge of the clock.

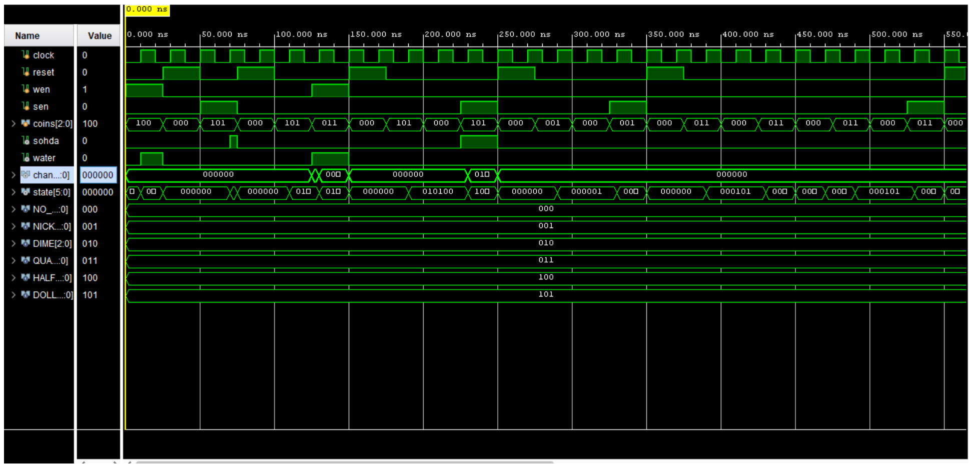


Figure 6a. Simulation Waveform

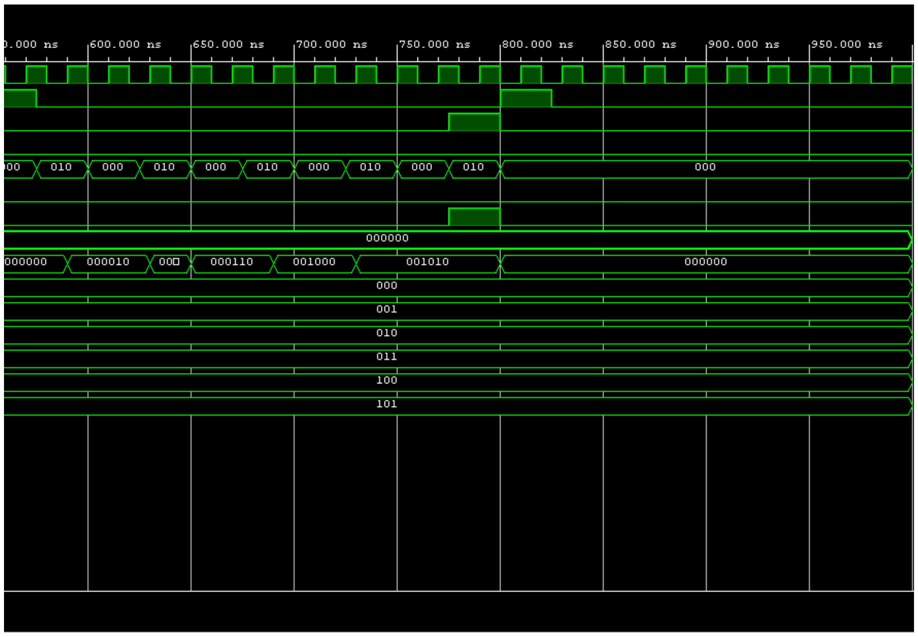
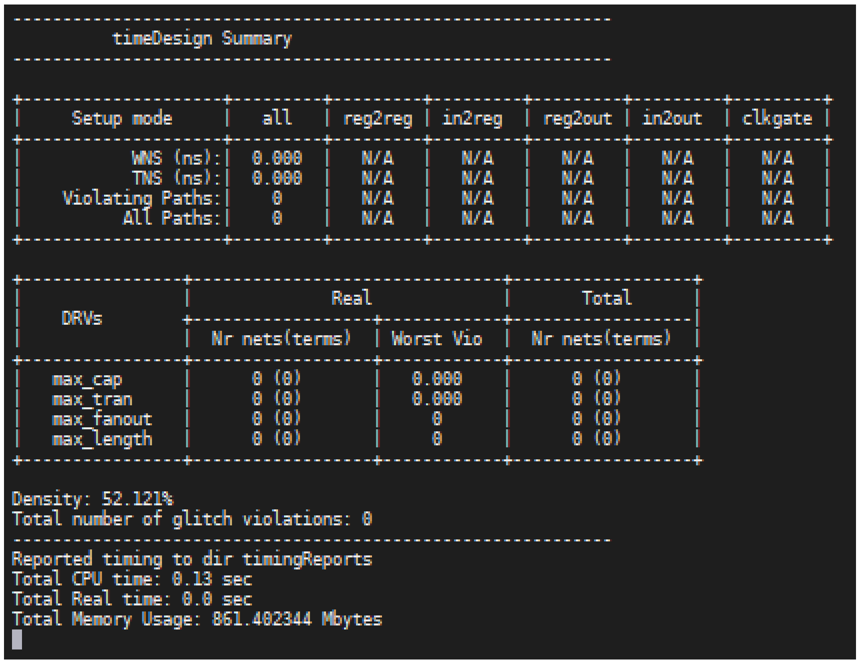


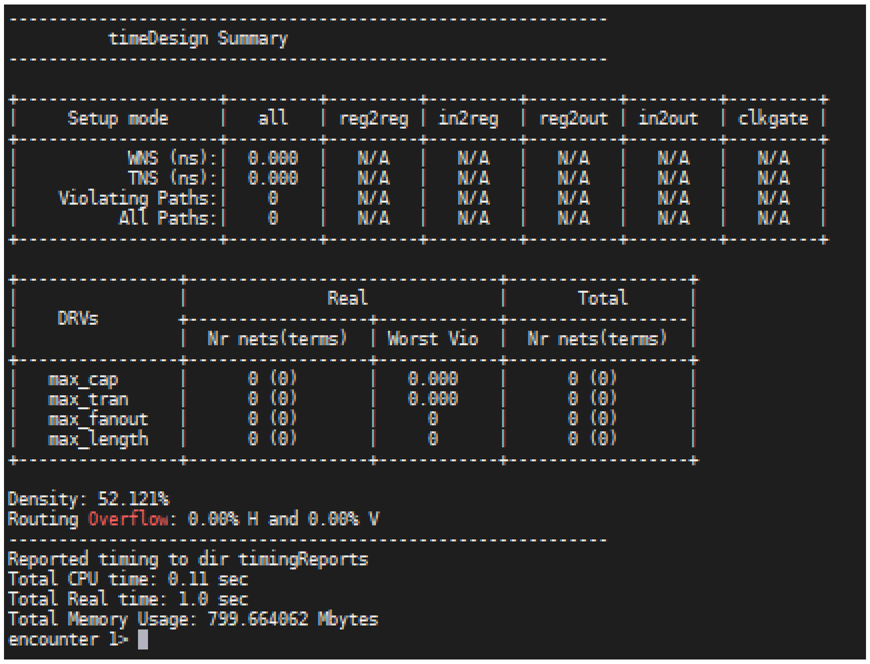
Figure 6b. Simulation Waveform

In Cadence Encounter, there were several tests performed. Two timing tests were performed. The first timing test was done before nano routing and the second test after nano routing. The second timing test shows faster CPU time and lower memory usage. A power test was performed, and no abnormalities were detected. Finally, both the geometry test and connectivity test showed no warnings or violations.



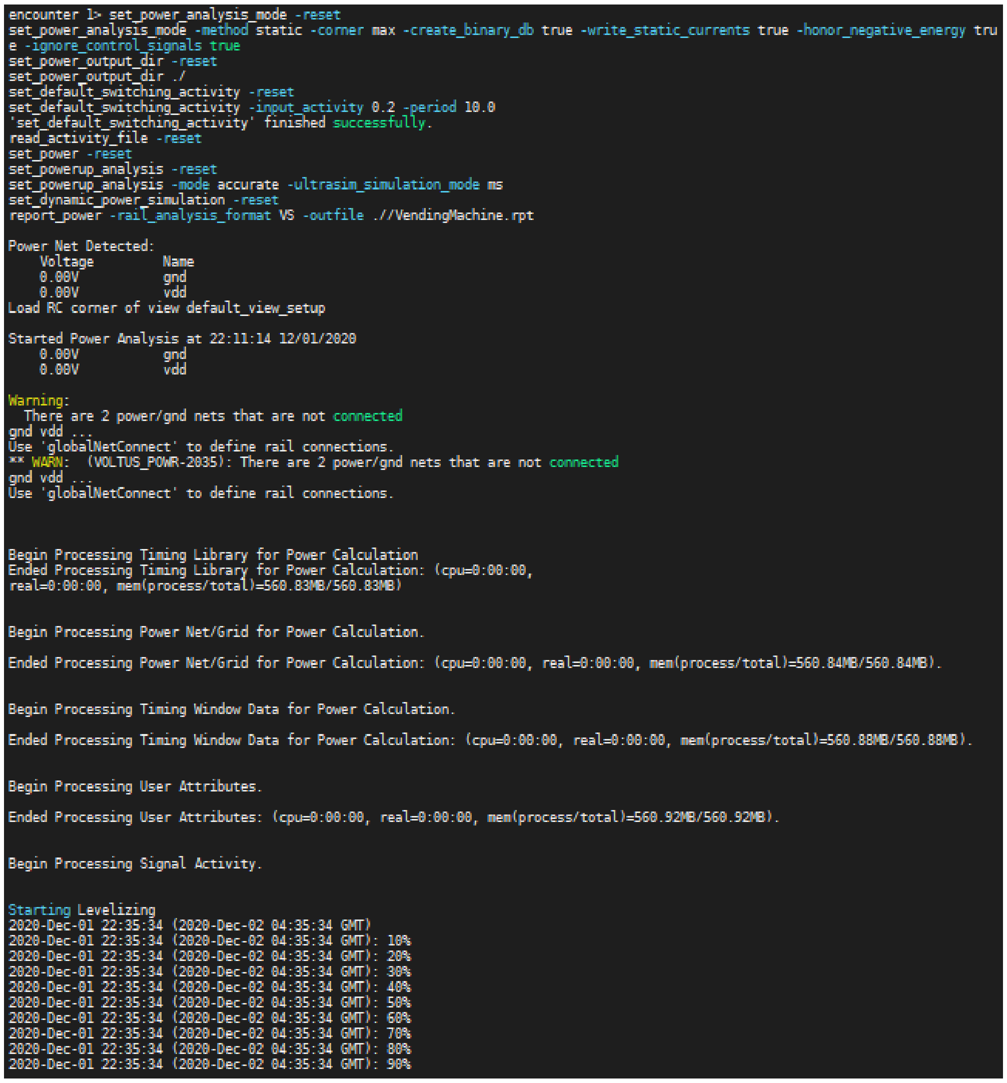
```
------------------------------------------------------------
         timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.000  |   N/A   |   N/A   |   N/A   |   N/A   |   N/A   |
|           TNS (ns):|  0.000  |   N/A   |   N/A   |   N/A   |   N/A   |   N/A   |
|    Violating Paths:|    0    |   N/A   |   N/A   |   N/A   |   N/A   |   N/A   |
|          All Paths:|    0    |   N/A   |   N/A   |   N/A   |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |     0 (0)        |   0.000    |     0 (0)        |
|   max_tran     |     0 (0)        |   0.000    |     0 (0)        |
|   max_fanout   |     0 (0)        |     0      |     0 (0)        |
|   max_length   |     0 (0)        |     0      |     0 (0)        |
+----------------+------------------+------------+------------------+

Density: 52.121%
Total number of glitch violations: 0
------------------------------------------------------------
Reported timing to dir timingReports
Total CPU time: 0.13 sec
Total Real time: 0.0 sec
Total Memory Usage: 861.402344 Mbytes
```

Figure 7. Time report before nano routing

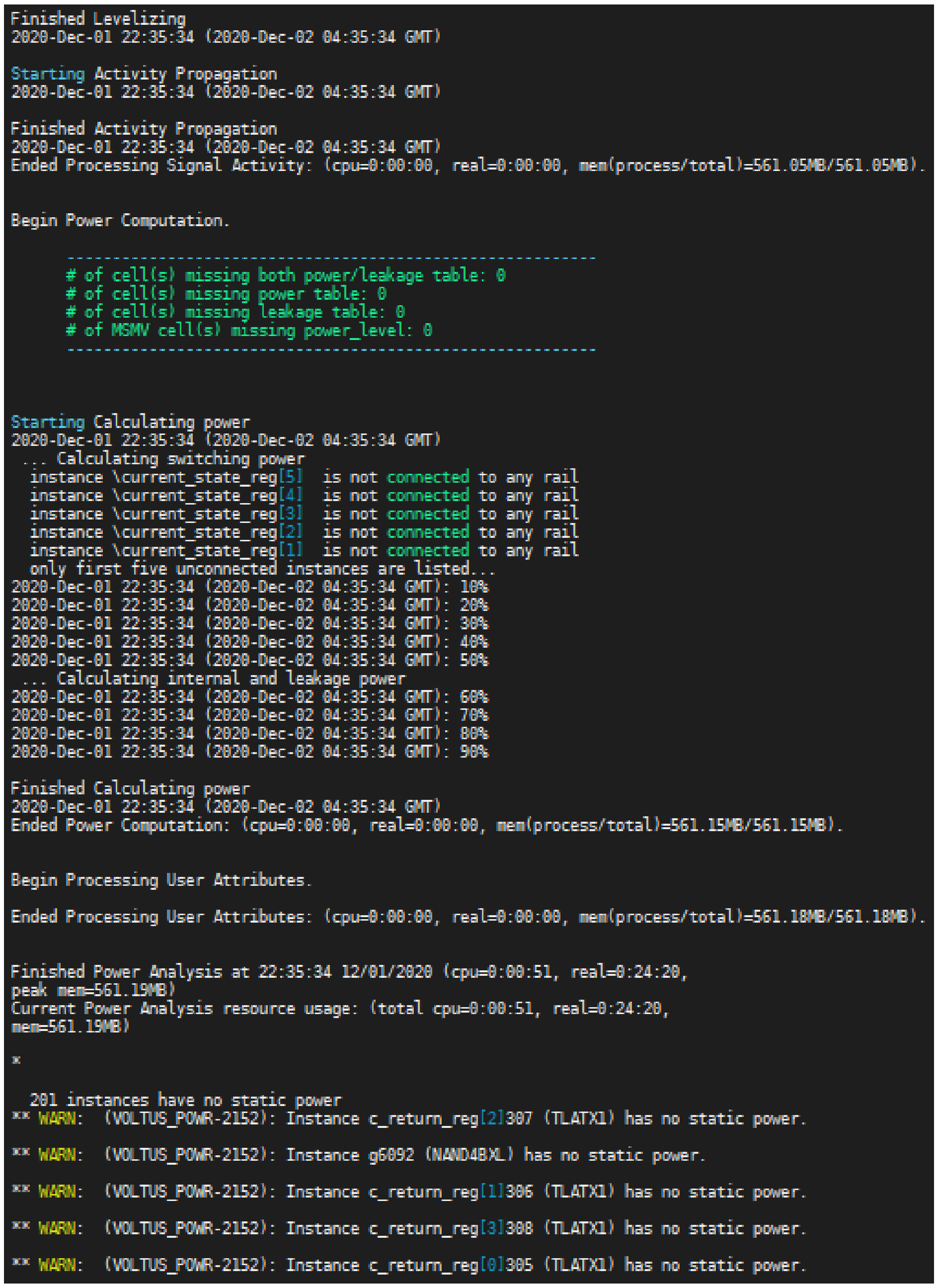


```
------------------------------------------------------------
         timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  0.000  |   N/A   |   N/A   |   N/A   |   N/A   |   N/A   |
|           TNS (ns):|  0.000  |   N/A   |   N/A   |   N/A   |   N/A   |   N/A   |
|    Violating Paths:|    0    |   N/A   |   N/A   |   N/A   |   N/A   |   N/A   |
|          All Paths:|    0    |   N/A   |   N/A   |   N/A   |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |     0 (0)        |   0.000    |     0 (0)        |
|   max_tran     |     0 (0)        |   0.000    |     0 (0)        |
|   max_fanout   |     0 (0)        |     0      |     0 (0)        |
|   max_length   |     0 (0)        |     0      |     0 (0)        |
+----------------+------------------+------------+------------------+

Density: 52.121%
Routing Overflow: 0.00% H and 0.00% V
------------------------------------------------------------
Reported timing to dir timingReports
Total CPU time: 0.11 sec
Total Real time: 1.0 sec
Total Memory Usage: 799.664062 Mbytes
encounter 1>
```

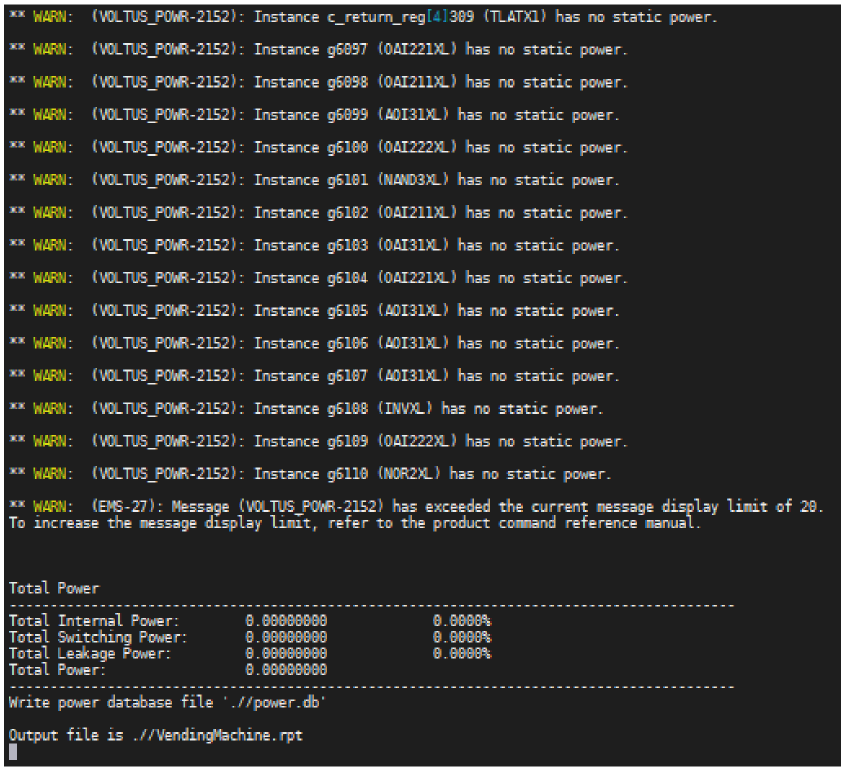
Figure 8. Time report after nano routing



```
encounter 1> set_power_analysis_mode -reset
set_power_analysis_mode -method static -corner max -create_binary_db true -write_static_currents true -honor_negative_energy tru
e -ignore_control_signals true
set_power_output_dir -reset
set_power_output_dir ./
set_default_switching_activity -reset
set_default_switching_activity -input_activity 0.2 -period 10.0
'set_default_switching_activity' finished successfully.
read_activity_file -reset
set_power -reset
set_powerup_analysis -reset
set_powerup_analysis -mode accurate -ultrasim_simulation_mode ms
set_dynamic_power_simulation -reset
report_power -rail_analysis_format VS -outfile .//VendingMachine.rpt

Power Net Detected:
    Voltage         Name
    0.00V           gnd
    0.00V           vdd
Load RC corner of view default_view_setup

Started Power Analysis at 22:11:14 12/01/2020
    0.00V           gnd
    0.00V           vdd

Warning:
  There are 2 power/gnd nets that are not connected
gnd vdd ...
Use 'globalNetConnect' to define rail connections.
** WARN:  (VOLTUS_POWR-2035): There are 2 power/gnd nets that are not connected
gnd vdd ...
Use 'globalNetConnect' to define rail connections.



Begin Processing Timing Library for Power Calculation
Ended Processing Timing Library for Power Calculation: (cpu=0:00:00,
real=0:00:00, mem(process/total)=560.83MB/560.83MB)


Begin Processing Power Net/Grid for Power Calculation.

Ended Processing Power Net/Grid for Power Calculation: (cpu=0:00:00, real=0:00:00, mem(process/total)=560.84MB/560.84MB).


Begin Processing Timing Window Data for Power Calculation.

Ended Processing Timing Window Data for Power Calculation: (cpu=0:00:00, real=0:00:00, mem(process/total)=560.88MB/560.88MB).


Begin Processing User Attributes.

Ended Processing User Attributes: (cpu=0:00:00, real=0:00:00, mem(process/total)=560.92MB/560.92MB).


Begin Processing Signal Activity.


Starting Levelizing
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT)
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 10%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 20%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 30%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 40%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 50%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 60%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 70%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 80%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 90%
```

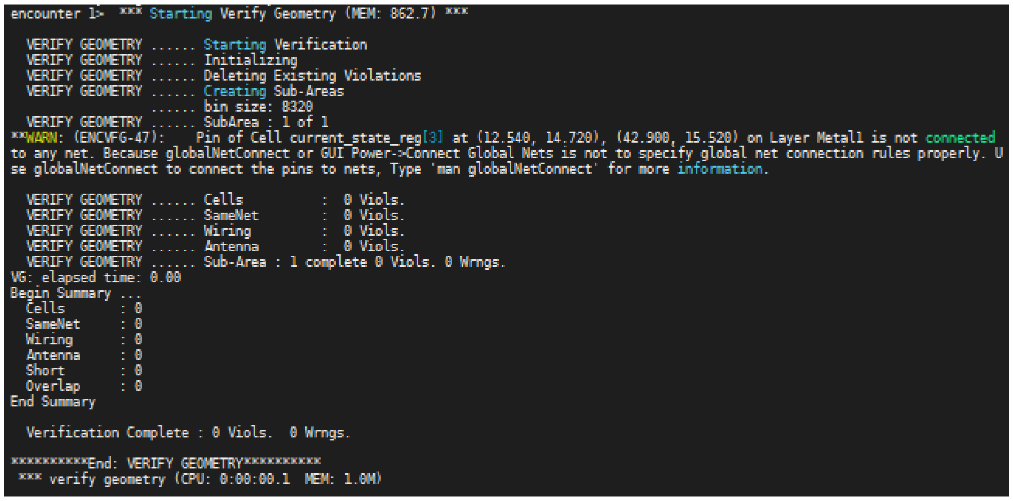


```
Finished Levelizing
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT)

Starting Activity Propagation
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT)

Finished Activity Propagation
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT)
Ended Processing Signal Activity: (cpu=0:00:00, real=0:00:00, mem(process/total)=561.05MB/561.05MB).


Begin Power Computation.

      ----------------------------------------------------------
      # of cell(s) missing both power/leakage table: 0
      # of cell(s) missing power table: 0
      # of cell(s) missing leakage table: 0
      # of MSMV cell(s) missing power_level: 0
      ----------------------------------------------------------



Starting Calculating power
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT)
 ... Calculating switching power
  instance \current_state_reg[5]  is not connected to any rail
  instance \current_state_reg[4]  is not connected to any rail
  instance \current_state_reg[3]  is not connected to any rail
  instance \current_state_reg[2]  is not connected to any rail
  instance \current_state_reg[1]  is not connected to any rail
  only first five unconnected instances are listed...
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 10%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 20%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 30%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 40%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 50%
 ... Calculating internal and leakage power
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 60%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 70%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 80%
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT): 90%

Finished Calculating power
2020-Dec-01 22:35:34 (2020-Dec-02 04:35:34 GMT)
Ended Power Computation: (cpu=0:00:00, real=0:00:00, mem(process/total)=561.15MB/561.15MB).


Begin Processing User Attributes.

Ended Processing User Attributes: (cpu=0:00:00, real=0:00:00, mem(process/total)=561.18MB/561.18MB).


Finished Power Analysis at 22:35:34 12/01/2020 (cpu=0:00:51, real=0:24:20,
peak mem=561.19MB)
Current Power Analysis resource usage: (total cpu=0:00:51, real=0:24:20,
mem=561.19MB)

*

  201 instances have no static power
** WARN:  (VOLTUS_POWR-2152): Instance c_return_reg[2]307 (TLATX1) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6092 (NAND4BXL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance c_return_reg[1]306 (TLATX1) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance c_return_reg[3]308 (TLATX1) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance c_return_reg[0]305 (TLATX1) has no static power.
```



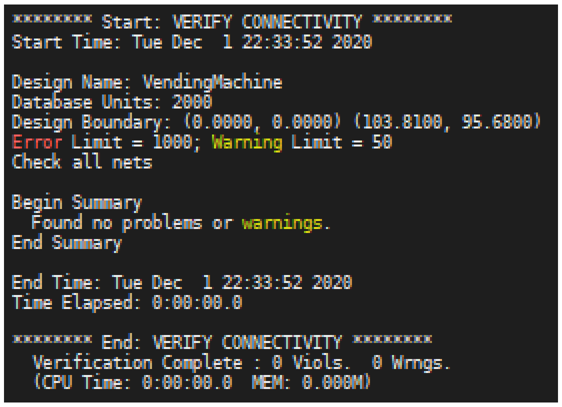
```
** WARN:  (VOLTUS_POWR-2152): Instance c_return_reg[4]309 (TLATX1) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6097 (OAI221XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6098 (OAI211XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6099 (AOI31XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6100 (OAI222XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6101 (NAND3XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6102 (OAI211XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6103 (OAI31XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6104 (OAI221XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6105 (AOI31XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6106 (AOI31XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6107 (AOI31XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6108 (INVXL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6109 (OAI222XL) has no static power.

** WARN:  (VOLTUS_POWR-2152): Instance g6110 (NOR2XL) has no static power.

** WARN:  (EMS-27): Message (VOLTUS_POWR-2152) has exceeded the current message display limit of 20.
To increase the message display limit, refer to the product command reference manual.


Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:       0.00000000            0.0000%
Total Switching Power:      0.00000000            0.0000%
Total Leakage Power:        0.00000000            0.0000%
Total Power:                0.00000000
-----------------------------------------------------------------------------------------
Write power database file './/power.db'

Output file is .//VendingMachine.rpt
```

Figure 9 (a,b,c.) Power Analysis report



```
encounter 1>  *** Starting Verify Geometry (MEM: 862.7) ***

  VERIFY GEOMETRY ...... Starting Verification
  VERIFY GEOMETRY ...... Initializing
  VERIFY GEOMETRY ...... Deleting Existing Violations
  VERIFY GEOMETRY ...... Creating Sub-Areas
                  ...... bin size: 8320
  VERIFY GEOMETRY ...... SubArea : 1 of 1
**WARN: (ENCVFG-47):    Pin of Cell current_state_reg[3] at (12.540, 14.720), (42.900, 15.520) on Layer Metal1 is not connected
to any net. Because globalNetConnect or GUI Power->Connect Global Nets is not to specify global net connection rules properly. U
se globalNetConnect to connect the pins to nets, Type 'man globalNetConnect' for more information.

  VERIFY GEOMETRY ...... Cells          :  0 Viols.
  VERIFY GEOMETRY ...... SameNet        :  0 Viols.
  VERIFY GEOMETRY ...... Wiring         :  0 Viols.
  VERIFY GEOMETRY ...... Antenna        :  0 Viols.
  VERIFY GEOMETRY ...... Sub-Area : 1 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 0.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:00.1  MEM: 1.0M)
```

Figure 10. Geometry analysis report



```
******** Start: VERIFY CONNECTIVITY ********
Start Time: Tue Dec  1 22:33:52 2020

Design Name: VendingMachine
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (103.8100, 95.6800)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Tue Dec  1 22:33:52 2020
Time Elapsed: 0:00:00.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.0  MEM: 0.000M)
```

Figure 11. Connectivity report

Overall, the design works properly based on these results, but there were some timing issues when the design was synthesized in Vivado. The timing issues were mostly based on how the sequential always block was designed. I could have modified the always block to develop faster hardware, but it would have changed the functionality of the hardware, which could have affected getting correct outputs.

Conclusion/Summary

There were several methods to implement a Vending Machine in Verilog but using 3 different always blocks where one was the sequential logic, one was the next state logic and the other was the combinational logic was the way to go in terms of designing. It did take a while writing the source file to get down all the states that were possibly needed for a machine that can take at most two dollars. If there was more time available, I might have added pennies as a coin denomination for the user to input and added more products to dispense from the machine.

My previous experience using Verilog and Vivado in Computer Organization and Architecture and the previous labs in this course, including example resources of finite state machines in YouTube and ASIC-World, helped me in writing the source code for the project by guiding me how to set up a state machine, understanding how Verilog works as a HDL and understand how Vivado works as a integrated development software for Verilog HDL.

Overall, it was an interesting project that I tackled on. It helped me refresh my understanding of finite state machines. Also, knowing that the process that was taken in this project is part of the process taken in the development and fabrication of integrated chips, which will be very useful for me as I am trying to pursue a career working closely with chip or VLSI development.

References

Videos

Finite State Machines in Verilog

https://www.youtube.com/watch?v=f7DSYhEGXFA

Websites

How Do I Write FSM in Verilog?

https://www.asic-world.com/tidbits/verilog_fsm.html

Other

Lab 8 for VLSI Design

Source Code (VendingMachine.v)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Fabricio Zuniga
// Final Project
// VLSI Design
// EE 4513
// Fall 2020
//////////////////////////////////////////////////////////////////////////////////

module VendingMachine //this VM can only accept upto two hundred cents or 2 dollars
(
    input clk, rst, water_en, soda_en, //clock & reset. water_en and soda_en are enable bits that
allow user to choose either soda or water bottle if they have enough money for whichever
product
    input [2:0] coin_in, //coins from user
    output reg soda, //1.00 dollar
    output reg waterbottle,//50 cents
    output reg [5:0] c_return, //change
    output reg [5:0] current_state
);
// all possible states for the state diagram, including change values
localparam Wait = 6'b000000, Five = 6'b000001, Ten = 6'b000010, Fifteen = 6'b000011,
           Twenty = 6'b000100, Twentyfive = 6'b000101, Thirty = 6'b000110,
           Thirtyfive = 6'b000111, Forty = 6'b001000, Fortyfive = 6'b001001, Fifty = 6'b001010,
           Fiftyfive = 6'b001011, Sixty = 6'b001100, Sixtyfive = 6'b001101, Seventy = 6'b001110,
           Seventyfive = 6'b001111, Eighty = 6'b010000, Eightyfive = 6'b010001, Ninety =
6'b010010,
        Ninetyfive = 6'b010011, Onehundred = 6'b010100, Onehundredfive = 6'b010101,
Onehundredten = 6'b010110,
        Onehundredfifteen = 6'b010111, Onehundredtwenty = 6'b011000, Onehundredtwentyfive
= 6'b011001,
        Onehundredthirty = 6'b011010, Onehundredthirtyfive = 6'b011011, Onehundredforty =
6'b011100,
        Onehundredfortyfive = 6'b011101, Onehundredfifty = 6'b011101, Onehundredfiftyfive =
6'b011110,
        Onehundredsixty = 6'b011111, Onehundredsixtyfive = 6'b100000, Onehundredseventy =
6'b100001,
        Onehundredseventyfive = 6'b100010, Onehundredeighty =
6'b100011,Onehundredeightyfive = 6'b100100,
        Onehundredninety = 6'b100101, Onehundredninetyfive = 6'b100110, Twohundred =
6'b100111;

//all possible coins to input for coin_in
localparam  NO_C = 3'b000, FIVE_C  = 3'b001, TEN_C = 3'b010,
         TWENTYFIVE_C = 3'b011, HALFDOLLAR_C = 3'b100,
         DOLLAR_C = 3'b101;

reg [5:0] next_state; //next state


initial //initialize variables
begin
   soda = 0;
   waterbottle = 0;
   c_return = 4'b0000; // will return value based on parameter
   current_state = Wait;
   next_state = Wait;
end


always @(current_state or coin_in) //always block to determine next state of SM
begin
   case(current_state) //check current_state (from Wait, Five...One Hundred)

   Wait: case(coin_in)
         FIVE_C: next_state <= Five; //all possible coin values (Nickel, Dime, Quarter, Half
Dollar Coin, Dollar Coin)
         TEN_C: next_state <= Ten;
         TWENTYFIVE_C: next_state <= Twentyfive;
         HALFDOLLAR_C: next_state <= Fifty;
         DOLLAR_C: next_state <= Onehundred;
         default: next_state <= Wait;
         endcase
 Five: case(coin_in)
         FIVE_C: next_state <= Ten;
         TEN_C: next_state <= Fifteen;
         TWENTYFIVE_C: next_state <= Thirty;
         HALFDOLLAR_C: next_state <= Fiftyfive;
         DOLLAR_C: next_state <= Onehundredfive;
         default: next_state <= Five;
         endcase
 Ten: case(coin_in)
         FIVE_C: next_state <= Fifteen;
         TEN_C: next_state <= Twenty;
         TWENTYFIVE_C: next_state <= Thirtyfive;
         HALFDOLLAR_C: next_state <= Sixty;
         DOLLAR_C: next_state <= Onehundredten;
         default: next_state <= Ten;
         endcase
 Fifteen: case(coin_in)
         FIVE_C: next_state <= Twenty;
         TEN_C: next_state <= Twentyfive;
         TWENTYFIVE_C: next_state <= Thirtyfive;
         HALFDOLLAR_C: next_state <= Sixtyfive;
         DOLLAR_C: next_state <= Onehundredfifteen;
         default: next_state <= Fifteen;
         endcase
 Twenty: case(coin_in)
         FIVE_C: next_state <= Twentyfive;
         TEN_C: next_state <= Thirty;
         TWENTYFIVE_C: next_state <= Thirtyfive;
         HALFDOLLAR_C: next_state <= Seventy;
         DOLLAR_C: next_state <= Onehundredtwenty;
         default: next_state <= Twenty;
         endcase
 Twentyfive: case(coin_in)
         FIVE_C: next_state <= Thirty;
         TEN_C: next_state <= Thirtyfive;
         TWENTYFIVE_C: next_state <= Forty;
         HALFDOLLAR_C: next_state <= Seventyfive;
         DOLLAR_C: next_state <= Onehundredtwentyfive;
         default: next_state <= Twentyfive;
         endcase
```

```
Thirty: case(coin_in)
        FIVE_C: next_state <= Thirtyfive;
        TEN_C: next_state <= Forty;
        TWENTYFIVE_C: next_state <= Fiftyfive;
        HALFDOLLAR_C: next_state <= Eighty;
        DOLLAR_C: next_state <= Onehundredthirty;
        default: next_state <= Thirty;
        endcase
Thirtyfive: case(coin_in)
        FIVE_C: next_state <= Forty;
        TEN_C: next_state <= Fortyfive;
        TWENTYFIVE_C: next_state <= Sixty;
        HALFDOLLAR_C: next_state <= Eightyfive;
        DOLLAR_C: next_state <= Onehundredthirtyfive;
        default: next_state <= Thirtyfive;
        endcase
Forty: case(coin_in)
        FIVE_C: next_state <= Fortyfive;
        TEN_C: next_state <= Fifty;
        TWENTYFIVE_C: next_state <= Sixtyfive;
        HALFDOLLAR_C: next_state <= Ninety;
        DOLLAR_C: next_state <= Onehundredforty;
        default: next_state <= Forty;
        endcase
Fortyfive: case(coin_in)
        FIVE_C: next_state <= Fifty;
        TEN_C: next_state <= Fiftyfive;
        TWENTYFIVE_C: next_state <= Seventy;
        HALFDOLLAR_C: next_state <= Ninetyfive;
        DOLLAR_C: next_state <= Onehundredfortyfive;
        default: next_state <= Fortyfive;
        endcase
Fifty: case(coin_in)
        FIVE_C: next_state <= Fifty;
        TEN_C: next_state <= Fifty;
        TWENTYFIVE_C: next_state <= Seventyfive;
        HALFDOLLAR_C: next_state <= Onehundred;
        DOLLAR_C: next_state <= Onehundredfifty;
        default: next_state <= Fifty;
        endcase
Fiftyfive: case(coin_in)
        FIVE_C: next_state <= Sixty;
        TEN_C: next_state <= Sixtyfive;
        TWENTYFIVE_C: next_state <= Eighty;
        HALFDOLLAR_C: next_state <= Onehundredfive;
        DOLLAR_C: next_state <= Onehundredfiftyfive;
        default: next_state <= Fiftyfive;
        endcase
Sixty: case(coin_in)
        FIVE_C: next_state <= Sixtyfive;
        TEN_C: next_state <= Seventy;
        TWENTYFIVE_C: next_state <= Eightyfive;
        HALFDOLLAR_C: next_state <= Onehundredten;
        DOLLAR_C: next_state <= Onehundredsixty;
        default: next_state <= Wait;
        endcase
Sixtyfive:case(coin_in)
        FIVE_C: next_state <= Seventy;
        TEN_C: next_state <= Seventyfive;
        TWENTYFIVE_C: next_state <= Ninety;
        HALFDOLLAR_C: next_state <= Onehundredfifteen;
        DOLLAR_C: next_state <= Onehundredsixtyfive;
        default: next_state <= Wait;
        endcase
Seventy: case(coin_in)
        FIVE_C: next_state <= Seventyfive;
        TEN_C: next_state <= Eighty;
        TWENTYFIVE_C: next_state <= Ninetyfive;
        HALFDOLLAR_C: next_state <= Onehundredtwenty;
        DOLLAR_C: next_state <= Onehundredseventy;
        default: next_state <= Wait;
        endcase
Seventyfive: case(coin_in)
        FIVE_C: next_state <= Eighty;
        TEN_C: next_state <= Eightyfive;
        TWENTYFIVE_C: next_state <= Onehundred;
        HALFDOLLAR_C: next_state <= Onehundredtwentyfive;
        DOLLAR_C: next_state <= Onehundredseventyfive;
        default: next_state <= Wait;
        endcase
Eighty: case(coin_in)
        FIVE_C: next_state <= Eightyfive;
        TEN_C: next_state <= Ninety;
        TWENTYFIVE_C: next_state <= Onehundredfive;
        HALFDOLLAR_C: next_state <= Onehundredthirty;
        DOLLAR_C: next_state <= Onehundredeighty;
```

```
            default: next_state <= Wait;
            endcase
    Eightyfive: case(coin_in)
            FIVE_C: next_state <= Ninety;
            TEN_C: next_state <= Ninetyfive;
            TWENTYFIVE_C: next_state <= Onehundredten;
            HALFDOLLAR_C: next_state <= Onehundredthirtyfive;
            DOLLAR_C: next_state <= Onehundredeightyfive;
            default: next_state <= Wait;
            endcase
    Ninety: case(coin_in)
            FIVE_C: next_state <= Ninetyfive;
            TEN_C: next_state <= Onehundred;
            TWENTYFIVE_C: next_state <= Onehundredfifteen;
            HALFDOLLAR_C: next_state <= Onehundredforty;
            DOLLAR_C: next_state <= Onehundredninety;
            default: next_state <= Wait;
            endcase
    Ninetyfive: case(coin_in)
            FIVE_C: next_state <= Onehundred;
            TEN_C: next_state <= Onehundredfive;
            TWENTYFIVE_C: next_state <= Onehundredtwenty;
            HALFDOLLAR_C: next_state <= Onehundredfortyfive;
            DOLLAR_C: next_state <= Onehundredninetyfive;
            default: next_state <= Wait;
            endcase
    Onehundred: case(coin_in)
            NO_C: next_state <= Onehundred;
            FIVE_C: next_state <= Onehundredfive;
            TEN_C: next_state <= Onehundredten;
            TWENTYFIVE_C: next_state <= Onehundredtwentyfive;
            HALFDOLLAR_C: next_state <= Onehundredfifty;
            DOLLAR_C: next_state <= Twohundred;
            default: next_state <= Wait;
            endcase
     default: next_state <= 6'bxxxxxx; // if current_state is neither cases above, set to default

     endcase
  end

  always @(posedge clk, posedge rst) //sequential circuit to update logic behind State Machine
     begin
     if (rst == 1) //reset enabled
```

```
        begin
            current_state <= Wait;
            c_return <= 4'b0000;
            soda <= 0;
            waterbottle <= 0;
        end
    else
        begin
            current_state <= next_state; //update current state from next state
        end
    end

always@(*) //combinational logic, determines
begin
    if(current_state >= Fifty && water_en == 1 && soda_en == 0)// This checks when the
current_state is at least fifty cents and the water enable is on and soda enable is off, indicating
the user wants water, not soda
        begin
            if(current_state == Fifty) // this will check every state at fifty and so on. It will also
determine change based on the current state.
                begin
                waterbottle = 1; //50 cent water will dispense
                end
             if(current_state == Fiftyfive)
                begin
                waterbottle = 1; //50 cent water will dispense
                c_return = Five; //c_return is change
                end
             if(current_state == Sixty)
                begin
                waterbottle = 1; //50 cent water will dispense
                c_return = Ten;
                end
             if(current_state == Sixtyfive)
                begin
                waterbottle = 1; //50 cent water will dispense
                c_return = Fifteen;
                end
             if(current_state == Seventy)
                begin
                waterbottle = 1; //50 cent water will dispense
                c_return = Twenty;
                end
```

```
 if(current_state == Seventyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Twentyfive;
    end
 if(current_state == Eighty)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Thirty;
    end
 if(current_state == Eightyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Thirtyfive;
    end
 if(current_state == Ninety)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Forty;
    end
 if(current_state == Ninetyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Fortyfive;
    end
 if(current_state == Onehundred)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Fifty;
    end
 if(current_state == Onehundredfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Fiftyfive;
    end
 if(current_state == Onehundredten)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Sixty;
    end
if(current_state == Onehundredfifteen)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Sixtyfive;
    end
if(current_state == Onehundredtwenty)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Seventy;
    end
if(current_state == Onehundredtwentyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Seventyfive;
    end
if(current_state == Onehundredthirty)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Eighty;
    end
if(current_state == Onehundredthirtyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Eightyfive;
    end
if(current_state == Onehundredforty)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Ninety;
    end
if(current_state == Onehundredfortyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Ninetyfive;
    end
if(current_state == Onehundredfifty)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundred;
    end
if(current_state == Onehundredfiftyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredfive;
    end
if(current_state == Onehundredsixty)
```

```
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredten;
    end
if(current_state == Onehundredsixtyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredfifteen;
    end
if(current_state == Onehundredseventy)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredtwenty;
    end
if(current_state == Onehundredseventyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredtwentyfive;
    end
if(current_state == Onehundredeighty)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredthirty;
    end
if(current_state == Onehundredeightyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredthirtyfive;
    end
if(current_state == Onehundredninety)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredforty;
    end
if(current_state == Onehundredninetyfive)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredfortyfive;
    end
if(current_state == Twohundred)
    begin
    waterbottle = 1; //50 cent water will dispense
    c_return = Onehundredfifty;
```

```
          end

     end


if(current_state >= Onehundred && soda_en == 1 && water_en == 0)
     begin
          if(current_state == Onehundred)
              begin
              soda = 1; //50 cent soda will dispense
              end
          if(current_state == Onehundredfive)
              begin
              soda = 1; //50 cent soda will dispense
              c_return = Five;
              end
          if(current_state == Onehundredten)
              begin
              soda = 1; //50 cent soda will dispense
              c_return = Ten;
              end
         if(current_state == Onehundredfifteen)
              begin
              soda = 1; //50 cent soda will dispense
              c_return = Fifteen;
              end
         if(current_state == Onehundredtwenty)
              begin
              soda = 1; //50 cent soda will dispense
              c_return = Twenty;
              end
         if(current_state == Onehundredtwentyfive)
              begin
              soda = 1; //50 cent soda will dispense
              c_return = Twentyfive;
              end
         if(current_state == Onehundredthirty)
              begin
              soda = 1; //50 cent soda will dispense
              c_return = Thirty;
              end
         if(current_state == Onehundredthirtyfive)
              begin
```

```
    soda = 1; //50 cent soda will dispense
    c_return = Thirtyfive;
    end
if(current_state == Onehundredforty)
    begin
    soda = 1; //50 cent soda will dispense
    c_return = Forty;
    end
if(current_state == Onehundredfortyfive)
    begin
    soda = 1; //50 cent soda will dispense
    c_return = Fortyfive;
    end
if(current_state == Onehundredfifty)
    begin
    soda = 1; //50 cent soda will dispense
    c_return = Fifty;
    end
if(current_state == Onehundredfiftyfive)
    begin
    soda = 1; //50 cent soda will dispense
    c_return = Fiftyfive;
    end
if(current_state == Onehundredsixty)
    begin
    soda = 1; //50 cent soda will dispense
    c_return = Sixty;
    end
if(current_state == Onehundredsixtyfive)
    begin
    soda = 1; //50 cent soda will dispense
    c_return = Sixtyfive;
    end
if(current_state == Onehundredseventy)
    begin
    soda = 1; //50 cent soda will dispense
    c_return = Seventy;
    end
if(current_state == Onehundredseventyfive)
    begin
    soda = 1; //50 cent soda will dispense
    c_return = Seventyfive;
    end
```

```
        if(current_state == Onehundredeighty)
            begin
            soda = 1; //50 cent soda will dispense
            c_return = Eighty;
            end
        if(current_state == Onehundredeightyfive)
            begin
            soda = 1; //50 cent soda will dispense
            c_return = Eightyfive;
            end
        if(current_state == Onehundredninety)
            begin
            soda = 1; //50 cent soda will dispense
            c_return = Ninety;
            end
        if(current_state == Onehundredninetyfive)
            begin
            soda = 1; //50 cent soda will dispense
            c_return = Ninetyfive;
            end
        if(current_state == Twohundred)
            begin
            soda = 1; //50 cent soda will dispense
            c_return = Onehundred;
            end

    end
end
endmodule
```

Testbench Code (Test.v)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Fabricio Zuniga
// Final Project
// VLSI Design
// EE 4513
// Fall 2020
//////////////////////////////////////////////////////////////////////////////////

module Test();
reg clock, reset, wen, sen;
reg [2:0] coins;
```

```
wire sohda, water;
wire [5:0] change;
wire [5:0] state;

localparam  NO_COIN = 3'b000, NICKEL  = 3'b001, DIME = 3'b010,
            QUARTER = 3'b011, HALFDOLLAR = 3'b100,
            DOLLAR = 3'b101;

VendingMachine test(clock, reset, wen, sen, coins, sohda, water, change, state);

initial
begin
   clock = 0;
   reset = 0;
   wen = 0;
   sen = 0;
end

always
#10 clock =!clock;

initial
begin
coins = HALFDOLLAR; //50 cents
wen = 1; //dispense water, no change
#25
reset = 1; //reset
coins = NO_COIN;
wen = 0;
#25
reset = 0;

coins = DOLLAR; //100 cents or 1 dollar
sen = 1; //dispense soda, no change
#25
reset = 1; // reset
coins = NO_COIN;
sen = 0;
#25
reset = 0;

coins = DOLLAR; //100 cents
#25
```

```
coins = QUARTER; // 25 cents
wen = 1; //dispense water, change is 75 cents
#25
reset = 1;
coins = NO_COIN;
wen = 0;
#25
reset = 0;

coins = DOLLAR;
#25
coins = NO_COIN;
#25
coins = DOLLAR; // 2 dollars, dispense soda, change is 1 dollar
sen = 1;
#25
reset = 1;
coins = NO_COIN;
sen = 0;
#25
reset = 0;


coins = NICKEL;
#25
coins = NO_COIN;
#25
coins = NICKEL;
sen = 1; //10 cents, should not dispense soda
#25
reset = 1;
coins = NO_COIN;
sen = 0;
#25
reset = 0;

coins = QUARTER;
#25
coins = NO_COIN;
#25
coins = QUARTER;
#25
coins = NO_COIN;
```

```
#25
coins = QUARTER;
#25
coins = NO_COIN;
#25
coins = QUARTER; // 1 dollar, soda dispensed, no change
sen = 1;
#25
reset = 1;
coins = NO_COIN;
sen = 0;
#25
reset = 0;

coins = DIME;
#25
coins = NO_COIN;
#25
coins = DIME;
#25
coins = NO_COIN;
#25
coins = DIME;
#25
coins = NO_COIN;
#25
coins = DIME;
#25
coins = NO_COIN;
#25
coins = DIME; //50 cents, water bottle dispense, no change
wen = 1;
#25
reset = 1;
coins = NO_COIN;
wen = 0;
#25
reset = 0;


end

endmodule
```