Name: Chris Baker

UID: 105.180.929

CS M152: Project 4

Vending Machine

Intro

A Finite State Machine can be used to provide the structure for many different electrical systems. As implied, there is a limited number of states and depending on the machine the outputs of a state could either depend on the current state or both the current state and the input. A Moore machine represents the former while a Mealy machine represents the latter FSM.

An example provided to us was the turnstile Mealy Machine which transitioned between the locked and unlocked state based on inputs and the current state. Similar to a real world turnstile a coin inserted into the machine would unlock it and once the turnstile was reset or pushed one rotation to allow someone through, then it would go back to the locked position.

Our lab is focused on designing a vending machine FSM and implementing the Verilog code to make it perform as our design intends it to behave. Our vending machine will contain 20 items total ranging from code inputs of 00 to 19. There is also a card slot and a machine door to read different inputs that would determine which state to move to next in certain situations. Other state transitions will be determined by actions like reset, reload, idle, get code, transact, and vend to name some of the functions encapsulated in our machine. The required inputs from the lab are clk, reset, reload, in, item_code [3:0], key_press, valid_tran, and door_open. The required outputs are vend, invalid_sel, cost [2:0], and failed_tran. We do implement some other variables to help contain and pass along values which are shown in our simulations and tests section as well as the source code for our lab submission. A couple of important one include items which is a 2d array with 5 rows and 4 columns of the items. The array contains 20 items from 00 to 19 and each item has a bit width of 4. The width of 4 allows us to store the 10 items capacity that we restock during a reload.

A reset is intended to reset all values to 0 and then go to idle when the reset signal goes low. In the idle state we will either go to reset, reload, or get_input_code1(C1) depending on which signals are high and low. In general a reset signal being high will force any state to go to reset next. Reload will only be possible when in idle and if reload signal goes high. When reload signal goes low then it can exit the reload state and return to idle if not going to reset. Again anything can go to reset and reset is the priority state when determining between multiple paths to go. If neither reload or reset are set then we can wait for a card_in signal to proceed with making a vending machine transaction. During the get_code_1(C1) we will wait up to, but not including, 5 positive clk edges to register a valid code and key press. If this is correct we proceed to C2, if not or if we have 5 cycles of waiting: we go to Invalid state. In C2 we perform similar checks and if they are valid and pass the logic in our code, then we go to Wait_Tran, else if its Invalid we go to the Invalid state like before. If we are in Wait _Tran then we wait for a valid transaction signal like the one sent between the bank and your card in the machine. If this does not register within 5 cycles, then we will go to Failed state, else we will go to the Vend state. In vend we check if we receive a door open signal within 5 cycles again, if not we go to idle. If we do receive door open signal then we go to WCLOSE. In WCLOSE we wait for the door open signal to go low to move to Idle.

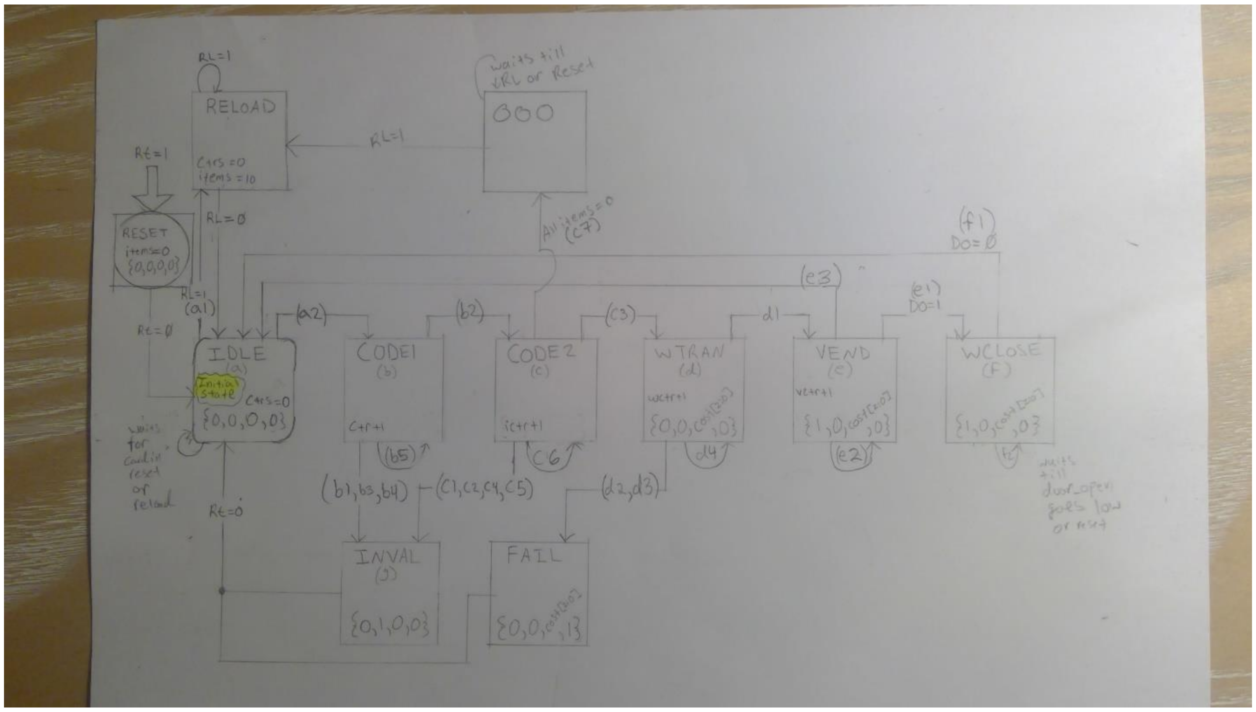
Name: Chris Baker

UID: 105.180.929

Again, any of these states can be interrupted by a reset signal that sends the next state to reset. Reload can only be done from idle state. Idle will start the transaction process to move forward the line of states. In waiting for exactly 5 or more cycles of posedge clks we will either go to invalid, failed, or idle depending on if we are in C1, C2, WTRAN, or VEND. We will also perform the output operations that are associated with each state (fail-failed, inval-invalid, vend-vend, wtran-cost) and resetting them in idle or reset.

Design Description

FSM Diagram: VENDING MACHINE



1. FSM Notes/Legend:

- Refer to the table below for conditions organized by letter and number
- The **Block Arrow** in **RESET** shows that **any state** can have a reset(Rt) **signal of 1 and move to RESET.** For this reason, reset is also shown with a square and concentric circle.
- **IDLE** state is represented with a rounded square block meant to signify it is **the initial state.**
- I originally had zeros in the states that had no output change, but I removed that. I believe this demonstrates better where the changes occur, so the other states just carry along the necessary information.
- {0,0,0,0} represents the 4 main outputs specified in manual {vend, invalid_sel, cost [2:0], fail}
- This table shows more inner workings of variables that take or set values, but do not make the transition

RESET	Ctrs= 0, Items = 0, outs = 0
RELOAD	Ctrs = 0, items = 0
IDLE	Ctrs = 0, outs = 0

Name: Chris Baker

UID: 105.180.929

CODE1	For valid and keypress: dig1 = item_code Ctr = ctr + 1

CODE2	Dig2 = item_code
	Sel = (Logic to determine equals correct key sequence of dig1 and dig2)
	Row = sel/4 , col = sel%4 (for item in array)
	Ictr = ictr +1

WTRAN	wctr = wctr +1 cost [2:0] = (determined by logic using value of sel to check increasing numbers and assign values)
VEND	vctr = vctr +1 items[row][col] = items[row][col] – 1 (unless already at 0, then we prevent this)
WCLOSE	This state just waits for door_open signal to go low or reset to go on.
INVAL	Set inval
FAIL	Sets fail
UPDATING COUNTERS always block Default (everything but, idle, reload, and reset.)	Ctrs = 0
OOO: Out of Order	Enters this state when there are no items left in any section (determined in C2). Only way to get out is through reset or reload. Reset will not solve out of order if there is no idle -> reload to put items back in the machine.

2. Table Notes:

- The variable name it is high or 1
- ! indicates it is low or 0
- Outs=0 refers to all outputs (vend, invalid_sel, failed, cost [2:0])
- Ctrs=0 refers to (ctr, ictr, vctr, wctr) which handle cycle counting.
- I will use parenthesis to explain some value like items is list of 20 items from 00 to 19 so in RELOAD I iterate throught he list and set each item to 10. Also CODE1 is written as C1 in my source code, card_in is written as 'in' for my source code etc. I will mention these once for clarity
- Legend: key_press = kp, item_code = ic, items[row][col] = items with necessary values in row and col, door_open = do
- Every state will go to reset if reset is 1. RESET is a priority state when multiple options are available. As such even though some of these conditions do not mention it, in my source code I check for the reset signal before checking for other states. So, mostly all of these conditions imply reset is not set, or else they would have gone to reset shown in the first row.

Name: Chris Baker

UID: 105.180.929

  - Side note: for all conditions (next state/output): such as vend, if all outputs result in vend = 1 then it is implied that the case where reset is high that the output would also be vend = 1. Then when we move to reset it will be set to 0 like when stats go to idle. Basically because we are setting the outputs upon entering these next state so they would apply those outputs to all conditions within the state (vend, invalid_sel, failed, cost [2:0], outs, ctrs).
- Certain states can loop or stay waiting in their state: RESET, RELOAD, IDLE, WCLOSE. RESET can either never go low (stays high). RELOAD can never go low (stays high). IDLE can keep waiting for card_in. WCLOSE can keep waiting for door_open to go low. However, IDLE can move with reset or reload signal and WCLOSE can move with reset signal. All other states are meant to decide where they will move within 5 clock cycles and make that move at the following posedge clk.

Current	Conditions number	Conditions	Next State/Output
ANY STATE	NA: Applies to all	Reset	RESET/outs=0
RESET	NA: few cases	!reset	IDLE/outs=0
RELOAD	NA: few cases	Reload	RELOAD/outs=0, ctrs=0, items = 10 (for every item 00 to 19)
	NA: few cases	!reload	IDLE/outs=0
IDLE	A1	Reload	RELOAD/outs=0
IDLE	A2	Ci	C1/outs=0
	Else	Stays: waits for input reset, reload, or ci	IDLE/outs=0
C1(CODE1)	B1	Kp & ic >1	INVAL
	B2	Kp & ctr <= 5	C2
	B3	Kp & > 5	INVAL
	B4	!kp & ctr > 5	INVAL
	B5	!kp & ctr < 5	C1
C2(CODE2)	C1	Kp & ic>6	INVAL
	C2	Kp & items == 0	INVAL
	C3	Kp & Ictr < 6	WTRAN
	C4	kp & ictr > 5	INVAL
	C5	!kp & ictr > 4	INVAL
	C6	!kp & ictr< 5	C2
	C7	All items = 0	OOO
WTRAN	D1	Valid & wctr < 6	VEND/cost [2:0]
	D2	Valid & wctr > 5	FAIL
	D3	!valid & wctr > 4	FAIL
	D4	!valid & wctr < 5	WTRAN
VEND	E1	!reset & do & vctr < 6	WCLOSE/vend = 1
	E2	!reset & !do & vctr < 5	VEND/vend= 1
	E3	!reset & !do & vctr >= 5	IDLE/vend = 1
WCLOSE	F1	!reset & !do	IDLE
	F2	!reset & do	WCLOSE
INVAL	G1	!reset	IDLE/invalid_sel = 1
FAIL	H1	!reset	IDLE/failed = 1

OOO	Few cases: (waits for reset or reload)	!reset & reload	RELOAD

3. Explanation:

For the duration of this explanation I will assume the respective signals or conditions are followed from my above charts and diagram to achieve the expected results. The FSM functions by starting with IDLE as the initial state. From here we can move to either reload, reset, or code1. We can reset here to make everything initialized with 0 along with some initial 0s set on program startup. Aster exiting reset to IDLE we can then go into RELOAD to stock the vending machine to 10 items in each spot. After exiting reload we can go back to IDLE and await the card_input. Once we detect a card we move forward to get code1. Assuming we notice the respective signals within 5 posedge clks for every state, then we will move to code 2. Upon noticing the keypress and valid input codes we can set the cost of the item in WTRAN. We wait for the transaction through a valid_transaction signal that represents the bank and card properly expending funds. Further, we move to VEND and decrement the item in that position of my 2d array. The vend signal is now also set. When the customer opens the door we send a door open signal and then we go into a wait stage in wclose until they close the door. They could keep the door open which would cause a problem and needs to be reset if this part is faulty (See Test area). Once they close it we go back to idle. Say we ran out of items, then in CODE2 we would be sent to OOO since we are out of order. We could only get out of here with reload or reset. However, reset does not solve our item problem so we would end up back here from CODE2. So if we reload we can continue as normal. If we wait for more then 5 cycles in CODE1 or CODE2 or have improper values then we get INVALID state and invalid_sel signal. If we wait for more then (and the exact instant) of 5 posedge clks then we go to FAIL state and register failed signal. If we wait for vend for 5 cycles then we go to idle state. Any state in our diagram can go to reload. Reload can loop/wait there if it never goes low. Reload can also loop or wait if it never goes low. Idle also can have this same behavior if no input to move it along. And wClose can also exhibit this behavior if no reset or door_open signal goes low.

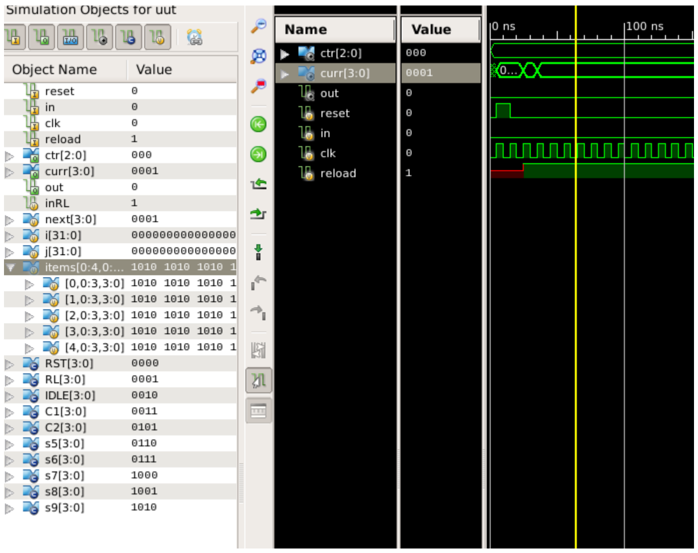
4. Methodology/Errors/Testing to achieve design:

Below is some early testing methodology that helped me largely go from skeleton code to achieving everything above. I started with an idea or goal and then tested what occurred or if I achieved it (checkmarks). I also wrote down things I learned, questions, or reasoning.

1. I began with working code from my TA.
2. Then I proceeded to fix nonblocking errors to get items to 10 in reload. Added logic to switch between reload and idle. Noticed that reset needed to be delayed since in my test bench I kept going to the wrong state. I fixed testbench with values that would move me along to achieve RT->IDLE->RT->RL->IDLE->RT

Name: Chris Baker

UID: 105.180.929

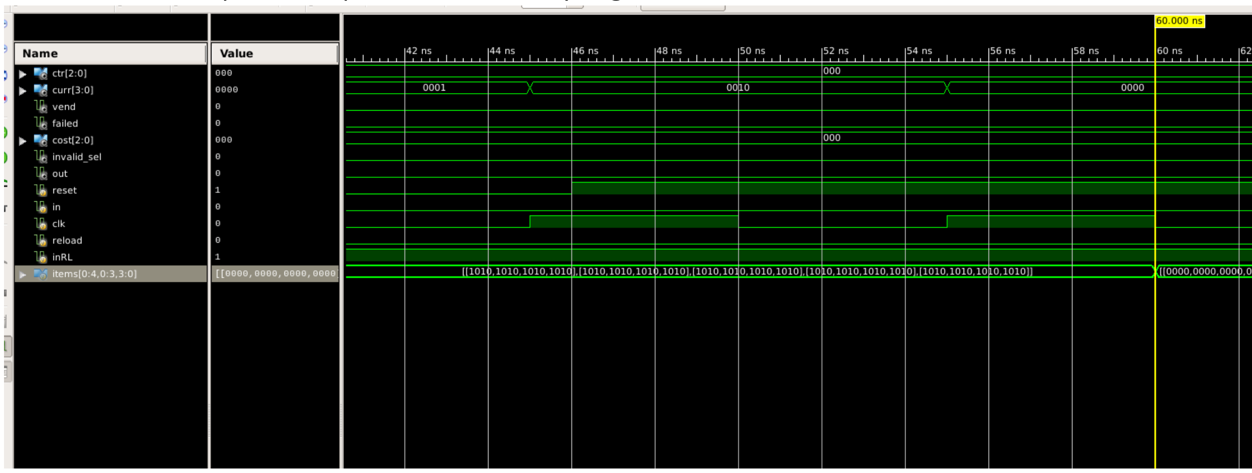


```
Simulation Objects for uut
Object Name        Value
reset              0
in                 0
clk                0
reload             1
ctr[2:0]           000
curr[3:0]          0001
out                0
inRL               1
next[3:0]          0001
i[31:0]            000000000000000
j[31:0]            000000000000000
items[0:4,0:...    1010 1010 1010 1
  [0,0:3,3:0]      1010 1010 1010 1
  [1,0:3,3:0]      1010 1010 1010 1
  [2,0:3,3:0]      1010 1010 1010 1
  [3,0:3,3:0]      1010 1010 1010 1
  [4,0:3,3:0]      1010 1010 1010 1
RST[3:0]           0000
RL[3:0]            0001
IDLE[3:0]          0010
C1[3:0]            0011
C2[3:0]            0101
s5[3:0]            0110
s6[3:0]            0111
s7[3:0]            1000
s8[3:0]            1001
s9[3:0]            1010

Name          Value
ctr[2:0]      000
curr[3:0]     0001
out           0
reset         0
in            0
clk           0
reload        1
```

a.

3. Change items to 0 and outputs. Add ports. Use 2D Array Logic to set items to 0 in ctr block

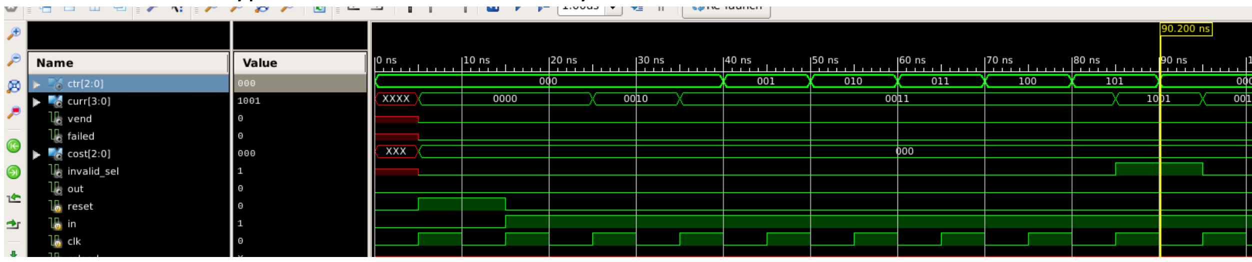


```
Name                  Value
ctr[2:0]              000
curr[3:0]             0000
vend                  0
failed                0
cost[2:0]             000
invalid_sel           0
out                   0
reset                 1
in                    0
clk                   0
reload                0
inRL                  1
items[0:4,0:3,3:0]    [[0000,0000,0000,0000
```

a.

   b. I achieved resetting items to 0 Also had RELOAD(0001) move to IDLE(0010) move to RESET(0000)

4. Go C1->INVAL->IDLE , then RESET->IDLE->CODE1 . With no keypress we have 5 cycles in code1, then we exit before 6th cycle to go to INVAL. INVAL shows invalid signal before IDLE resets the output. INVAL works for checking invalid 5th cycle and stay invalid for one cycle. Need to make sure valid item code and keypress occur for less than 5 cycles.



```
Name           Value
ctr[2:0]       000
curr[3:0]      1001
vend           0
failed         0
cost[2:0]      000
invalid_sel    1
out            0
reset          0
in             1
clk            0
```

a.

   b. Shows my early testing of 5 cycles and catching the invalid signal. RST->IDLE->C1

5. Check Move into WTRAN. Check item code > 9 in C2 goes to INVALID. Check Sel > 19 in C2 goes to INVALID. Set IDLE as initial state. Catch the 5ns key_press clkedge rise to verify cycles in C1 and C2.

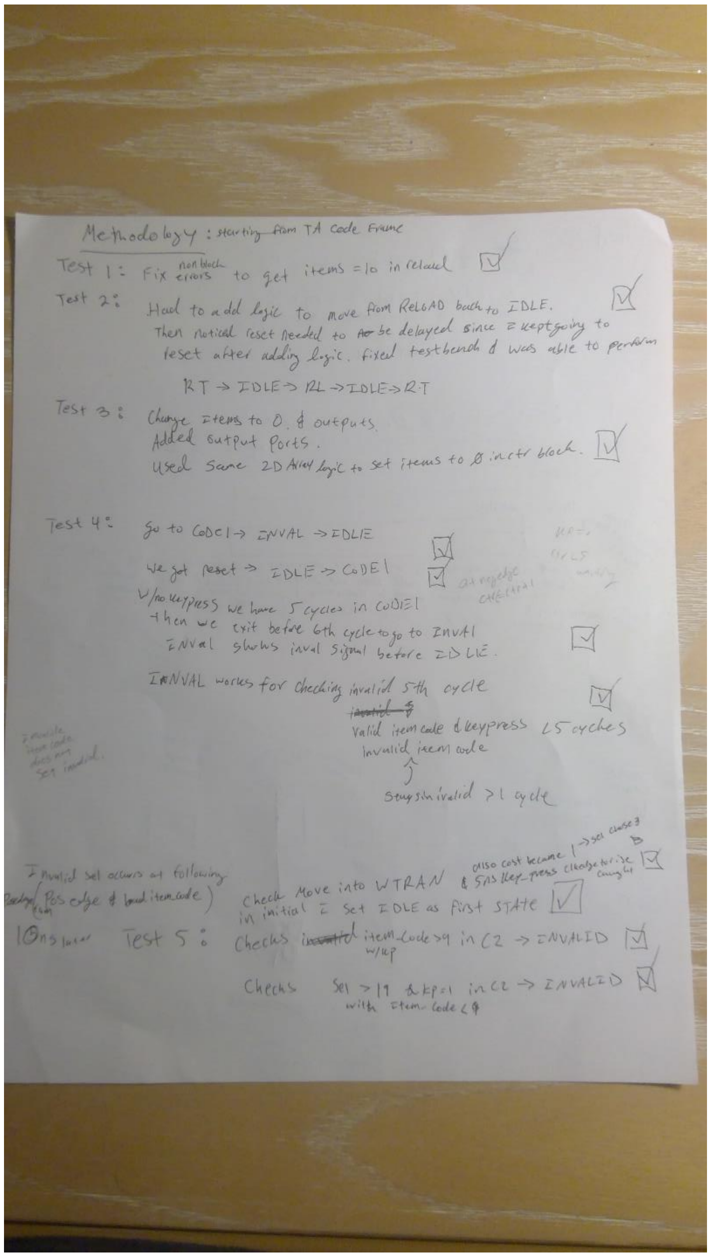
Name: Chris Baker

UID: 105.180.929

6. WTRAN goes to fail when supposed to. WTRAN->FAIL->IDLE achieved (show fail signal, reset ctrs and fail).
7. Checking counter ==5 or >5 for transitions. TA said depending on our logic we will either be checking for 4.5 cycles or 5.5 cycles from beginning of state. This happens as a result of the difference between negedge of clock and posedge of clock. One determines the next state, the other performs the update. Even with a change in my sensitivity list I am able to capture changes up to the 5th posedge of clock (just before the instant of 5th posedge) but not during it. This is also how Verilog applies instantanoues values which take a small delta time for recognition so it could be different results 1ns before or after. Also, some logic may depend upon noticing a falling edge/rising edge in general.
8. Also show some scratch calculations for determining array in bottom right corner and how the clock edge and counters are incrementing.
9. **Errors:** The errors I encountered in this lab came from not using nonblocking assignment when I should have and also not originally understanding the updates of counters and clks at the two different edges. After talking with the TA I confirmed what I believed was happening during my test7 case in this section. I chose to include up to the 5th clock pos edge when I check for cycles. I wrote out the counter and clk increments on scratch paper and I also wrote down the 4 always blocks clearly defining what their roles are (Updates state to next, updates counters, instant changes values for comparison, set output values). By understanding this and working on scratch paper and printed out code I knew my logical errors were from missing (<=) assignment and there were some transitions that were made cleaner. I fixed the below issues with nonblocking assignments and where I assigned the values to occur logically.

Name: Chris Baker

UID: 105.180.929

10. 

Methodology : starting from TA code Frame

Test 1: Fix non block errors to get items = 10 in reload ☑

Test 2: Had to add logic to move from RELOAD back to IDLE. ☑
Then noticed reset needed to ~~Ao~~ be delayed since I kept going to reset after adding logic. Fixed testbench & was able to perform

RT → IDLE → RL → IDLE → RT

Test 3: Change Items to 0. & outputs.
Added output Ports.
Used same 2D Array logic to set items to 0 in ctr block. ☑

Test 4: go to CODE1 → INVAL → IDLE ☑

we got reset → IDLE → CODE1 ☑ [illegible]

w/no keypress we have 5 cycles in CODE1
then we exit before 6th cycle to go to INVAL
INVAL shows inval signal before IDLE. ☑

INVAL works for checking invalid 5th cycle ☑
~~invalid~~ &
Valid item code & keypress <5 cycles
invalid item code
↑
Stays in invalid >1 cycle

[illegible]

Invalid sel occurs at following (Pos edge & bad item code)

10ns later

Check Move into WTRAN also cost became 1 → sel case 3 & 5ns key_press [illegible] caught ☑

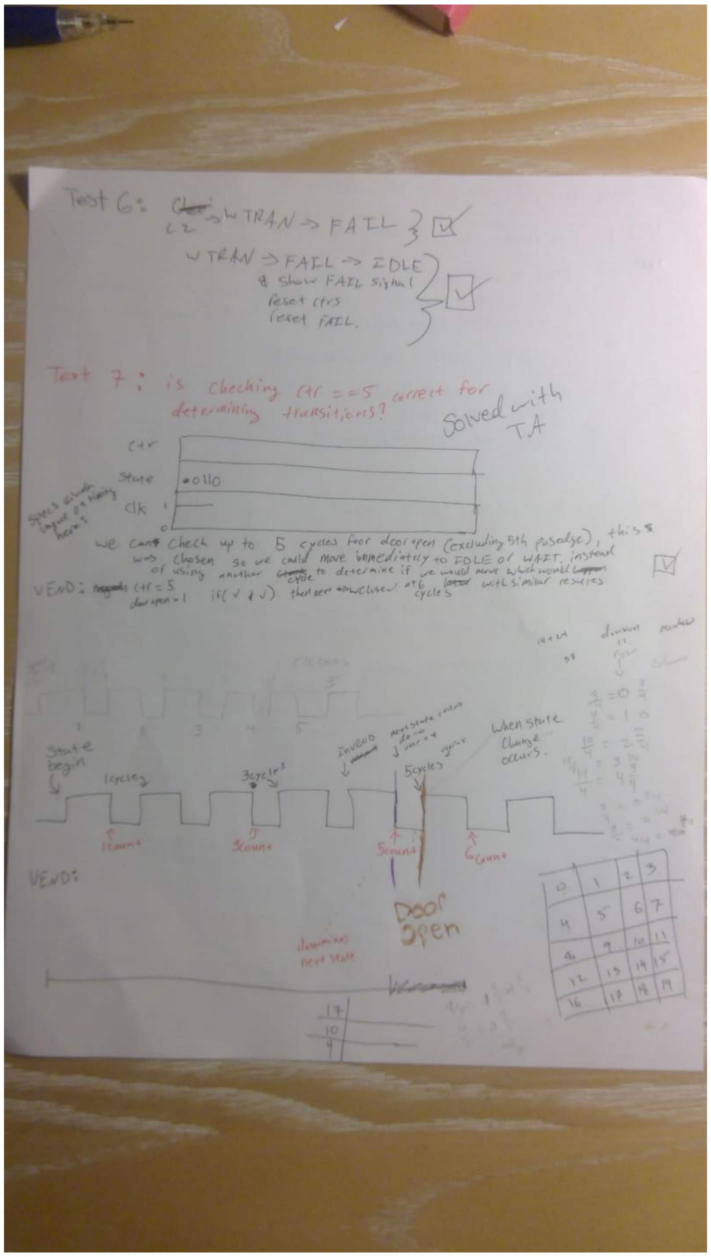
in initial I Set IDLE as first STATE ☑

Test 5: Checks ~~invalid~~ item_code > 9 w/kp in C2 → INVALID ☑

Checks Sel > 11 & kp = 1 in C2 → INVALID ☑
with item_code < 9

Name: Chris Baker

UID: 105.180.929



Test 6: ~~Ches~~ C2 → W TRAN → FAIL } ☑

W TRAN → FAIL → IDLE
& show FAIL signal
reset ctrs
reset FAIL. } ☑

Test 7: is checking ctr == 5 correct for determining transitions?

Solved with TA

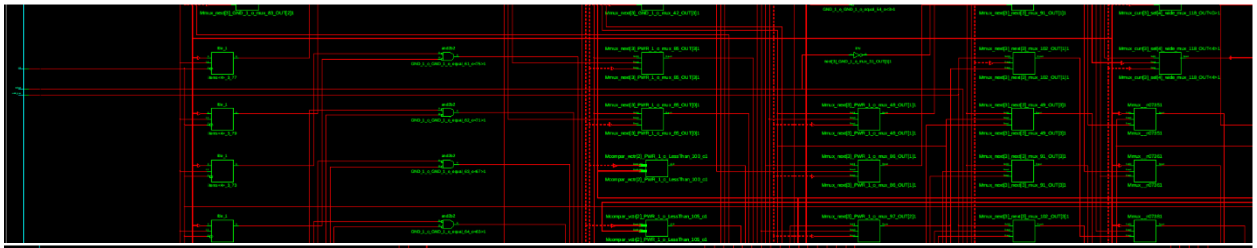
we can check up to 5 cycles for door open (excluding 5th posedge), this was chosen so we could move immediately to IDLE or WAIT, instead of using another ~~clock~~ cycle to determine if we would move which would ~~happen~~ ~~later~~ with similar results ☑

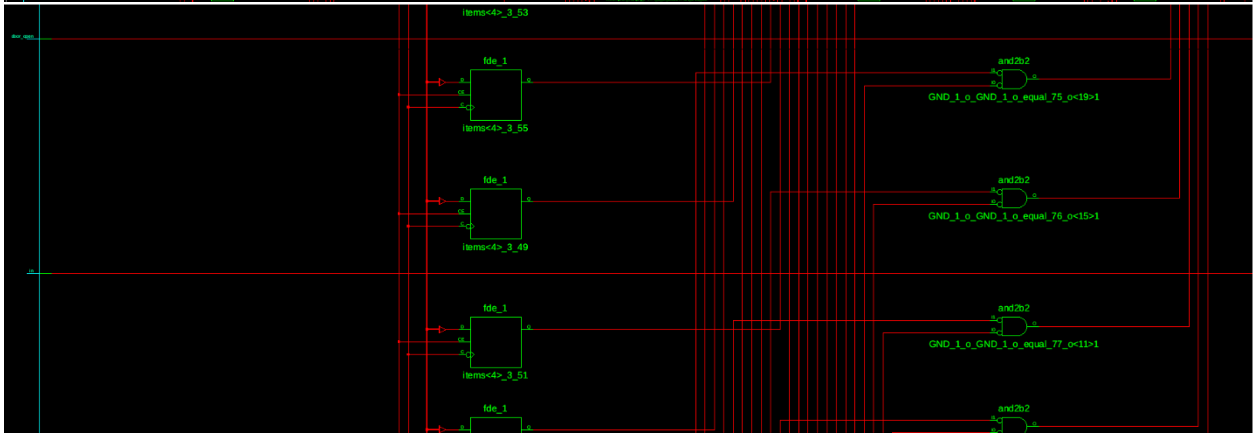
VEND: ~~[illegible]~~ ctr = 5
door open = 1 if(✓ & ✓) then next → WClosed at 6 cycles

VEND:

Name: Chris Baker

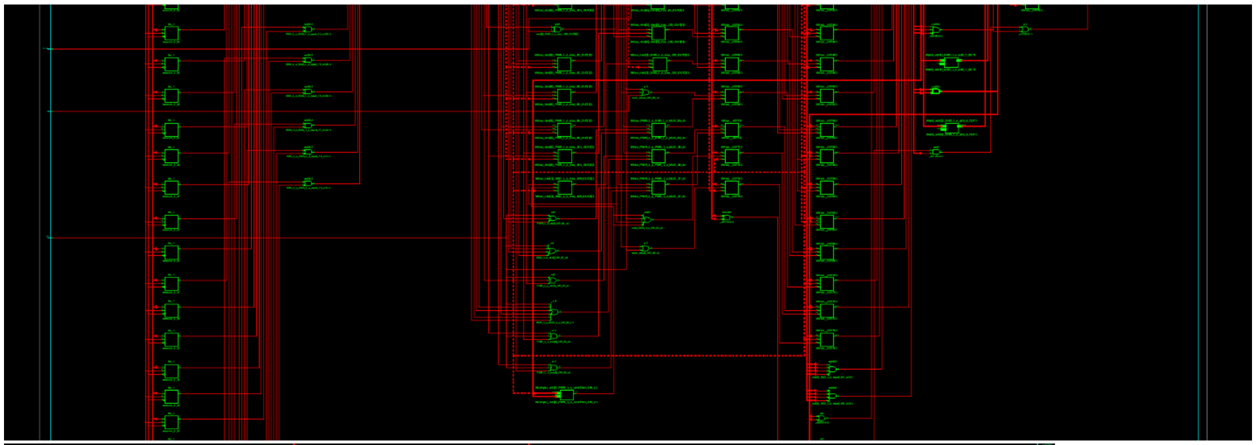
UID: 105.180.929

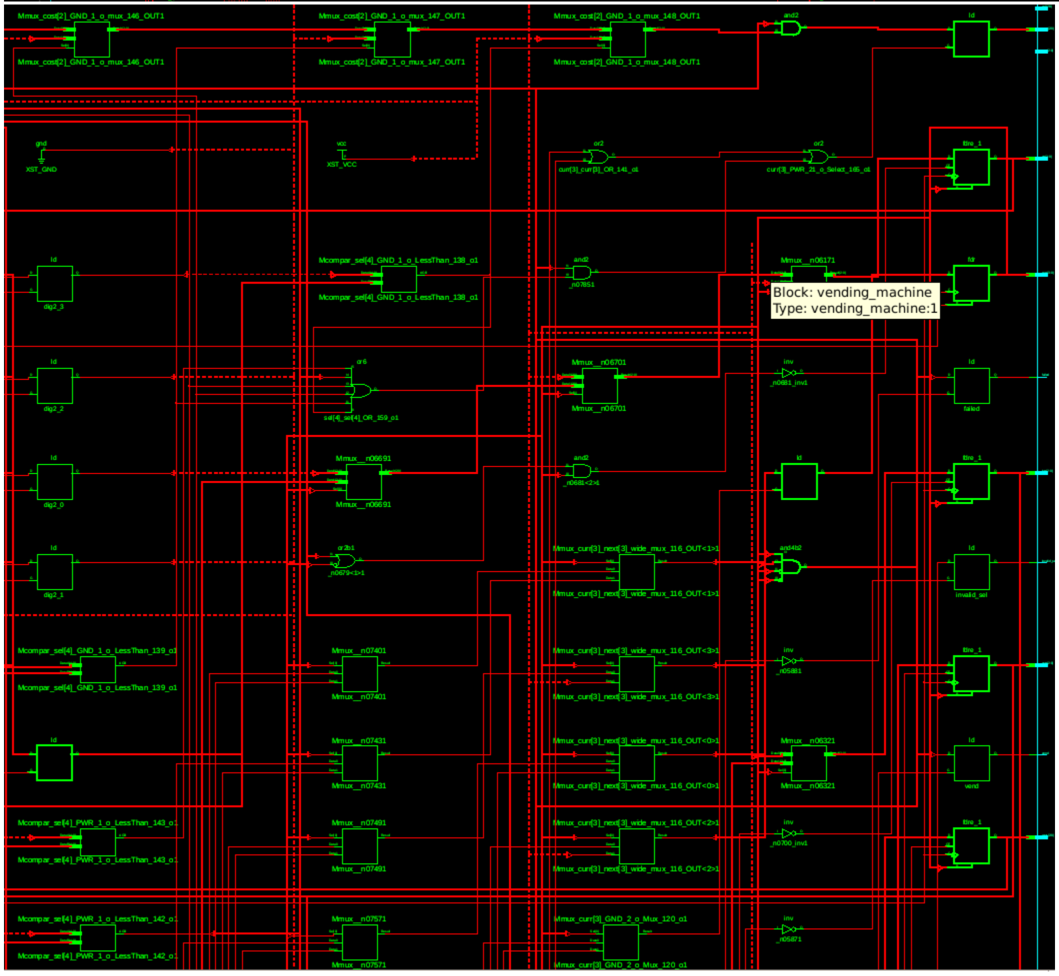




Name: Chris Baker

UID: 105.180.929





Muxs choose numbers for deciding which items we are at and using those items for checking valid conditions later.

The flip flops hold the register values, states, and outputs. Also decide the clk for reset or not. Very important for the always block logic and holding and changing values.

The Combination logic (ORS, ANDS, Square blocks, etc) perform much of the checking going on within the various code blocks. For example the sel less then 138 , determines the selected item if it within a certain number range before choosing invalid. Other selected value wires in our mux determine the cost.

Simulations and Tests

I made several tests along the way, but **for grading purposes I recommend you see the highlighted bold test cases below and choose the ones you find most interesting**. This is because I included more tests then the recommended 1 working test and at least 4 for 80%. I tried to include a couple unique ones and several that are interesting with highlights. **Errors section included above as I checked testing and methodology for design.**

TEST 1: Full Working Simulation

Shown below is the 0ns to 60ns timeframe of our working example.

We will traverse as such: IDLE->RESET->IDLE->RELOAD->IDLE->C1->C2

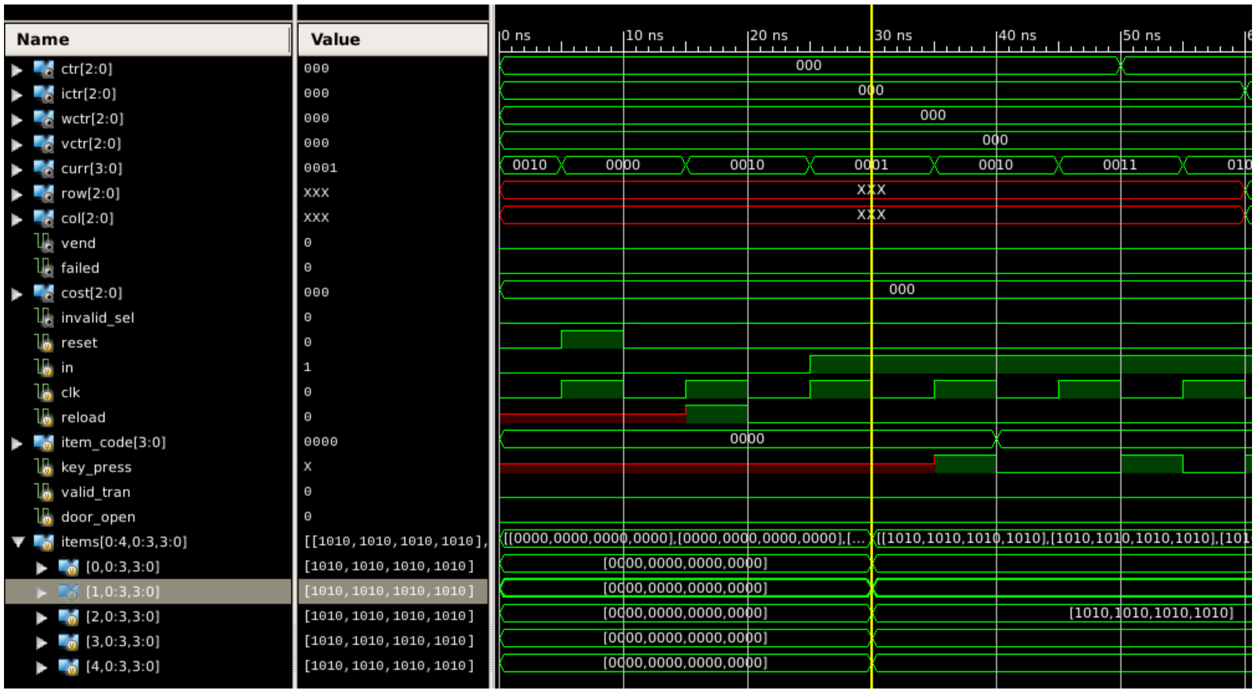
Digits for these are such: 0010->0000->0010->0001->0010->0011->0100

By doing this traversal we will see the correct implementation of several features:

1. IDLE is the initial state
2. IDLE ->RESET, IDLE->RELOAD, IDLE->C1 if given correct signals (reset, reload, or in)
3. RESET->IDLE and RELOAD->IDLE when their signals go low within their state
4. Item counts are set to 0000 for every item initially in vending machine
5. Item counts are set to 1010 for every item when in RELOAD for vending machine
6. When in IDLE and no reset or reload signal, and if there is a card in signal, IDLE->C1
7. If C1 has a valid input (item_code is 0 or 1, its less than 5 cycles, and keypress is on) then C1->C2

Name: Chris Baker

UID: 105.180.929



Name	Value
ctr[2:0]	000
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0001
row[2:0]	XXX
col[2:0]	XXX
vend	0
failed	0
cost[2:0]	000
invalid_sel	0
reset	0
in	1
clk	0
reload	0
item_code[3:0]	0000
key_press	X
valid_tran	0
door_open	0
items[0:4,0:3,3:0]	[[1010,1010,1010,1010],
[0,0:3,3:0]	[1010,1010,1010,1010]
[1,0:3,3:0]	[1010,1010,1010,1010]
[2,0:3,3:0]	[1010,1010,1010,1010]
[3,0:3,3:0]	[1010,1010,1010,1010]
[4,0:3,3:0]	[1010,1010,1010,1010]

We begin in the IDLE state (0010). Reset (0000) is high at the rising clock at 5ns which triggers a move into RESET state. We turn off the reset signal so on the next posedge of clk we can move into IDLE at 15ns. From IDLE we send the reload signal as high which causes another move into RELOAD (0001) on the posedge of the clock at 25ns. In RELOAD notice that the 2d array resembles the lab manual display from 00 in top left to 19 in the bottom right corner. Each of the items is now set to 10 during RELOAD. It is also important to notice that we initially set items to 0 in the item 2d array on startup and that value was changed again (stayed as zero) when we called RESET.

We turn off the reload signal at 15 ns, which means that our RELOAD state will check on the negedge of clk at 25ns which state it will move into next. Since there is no reset and reload is low, then we will move into IDLE again at the next posedge of the clk. Back in IDLE at 35ns we want to move into C1 to collect the first input code. For this to happen I make sure there is no reset or reload signal, but I also must apply the card in signal. The card in signal has been on since 25ns. This demonstrates that a new transaction (we cannot move into C1 to get codes) directly from the RELOAD state as described in the Lab Manual. We can only go to C1 if we are in the IDLE state. With the card in signal high we will move from the IDLE state to C1 state at 45ns. Take note here that the cost has been 000 all this time as well on the vending machine since it does not know which item is selected yet. At the negedge of the clk I register a keypress signal at 50ns to signal the always block code to determine the next state. At this time, we will pass an item_code of 0001 (shown in the next photo, 0000 ends at 40ns, where 0001 begins) during that key_press signal. This was a valid number and happened within 5 cycles, so we move into C2.

The next traversal is such (starting at 55ns): C2->WTRAN->VEND->WCLOSE->IDLE

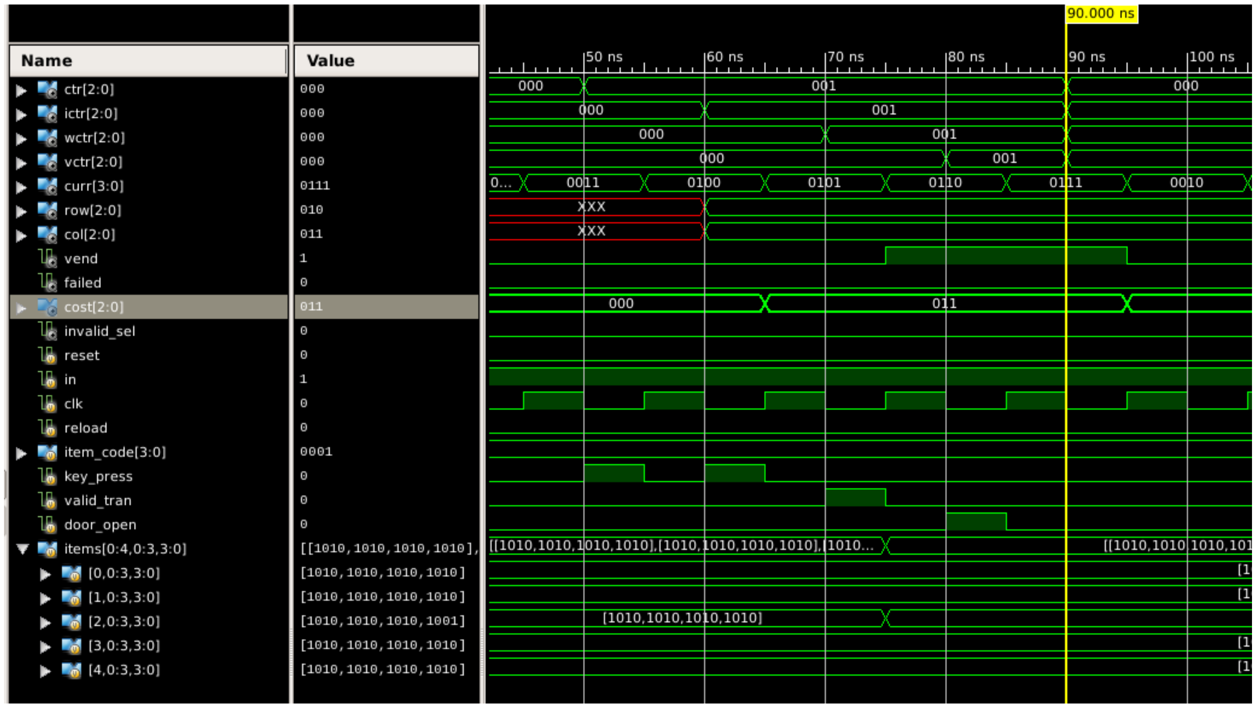
Digits for these corresponding states are: 0100 -> 0101 -> 0110 -> 0111 -> 0010

These features demonstrate:

Name: Chris Baker

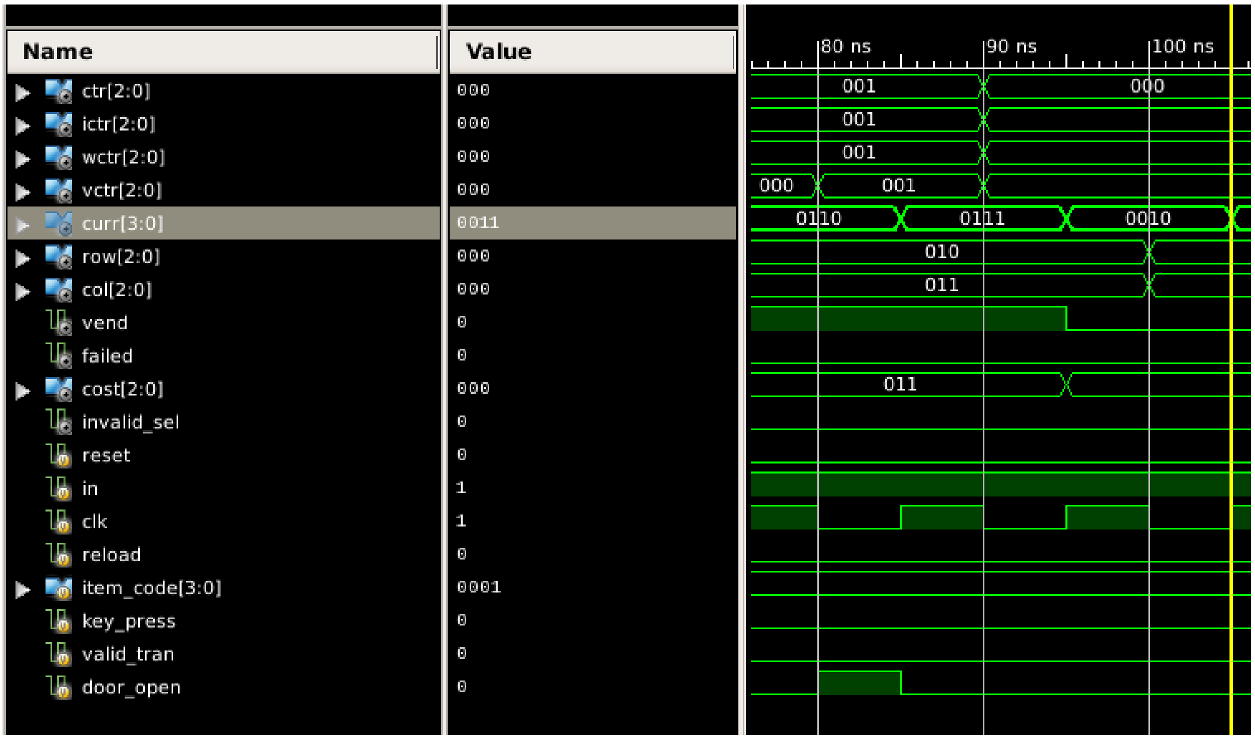
UID: 105.180.929

1. C2 properly obtaining the keypress and item code and tests valid inputs
2. If not INVALID then C2->WTRAN to wait for valid_signal and sets the cost of the item selected.
3. A valid_tran signal in WTRAN within 5 cycles will cause WTRAN->VEND
4. VEND causes the vend signal to be set and simultaneously our item quantity changes in items[2][3] from 1010 to now 1001.
5. VEND responds to the door_open signal that occurs within 5 cycles then moves to DOOR_OPEN
6. Where door_open signal goes low, we move back to IDLE to await a RESET, RELOAD, or new transaction in C1.



Name	Value
ctr[2:0]	000
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0111
row[2:0]	010
col[2:0]	011
vend	1
failed	0
cost[2:0]	011
invalid_sel	0
reset	0
in	1
clk	0
reload	0
item_code[3:0]	0001
key_press	0
valid_tran	0
door_open	0
items[0:4,0:3,3:0]	[[1010,1010,1010,1010],
[0,0:3,3:0]	[1010,1010,1010,1010]
[1,0:3,3:0]	[1010,1010,1010,1010]
[2,0:3,3:0]	[1010,1010,1010,1001]
[3,0:3,3:0]	[1010,1010,1010,1010]
[4,0:3,3:0]	[1010,1010,1010,1010]

At 55ns we moved into C2(0100) where we registered another key press occurred at 60ns and the number of cycles was within 5 cycles. The item code also matched appropriate checks such as the second digit was not greater than 9 and the combination of the first and second values gave a valid item selection from 00 through 19 in our items. I also used the variable for row and col to get the correct 2d array location of the item from items by using another variable called sel and performing modulo and division arithmetic. Because we meet the conditions we are able to pass this new item_code along as our selection (sel is a separate variable I used in my code to combine the values, not shown above, but is used to access the correct item and change its value later proving it works correctly). At 65ns we are now in WTRAN(0101) state. Notice here that our cost has been set to 011 = \$3. Further, row 010 = 2 and col 011 = 3 have also been determined. This is correct since our item_code was 0001 for the first digit and 0001 for the second digit. This means we have an item selection of 11 which matches row 2 and column 3 -> 2*4 + 3 = 11^{th} item. In WTRAN we also receive our signal for valid_tran at 70ns which allows us to set next state and change that state to VEND at the next posedge of clk. At 75ns we are in VEND(0110) and simultaneously set our vend signal high. At this point we also see that the 11^{th} item in our vending machine has changed from 1010 to 1001 to signify it has been vended and is awaiting retrieval. We must wait for the door_open signal to go high at 80ns. If this did not happen then we would have waited 5 cycles and returned to the IDLE state. We see the door_open signal go high, so we move to the WCLOSE(0111) state where we wait for the user to close the door. The door_open signal went low at 85ns right when WCLOSE and posedge of clk happened. so WCLOSE is already prepared to move back to the IDLE state upon the next posedge of the clk at 95ns.



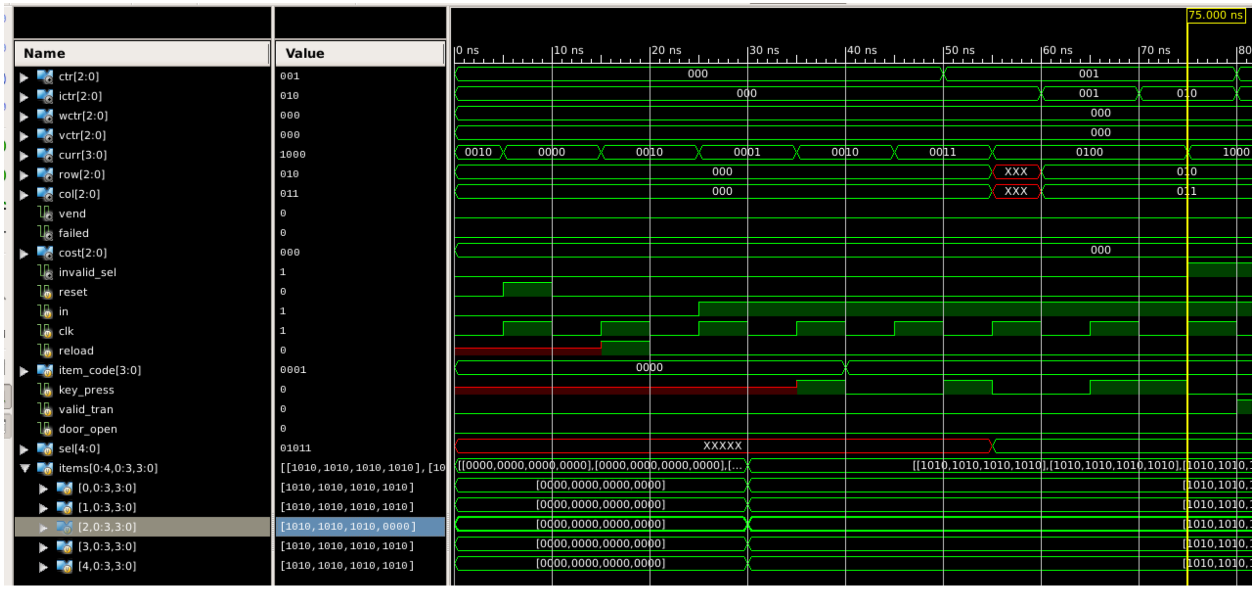
This last photo above shows that counters and output values (cost, invalid_sel, failed, vend) went to zero. The row and col were also reset. A couple of the input values like item_code and card were left on to keep testing other details. This demonstrates our successful completion of one full transaction from beginning to end while also showing the ability to move correctly between RESET, RELOAD, and IDLE at the very beginning of this transaction. In general, RELOAD can only happen during IDLE. While RELOAD is happening a new transaction (moving to C1 and collecting codes ... etc) will never occur. Lastly, any state will move to RESET if given the signal. Some of these may be demonstrated in the following tests, however my source code also shows these facts.

TEST 2: Trying to Purchase a Sold-Out Item

Code Note: For simplicity I placed a comment under the RELOAD state in my Update Counters section of the code source file for vending_machine.v where I set the items[2][3] <= 0 . That way we can run the same program values from before, but we should see an INVALID case from the soldout item.

Name: Chris Baker

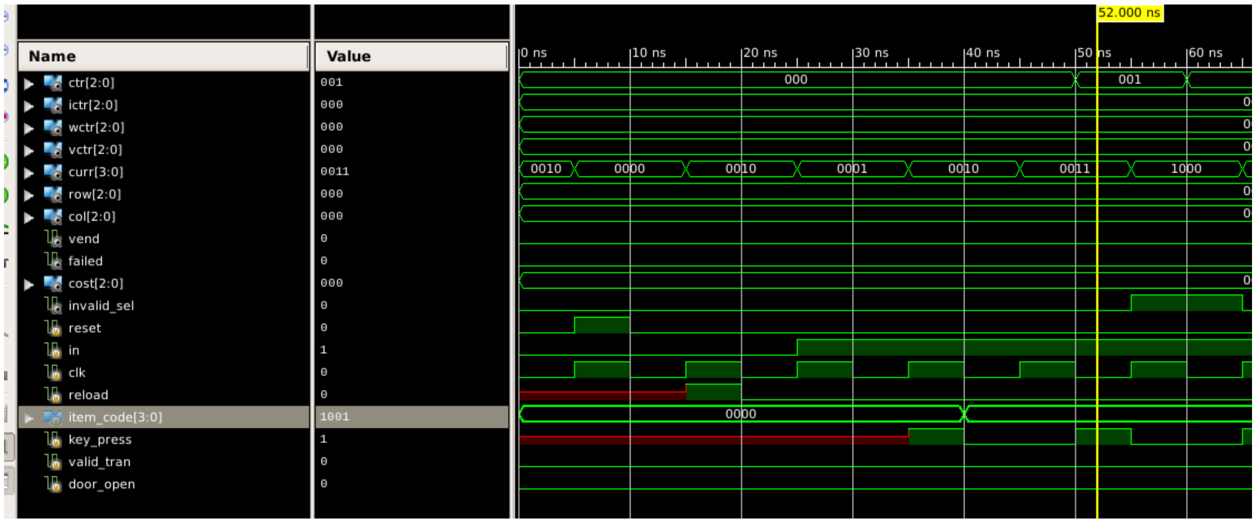
UID: 105.180.929



Name	Value
ctr[2:0]	001
ictr[2:0]	010
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	1000
row[2:0]	010
col[2:0]	011
vend	0
failed	0
cost[2:0]	000
invalid_sel	1
reset	0
in	1
clk	1
reload	0
item_code[3:0]	0001
key_press	0
valid_tran	0
door_open	0
sel[4:0]	01011
items[0:4,0:3,3:0]	[[1010,1010,1010,1010],[10
[0,0:3,3:0]	[1010,1010,1010,1010]
[1,0:3,3:0]	[1010,1010,1010,1010]
[2,0:3,3:0]	[1010,1010,1010,0000]
[3,0:3,3:0]	[1010,1010,1010,1010]
[4,0:3,3:0]	[1010,1010,1010,1010]

Show above we transition from IDLE->RESET->IDLE->RELOAD->IDLE->C1->C2 and during C2(0100) we perform a check on the item in items[row][col] = items[2][3] = 0000 . Since this item is out of stock, then our next state at 75ns becomes INVALID (1000). Also, the value of the out of stock item does not change, it remains 0000 until we implement a reload or force a change with the test bench.

TEST 3: User enters a key greater than 1 for first key

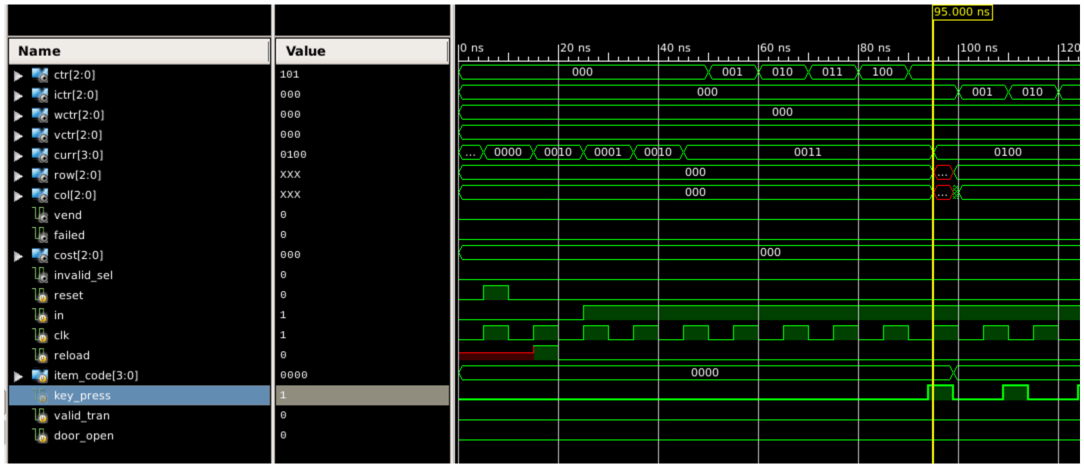


Name	Value
ctr[2:0]	001
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0011
row[2:0]	000
col[2:0]	000
vend	0
failed	0
cost[2:0]	000
invalid_sel	0
reset	0
in	1
clk	0
reload	0
item_code[3:0]	1001
key_press	1
valid_tran	0
door_open	0

When the user enters a key greater then 3, item_code = 1001 which is 9, such as In this case. Then my logic checks if dig1 > 1 since we only have vending machine item spots for 00 to 19.

Thus, anything other then 0 and 1 do not exist for the first digit. We do this during our state C1(0011) beginning at 45ns; which gathers the item_code input for the first digit. By checking for this during the key press and with that incorrect item_code value we can correctly update our next state to be INVAL. The transition to this state occurs at the next posedge of the clk as seein at 55ns when we move to INVAL(1000)

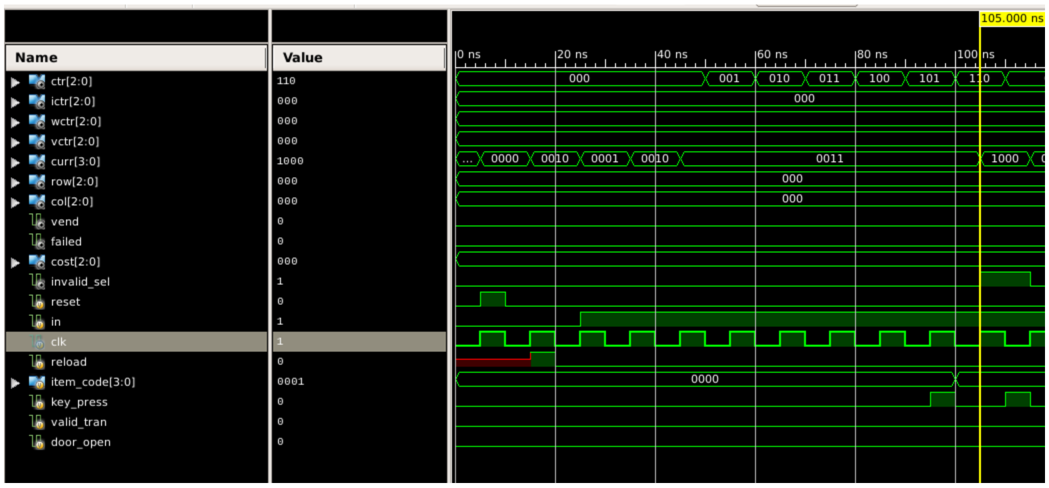
TEST 4: User waits almost 5 cycles (4a) or waits at least 5 cycles (4b) before entering first key



(**4a**) For the picture above we allow a key press signal to be registered just before the last 5th posedge of clk which makes for a smooth transition to C2. Our first always block changes current state on posedge of clk. Our second always block decides the ctr values at negedge of clock. The sensitivity list in our third block detected changes in the ctr values and in the key_press signal at 94 ns from low to high. This caused the next state to be recalculated and change to occur at the next posedge of the clk.

Name: Chris Baker

UID: 105.180.929



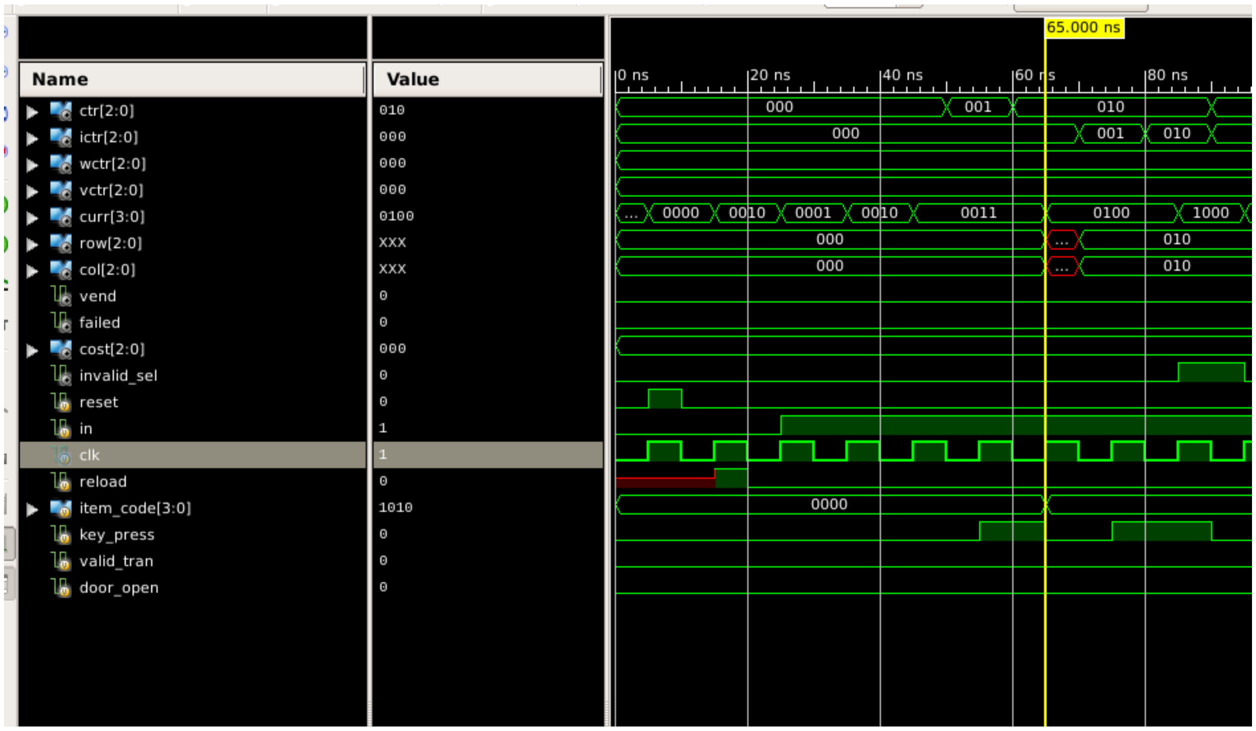
Name	Value
ctr[2:0]	110
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	1000
row[2:0]	000
col[2:0]	000
vend	0
failed	0
cost[2:0]	000
invalid_sel	1
reset	0
in	1
clk	1
reload	0
item_code[3:0]	0001
key_press	0
valid_tran	0
door_open	0

(4b) In the above we make the keypress signal go high 1 ns later at 95ns. This means the change occurred right on the 5th posedge of the clk and we did not want to transition from C1 to C2 as we did in (4a). Therefore, we had logic determine to not make this transition when updating counters and checking the sensitivity list conditional logic. Further, the update occurs at the next posedge of the clk and our counter has moved to value 6 as well. We correctly went to the INVALID state waiting 5 clock cycles. My my TA and I discussed the general effect between using the posedge and negedge to count events. Because of the 5ns difference in their checking of events "there is half a cycle delay between counter update and state update and your condition for time out counter > 5 or (counter == 5) will determine if you will allow 5.5 cycles or 4.5 cycles after the [initial prior] state change" to detect and transition to the next state. With my design we can capture just up until the 5th posedge of the clk from the beginning of the state, but we will not detect the exact instant of the 5th posedge of the clock; just like here where we then decide to move to the INVALID state at the next posedge as a result. This is also due to small delta times in Verilog where a value turning on or turning off at a given instant can play a role in detection too, so you would have to extend set periods to check at those instances. My TA agreed the lab manual did not specify whether to include or exclude the exact 5th posedge of the clk so I interpreted and implemented it to take up to the 5th posedge which is nice to allow those quick transitions on detection to C2, but if not we will wait till next posedge in this case and then go to INVALID.

TEST 5: User enters a key greater than 9 for second key

Name: Chris Baker

UID: 105.180.929

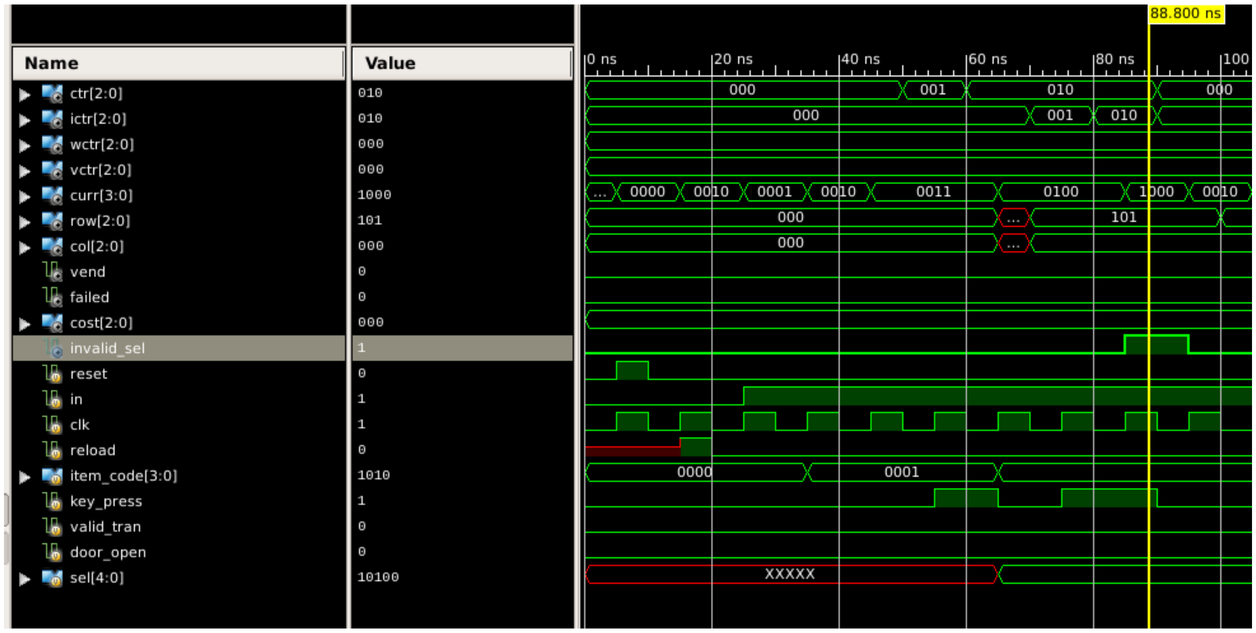


Name	Value
ctr[2:0]	010
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0100
row[2:0]	XXX
col[2:0]	XXX
vend	0
failed	0
cost[2:0]	000
invalid_sel	0
reset	0
in	1
clk	1
reload	0
item_code[3:0]	1010
key_press	0
valid_tran	0
door_open	0

The above example shows that in C2(0100) starting at 65ns we only count from 000 to 010 in our ictr before a state transition takes place at 80ns to INVAL(1000). This means that we did not wait until the 5th posedge of the clock to determine the input was invalid. Instead we determined that if the user entered a key (key_press) and if they entered a value unusable (item_code > 9 in C2) then we would determine the next state to be INVAL. Shown in our chart of white values in the black widow item_code = 1010 . Thus, 10 was entered > 9 and we went to INVAL state as a result.

TEST 6: User tries to enter a key greater than 19 (My logic prevents this case: I check dig1 > 1, dig2 > 9, but we hold this value in a variable and check it)

Name: Chris Baker
UID: 105.180.929



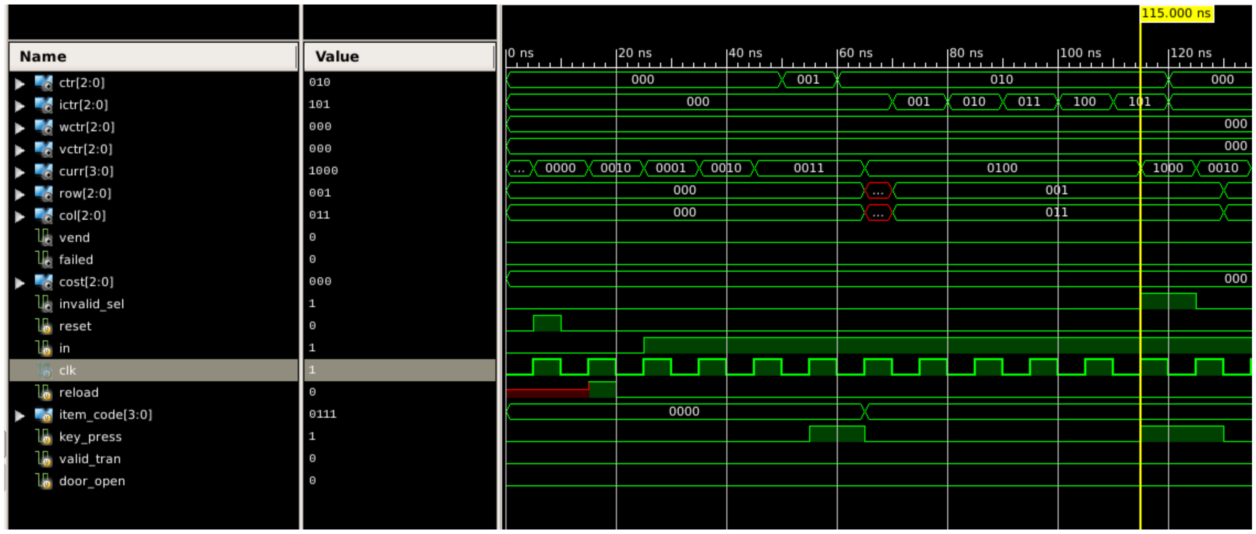
Name	Value
ctr[2:0]	010
ictr[2:0]	010
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	1000
row[2:0]	101
col[2:0]	000
vend	0
failed	0
cost[2:0]	000
invalid_sel	1
reset	0
in	1
clk	1
reload	0
item_code[3:0]	1010
key_press	1
valid_tran	0
door_open	0
sel[4:0]	10100

In this case I added item_code = 0001 starting at around 35 ns and then around 65 ns the item_code = 1010 like in our previous example. Instead of using the test bench I was able to show with my sel (item selection register) that sel = 10100 . This means sel correctly grabbed 1 and somehow was able to take 10 (Hacking? Because this vending machine should only have keys for 0 – 9 , so this should not be possible to enter something greater than 10 on a single key). My logic in C1 handles for dig1 > 1 by sending the code to invalid at that point. The handling of C1 was already demonstrated in **(TEST3).** So, anything above 1 would already be invalid. My logic in C2 handles for dig2 > 9 and would send it to invalid at that point too. Therefore, I handle these cases separately in C1 and C2 to correctly decide the next state, but I also put another else if condition in C2 to detect when sel > 19 too as a safety net.

TEST 7: User waits 5 cycles (7a) vs User waits almost 5 cycles (7b) for second input_code

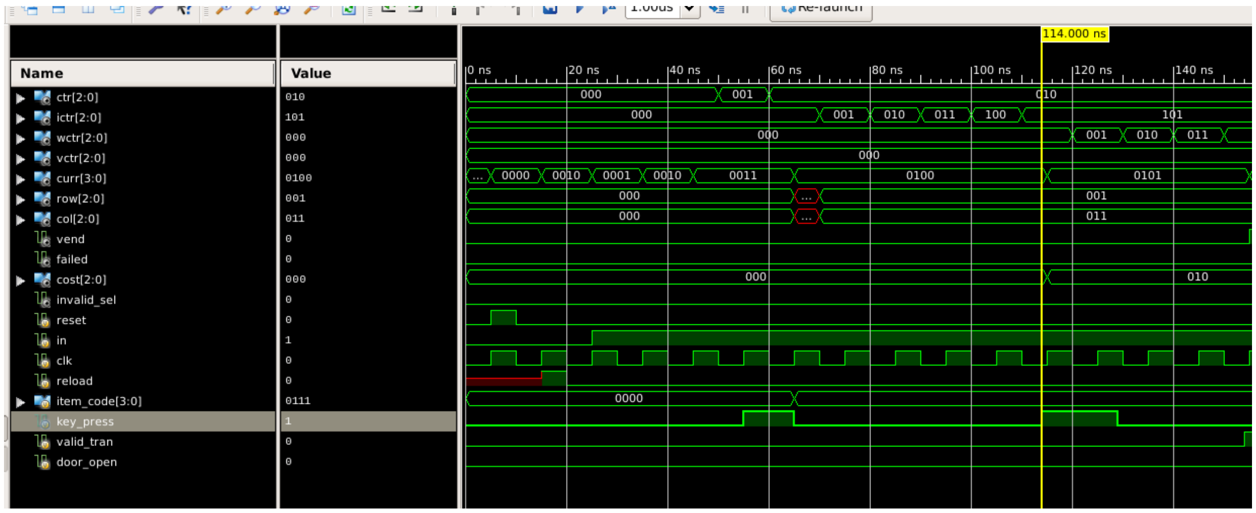
Same concept, slightly different code to perform results from TEST 4

Name: Chris Baker
UID: 105.180.929



Name	Value
ctr[2:0]	010
ictr[2:0]	101
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	1000
row[2:0]	001
col[2:0]	011
vend	0
failed	0
cost[2:0]	000
invalid_sel	1
reset	0
in	1
clk	1
reload	0
item_code[3:0]	0111
key_press	1
valid_tran	0
door_open	0

(7a) briefly: Just like in **TEST 4** we count posedge clks starting in C2(0100) at 65ns. On the 5th posedge clk we are at 115ns (115 – 60 = 55ns makes sense for the 5ns difference between posedge and negedge as well as the 10ns clock periods, 5 of them).Because keypress turned on just at the 5th posedge we already were making the transition into INVAL(1000).

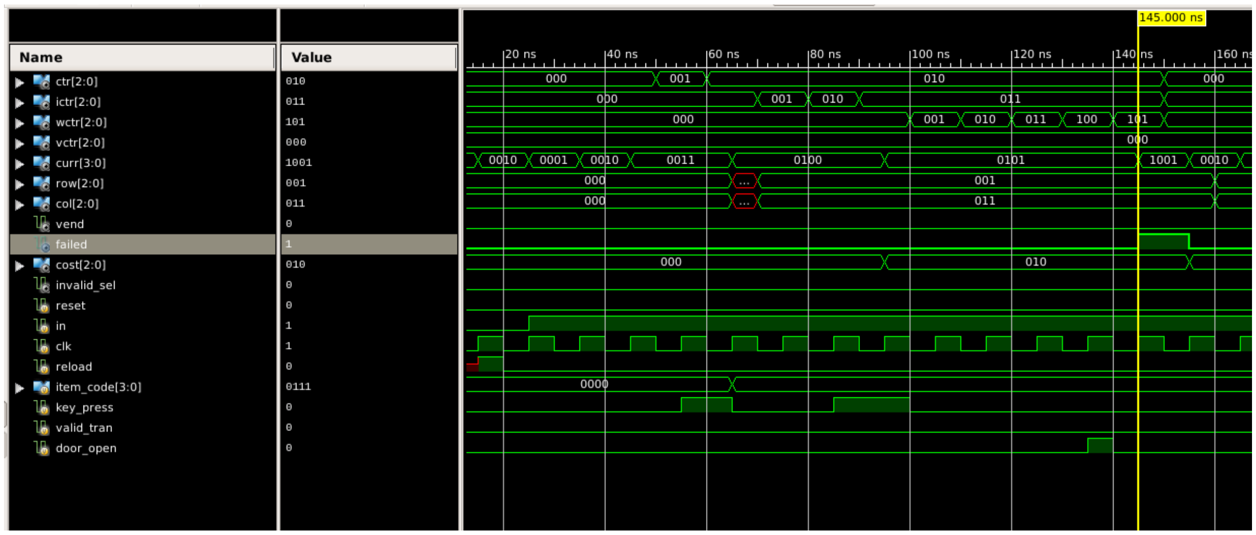


Name	Value
ctr[2:0]	010
ictr[2:0]	101
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0100
row[2:0]	001
col[2:0]	011
vend	0
failed	0
cost[2:0]	000
invalid_sel	0
reset	0
in	1
clk	0
reload	0
item_code[3:0]	0111
key_press	1
valid_tran	0
door_open	0

(7b) briefly: Just like in **TEST 4** we see that checking up until the very last moment before the 5th posedge of the clk will allow us to register the item_code and the key_press at 114ns in C2(0100) to make a smooth and immediate transition to WTRAN(0101) at 115ns.

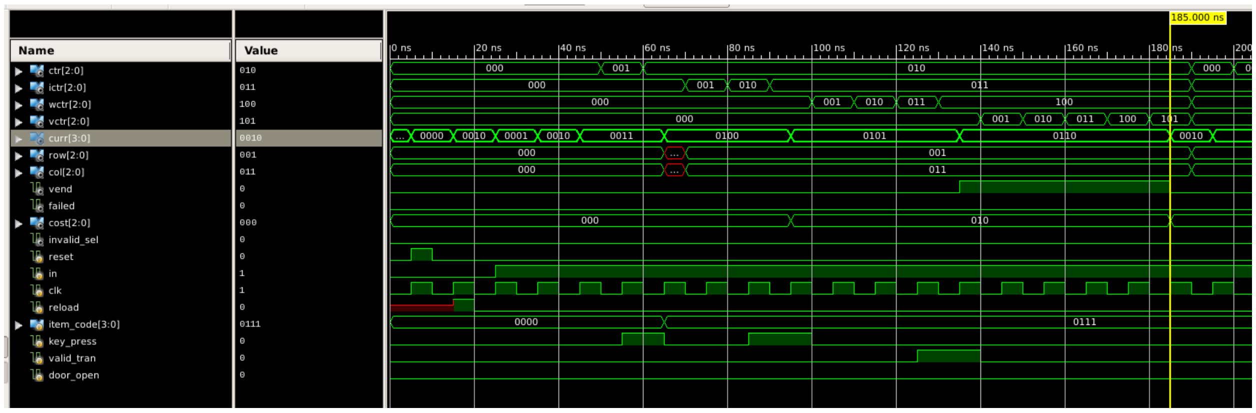
TEST 8: User waits 5 cycles for valid signal

Name: Chris Baker
UID: 105.180.929



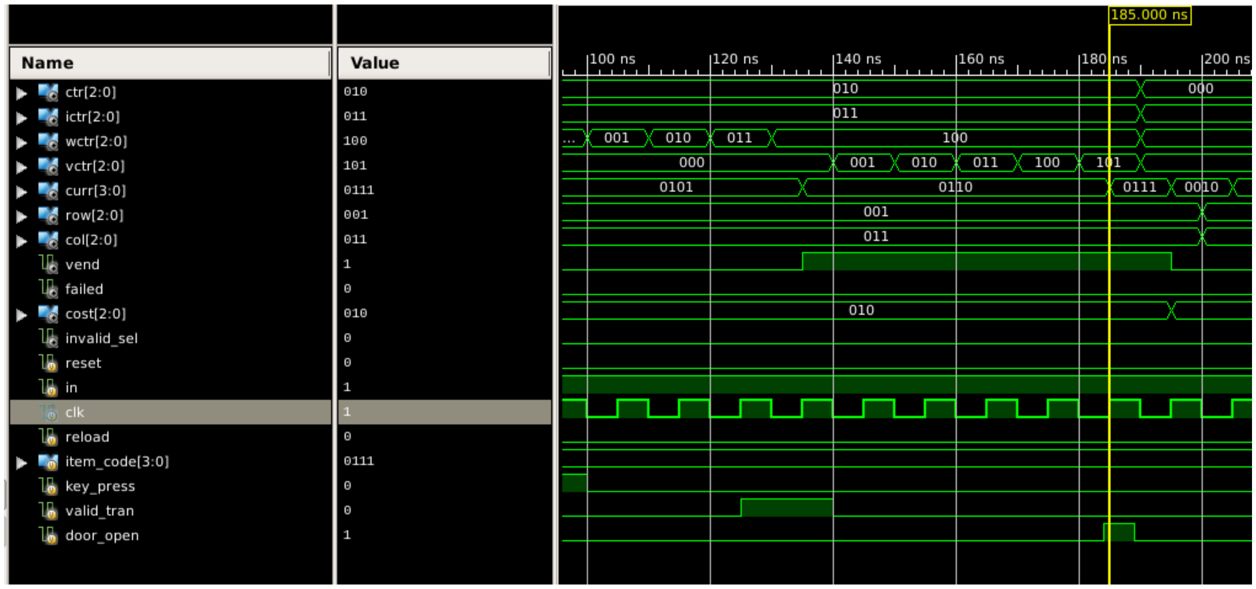
In the above test we wait 5 cycles for the valid_tran signal in WTRAN(0101), but since I never set this signal we will time out. This results in the failed signal being set to 1 at 145ns. Further, we notice that cost was also set upon leaving the CODE2 with a valid item_code (dig1 = 0, dig 2 = 7 -> sel = 7) . Sel 7 is also supported by row being 001 and col being 011 , which is position 7 in the list from 00 in upper left corner to 19 in bottom right corner as displayed in lab manual vending machine figure. The cost for item 7 is $2 shown in our lab manual. After fail, the next state goes to IDLE (0010) shown at 155ns.

TEST 9: User does not open door in 5 cycles



In the above test we successfully achieved the VEND(0110) state starting at 135ns and set the vend signal until the next transaction. A door open signal would trigger the next state where we would wait for the door to close before going to IDLE. However, in this case the door is not opened for 5 posedge clk cycles, so we move to IDLE as a result. We then reset output values in IDLE shown with vend going low at 185ns.

TEST 10: User opens door just before 5 posedge clks occur (up to last instant possible)



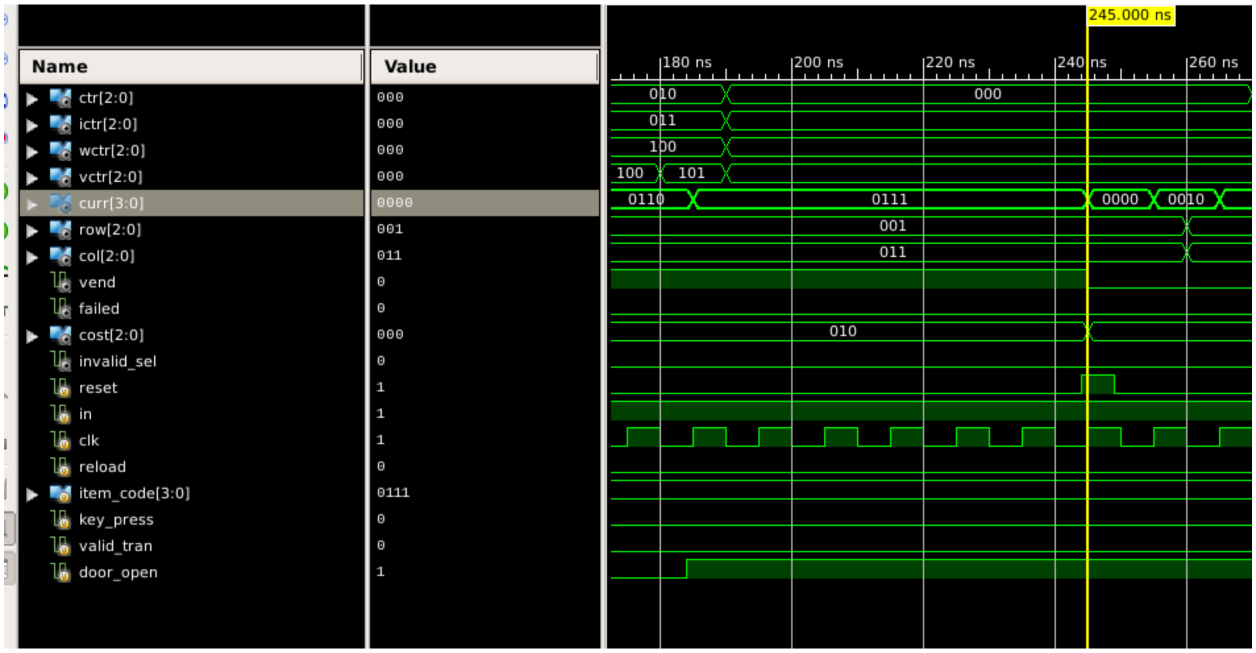
Name	Value
ctr[2:0]	010
ictr[2:0]	011
wctr[2:0]	100
vctr[2:0]	101
curr[3:0]	0111
row[2:0]	001
col[2:0]	011
vend	1
failed	0
cost[2:0]	010
invalid_sel	0
reset	0
in	1
clk	1
reload	0
item_code[3:0]	0111
key_press	0
valid_tran	0
door_open	1

This case is nearly the same as before, but we created a door open signal just before the 5thp pos edge of the clk in the VEND(0110) state. This causes a successful transition into the WCLOSE(0111) state. In this case too I have the door_open signal go low at 184ns, so WCLOSE registers this event going low and peacefully transitions back to IDLE (0010).

TEST 11: Mean kid keeps door open with a doorstopper for 5 cycles, then resets machine and skips away. Or they break the door close mechanism!!! – User can use this to get back to IDLE state without closing door, however door will never close so reset must always be used to get back to IDLE state.

Name: Chris Baker

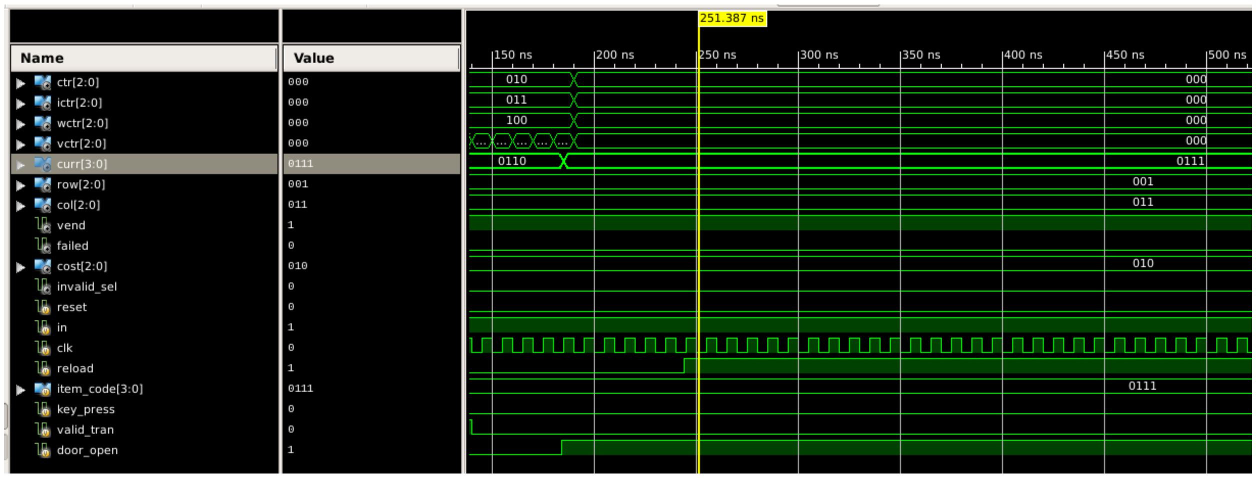
UID: 105.180.929



Name	Value
ctr[2:0]	000
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0000
row[2:0]	001
col[2:0]	011
vend	0
failed	0
cost[2:0]	000
invalid_sel	0
reset	1
in	1
clk	1
reload	0
item_code[3:0]	0111
key_press	0
valid_tran	0
door_open	1

This follows the previous example where WCLOSE(0111) is waiting on a door close signal. However, the door stays open starting at 184ns and continuing forever (since I left it set in the TestBench to simulate a door propped open or malfunction that is registered as a never closing door). This means that we can run through RESET(0000) to get back to the IDLE (0010) state. This is actually our only option with a broken or forced open door that is unable to close. Either that or we stay in door_open forever.

TEST 12: Door is stuck open and reset is broken too! Only one transaction then out of order. Employee even tries to reload the machine (is ineffective due to our transition logic and FSM), but is unable to and now has a personal issue with that mean kid.

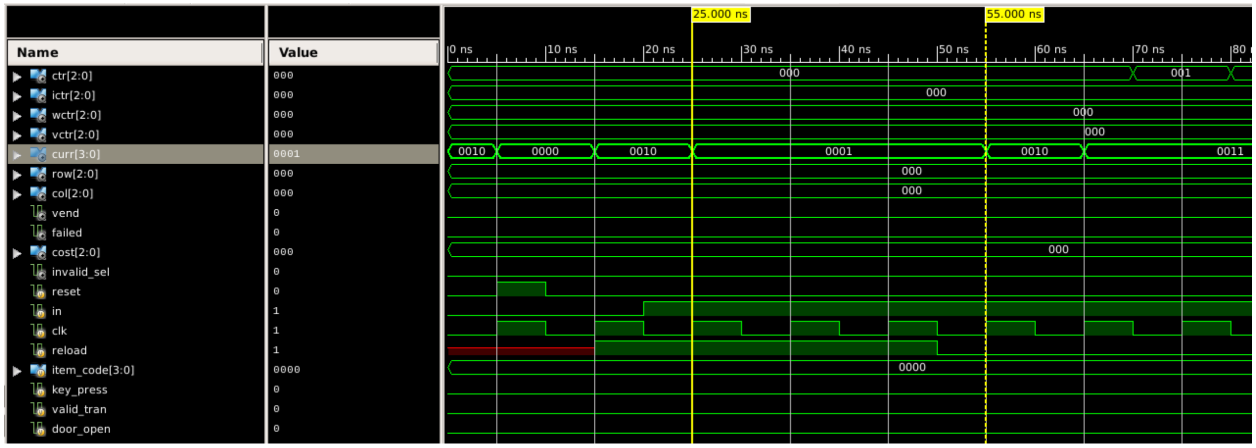


Name	Value
ctr[2:0]	000
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0111
row[2:0]	001
col[2:0]	011
vend	1
failed	0
cost[2:0]	010
invalid_sel	0
reset	0
in	1
clk	0
reload	1
item_code[3:0]	0111
key_press	0
valid_tran	0
door_open	1

Name: Chris Baker
UID: 105.180.929

In this photo above we see long lines continuing forever from the moment we move from VEND(0110) into WCLOSE(0111) at about 185ns . Notice that at the yellow line of 251ns we have even set the reload to high to see if that would incorrectly move us out of this predicament. Unfortunately for the user, RELOAD can only be performed from IDLE. Further, IDLE can only be reached via a RESET or a door_open signal going low in WCLOSE. Therefore, the machine performed one transaction and vended one item just before it became Out Of Order stuck in WCLOSE. If the machine can't be turned off or the plug is inaccessible then there is truly nothing a UCLA student would be able to do when staring at this vending machine. However, turning the machine off and back on if you can reach the plug may trigger its initial state set up again which could be useful to perform transactions only once every time if a vending machine is designed like that.

TEST 13: User inputs card while reload stays high to try and see if a new transaction will occur



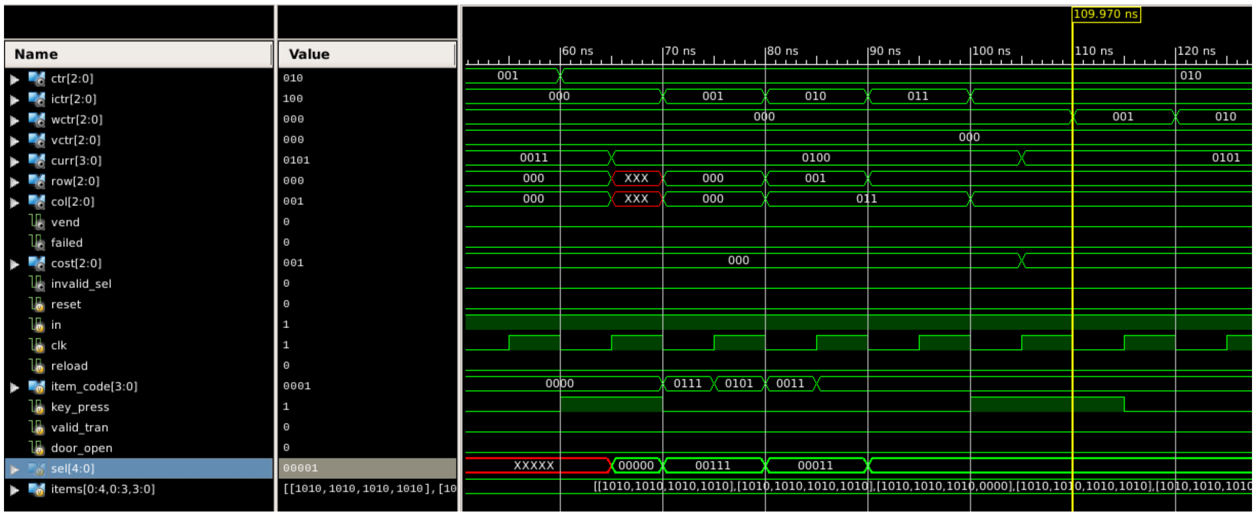
Name	Value
ctr[2:0]	000
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0001
row[2:0]	000
col[2:0]	000
vend	0
failed	0
cost[2:0]	000
invalid_sel	0
reset	0
in	1
clk	1
reload	1
item_code[3:0]	0000
key_press	0
valid_tran	0
door_open	0

In the photo above the reload signal goes on at 15ns and the card in signal goes on at 20ns. This causes us to move from IDLE(0010) to RELOAD (0001) at next posedge of clk. Notice that even though card in is high, we will not move from RELOAD until the reload signal goes low. Reload goes low at 55ns where we move back into IDLE after which we could then go to RESET, RELOAD, or C1 to start a transaction depending on which signals are on or off. In this case reload is now off by 55ns as well as reset, so the next state will be C1(0011).

TEST 14: Multiple Item_Codes entered with multiple Key_Presses attempted

Name: Chris Baker
UID: 105.180.929

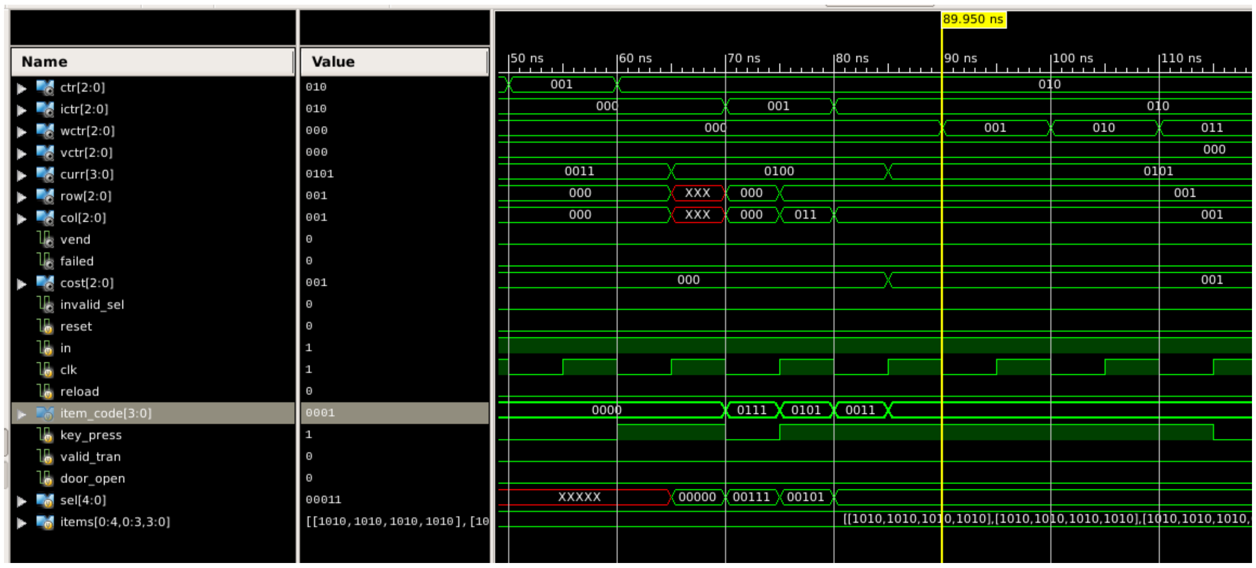
User punches in different item codes immediately to change selection (They wanted Cheesy Poofs, not Milk Duds) or thought they could get multiple selected items during one transaction.



Name	Value
ctr[2:0]	010
ictr[2:0]	100
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0101
row[2:0]	000
col[2:0]	001
vend	0
failed	0
cost[2:0]	001
invalid_sel	0
reset	0
in	1
clk	1
reload	0
item_code[3:0]	0001
key_press	1
valid_tran	0
door_open	0
sel[4:0]	00001
items[0:4,0:3,3:0]	[[1010,1010,1010,1010],[10

In this case I tried to set 4 item_code values within C2(0100) and test one key_press signal. In the example above key_press was high from 60ns to 70ns. While checking this I noted that my sel variable is set on the negedges of the clocks, so it did take on the various item_code values, but this variable is only used to set the cost for the 2d array items, determine item position, and make sure there is no value greater than 19 (a redundant safety check in my implementation). My main logic in the next case always block appropriately determines moving to WTRAN(0101) from C2 based on key_press =1 and a valid item_code (including and within 0 to 1 for C1, and 0 to 9 for C2, making item 00 to 19 possible). Thus, we did not see a keypress signal until 100ns which registered at the nededge of the clock making it that we would transition to WTRAN at the next posedge at 105ns. That is why regardless of having 4 item_codes input by the user we did not recognize any of them without the keypress signal. I had my test variable sel identify the value of the resulting two item_code inputs, but we do not act on that value without item_code passing each test individually in C1 and C2, so item_code still largely determines moving forward to wait for transaction to complete with valid transaction signal. Ultimately, in this case we recognize the item_code 0001 at 100ns and passed that forward as our second item_code to wtran and our first item_code was 0000 seen from before 70ns. That is why my sel variable shows 1 at 110ns and my item_code shows 1 at 100ns.

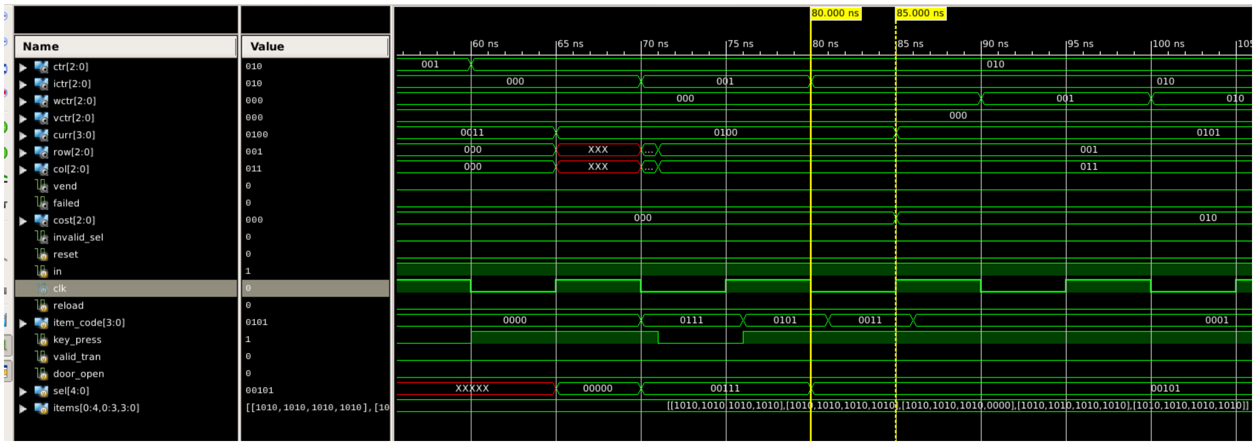
Name: Chris Baker
UID: 105.180.929



In order to more clearly demonstrate this idea we put in another keypress that started at 60ns and stayed on in addition to the key_press at 100ns. But as I expected the recognizing of an earlier key_press would make the later one irrelevant if it recognizes the user input and correctly moves to WTRAN. In this case the item_code was changed several times, but the values recognize in the testbench occurred at negedges of the clock. Further, we take the value during the key_press to run the testing to move forward to WTRAN or else we are testing until we cause an INVAL transition. For these reasons we see that even as we create new item_codes from 70ns with 0111 to 0101 to 0011 to 0001 at 85 ns, our sel value is 0011 when we are in WTRAN. As stated earlier we look for the key_press to determine our logic and set the sel value just prior to determining our next state. So at minimum we can see the next state 5ns before (at negedge) we make the transition to the new state (posedge clk). For this reason we took 0011 as our value since 0111 was being set right after that negedge of 70ns. If the negedge did not fall at 80ns on 0011 and instead fell on 0101 then we would have a different sel value and our resulting vending machine item would be 5 instead of 3.

Name: Chris Baker

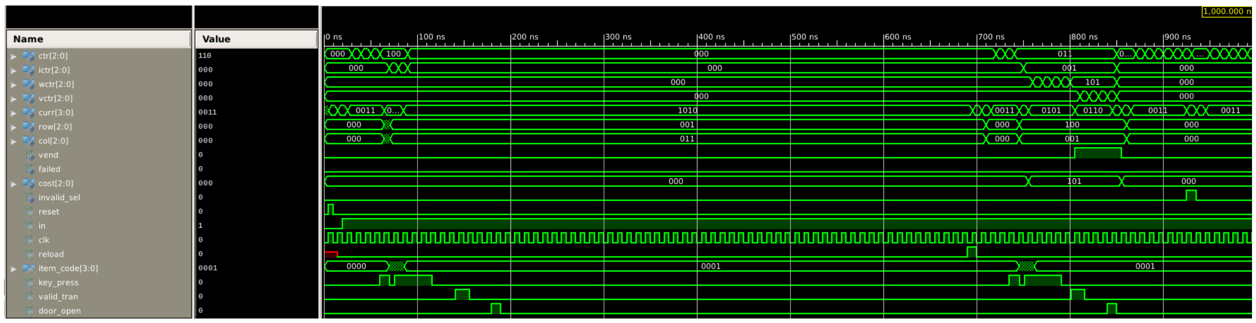
UID: 105.180.929



Name	Value
ctr[2:0]	010
ictr[2:0]	010
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0100
row[2:0]	001
col[2:0]	011
vend	0
failed	0
cost[2:0]	000
invalid_sel	0
reset	0
in	1
clk	0
reload	0
item_code[3:0]	0101
key_press	1
valid_tran	0
door_open	0
sel[4:0]	00101
items[0:4,0:3,3:0]	[[1010,1010,1010,1010],[10

This photo demonstrates that last hypothesis. Notice how at 80ns I now had the negedge of the clock falling at 0101 and the sel value is 0101. This value remains after the transition to WTRAN from C2 and can be verified through the transition occurring 5ns later at the psedge of the clk. This is because our next state was determined 5ns earlier as stated multiple times throughout my testing explanations. Because we had our key press, experienced negedge , and had valid item_code we were able to synthesize our combined 2 item_codes from C1 and C2 and transition into WTRAN with 0101 instead of the other multiple item_codes that were entered crazily during C2 and even 0001 which was entered after the transition which had no effect on the resulting sel value either.

Special optional test: Out Of Order



Name	Value
ctr[2:0]	110
ictr[2:0]	000
wctr[2:0]	000
vctr[2:0]	000
curr[3:0]	0011
row[2:0]	000
col[2:0]	000
vend	0
failed	0
cost[2:0]	000
invalid_sel	0
reset	0
in	1
clk	0
reload	0
item_code[3:0]	0001
key_press	0
valid_tran	0
door_open	0

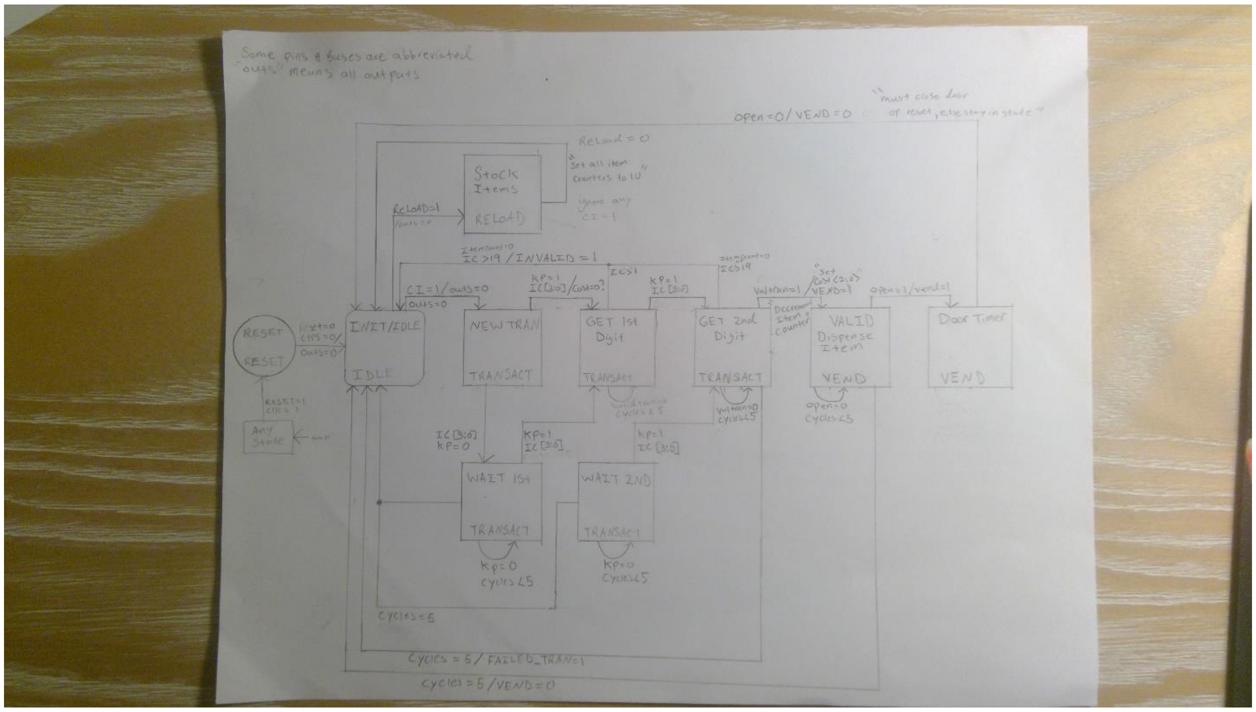
I added this shortly before submission as an interesting case. I start with initial IDLE state, but then I reset. There is no items initially. Then I added a spot in C2 to check if there are no items. If this is determined then the next state is OOO(1010) which is the long nearly 550ns period of curr shown above. In OOO there is only two options, go to reset, or go to reload. If you reset, then you can get back to idle, but this would still be an issue if you start another transaction without reloading from idle. Else you will be stuck in out of order unable to do anything with the machine until an employee reloads it, with the reload signal. After this we were able to proceed through states like WTRAN(0101) to

Vend(0110) then the two short periods at 840ns represent VEND and WCLOSE occurring properly and leading us back to IDLE (0010) to start another transaction. I used the reload signal to get us out of the OOO state and I then turned it off to go into IDLE to set up the last half of this picture that worked as intended.

Conclusion

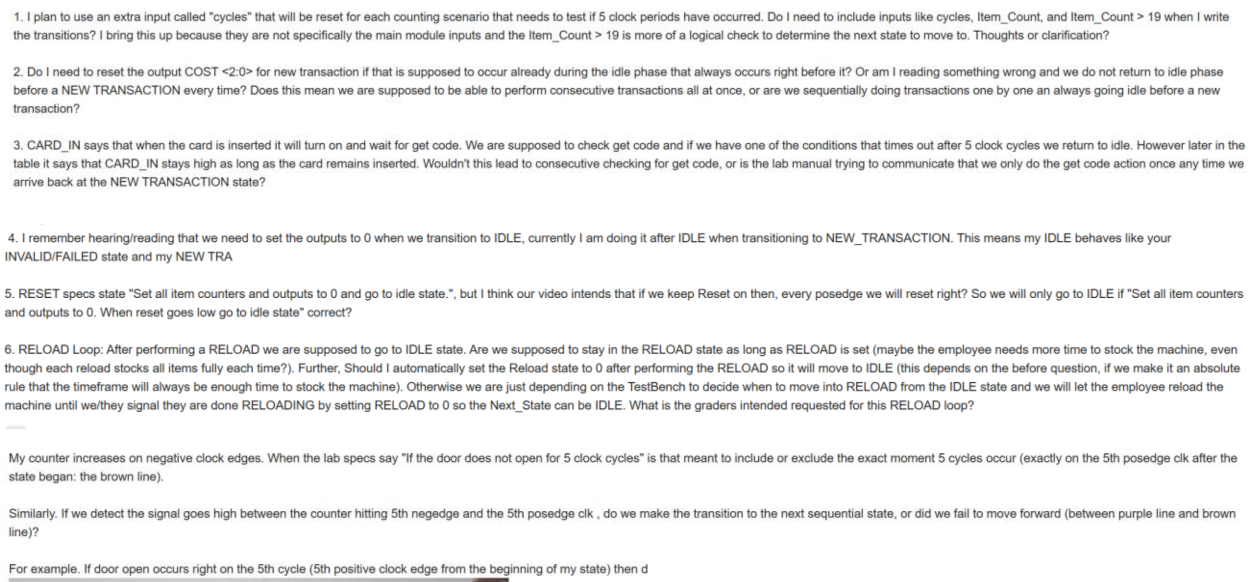
Essentially this FSM Vending Machine performs multiple functions that resemble a real vending machine. It has a reset state, which could be like turning it off and back on depending on the design, but is meant to place it into a general position when needed to reset any stuck issues and avoid a reboot if not needed. It also incorporates a reload so that after the employee fills the 10 items in each slot, then it will change those values and start decrementing each time a item is purchased. In Idle a transaction can be made and there are realistic features like checking valid inputs, waiting for the machine to register the keypress (sometimes we have to push a button several times) , and even timers that will prevent/create an invalid transaction. Further, there are also cycles to send the machine to the idle state even though an item has vended. This means someone can buy several items before retrieving them. It is also why we may see the price output and then when the machine goes idle we no longer see the price displayed just like real life vending machines that wait for user input after the vend. Finally it has a mechanism to determine if the door is open and to wait till it is closed before doing anything else (unless reset). This FSM really closely resembles the machines on campus at UCLA and it was a interesting, rewarding, but long and tedious project for me.

In this lab the difficulties I encountered was trying to create the FSM and the code on my own. I originally designed this FSM.



Name: Chris Baker
UID: 105.180.929

Which functioned nearly identical to my current one. I spent a whole day working on this FSM trying to make it neat and ready to go for when I would do this lab writeup, but I noticed after watching the video lecture on the 21st and 22nd that I may have issues with detecting the invalid and failed signals. I also was not sure of what to write for outputs and inputs necessarily. Those lectures provided a better framework to tackle this lab, but they only occurred a couple of days ago and I had used my time inefficiently trying to design this FSM that I scrapped. That was a major difficulty because I spent several hours making a new FSM, rereading the lab manual, and checking old logic vs current logic on a scratch paper where I was writing out my ideas. Largely I was able to overcome my problems through a ton of testing, staring at printouts of my code for long periods of time looking for issues, and even in the beginning I tossed out the whole code I worked on to go back to a discussion example that was previously working for basic transitions. I also reached out several times to my TA to discuss certain vague concepts in the Lab document. I also lost points last lab for following something written in the specs, but the TA cautioned me that I needed to include more then what the specs proposed there. Hopefully, the specs on this lab capture exactly what is meant to earn the grade that students are aiming for, but given the unusual circumstances this quarter and the possible future of more hybrid or online courses I think it is best to clearly communicate everything as best as possible due to the lack of physical interaction to clear up inconsistencies. In addition to the FAQS and other private student questions, here were my questions:



1. I plan to use an extra input called "cycles" that will be reset for each counting scenario that needs to test if 5 clock periods have occurred. Do I need to include inputs like cycles, Item_Count, and Item_Count > 19 when I write the transitions? I bring this up because they are not specifically the main module inputs and the Item_Count > 19 is more of a logical check to determine the next state to move to. Thoughts or clarification?

2. Do I need to reset the output COST <2:0> for new transaction if that is supposed to occur already during the idle phase that always occurs right before it? Or am I reading something wrong and we do not return to idle phase before a NEW TRANSACTION every time? Does this mean we are supposed to be able to perform consecutive transactions all at once, or are we sequentially doing transactions one by one an always going idle before a new transaction?

3. CARD_IN says that when the card is inserted it will turn on and wait for get code. We are supposed to check get code and if we have one of the conditions that times out after 5 clock cycles we return to idle. However later in the table it says that CARD_IN stays high as long as the card remains inserted. Wouldn't this lead to consecutive checking for get code, or is the lab manual trying to communicate that we only do the get code action once any time we arrive back at the NEW TRANSACTION state?

4. I remember hearing/reading that we need to set the outputs to 0 when we transition to IDLE, currently I am doing it after IDLE when transitioning to NEW_TRANSACTION. This means my IDLE behaves like your INVALID/FAILED state and my NEW TRA

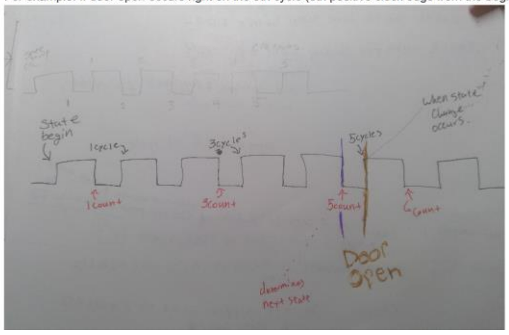
5. RESET specs state "Set all item counters and outputs to 0 and go to idle state.", but I think our video intends that if we keep Reset on then, every posedge we will reset right? So we will only go to IDLE if "Set all item counters and outputs to 0. When reset goes low go to idle state" correct?

6. RELOAD Loop: After performing a RELOAD we are supposed to go to IDLE state. Are we supposed to stay in the RELOAD state as long as RELOAD is set (maybe the employee needs more time to stock the machine, even though each reload stocks all items fully each time?). Further, Should I automatically set the Reload state to 0 after performing the RELOAD so it will move to IDLE (this depends on the before question, if we make it an absolute rule that the timeframe will always be enough time to stock the machine). Otherwise we are just depending on the TestBench to decide when to move into RELOAD from the IDLE state and we will let the employee reload the machine until we/they signal they are done RELOADING by setting RELOAD to 0 so the Next_State can be IDLE. What is the graders intended requested for this RELOAD loop?

My counter increases on negative clock edges. When the lab specs say "If the door does not open for 5 clock cycles" is that meant to include or exclude the exact moment 5 cycles occur (exactly on the 5th posedge clk after the state began: the brown line).

Similarly. If we detect the signal goes high between the counter hitting 5th negedge and the 5th posedge clk , do we make the transition to the next sequential state, or did we fail to move forward (between purple line and brown line)?

For example. If door open occurs right on the 5th cycle (5th positive clock edge from the beginning of my state) then d



Name: Chris Baker

UID: 105.180.929

COVID has impacted many classes this quarter making things less clear, affecting deadlines, and understating the extra effort applied to courses. I really did find this lab to be amusing and a learning experience, but I also felt like I was starting to fall behind. I worked on it for about two days before I felt the discussions, FAQs, and other resources helped me to efficiently get more done. I understand this may be a new lab created in response to COVID, so I appreciate all the effort of the Tas and the instructor. Hopefully, it will be easier to distribute the recent discussions earlier on to students. I think I would have finished a day or two earlier and would have managed my time better if that was the case, but I also see the reward in all the extra effort and time I put into this lab, but it also worries me with all my other course work still left to work on. Thank you all for your effort, time, and understanding. I am also incredibly happy that my TA responded frequently to my emails to help me out to overcome some of these issues. I was impressed with the responsiveness and assistance provided by the TAs, very well done!