Ryan Kirkpatrick

ECE 156A Hw #5

11/13/17

Introduction:

In this homework, we take a large complex circuit of a vending machine and break it into several different unit modules, including a coin sensor, a piggy bank, a purchase manager, and a display manager. This allows us to perform tests on both the units, and on the vending machine as a whole to speed the process of finding bugs and correctly implementing the circuit.

Procedure:

The vending machine is stored in the vendingMachine.v file and the unit test bench for the vending machine is in the vendingMachine_tb.v file. The vending machine calls on 4 different modules: the coin sensor which was created for HW4, the piggy bank which stores the value of the coins outputted by the coin sensor, the purchase manager which takes input of which item to buy and when to buy it, and the display manager which displays how much money has been inputted or the item being purchased.

The piggy bank is stored in the piggyBank.v file and the unit test bench for the piggy bank is in piggyBank_tb.v. The piggy bank has input for the different coins and the different items, and output for the total amount of money stored in the bank, capping off at 255 cents. When coins are inputted, the amount of money in the bank increases, and when an item is chosen, the amount of money is decreased by the amount necessary to buy the item.

The purchase manager is stored in the purchaseManager.v file and the unit test bench for the purchase manager is stored in the purchaseManager_tb.v file. The purchase manager takes signals for buy and product id as well as the credit output of the piggy bank in as input, and outputs signals for identifying the item bought as well as an error signal for identifying whether enough money is available for purchasing the item.

The display manager is stored in the sevenSegDispMngr.v file and the unit test bench for display manager is stored in the sevenSegDispMngr_tb.v file. The display manager uses the seven segment decoder that was created in HW2 to decode either the current credit that is stored in the piggy bank, or to decode the item that is being bought and display a code for it for 6 clock cycles.

The coin sensor and test bench are stored in coinSensor.v and coinSensor_tb.v, respectively. The coin sensor reads in a serial input of the coin's diameter and determines the coin and outputs high on the wire the coin represents. The seven segment decoder and test bench are stored in segment_decoder_7bit.v and segment_decoder_7bit_tb.v. and takes a 4 bit input and outputs a 7 bit output that is meant to be hooked to a seven segment display to display the hexadecimal value of the input.

The wiring of the vending machine can be seen on the following page in figure 1.

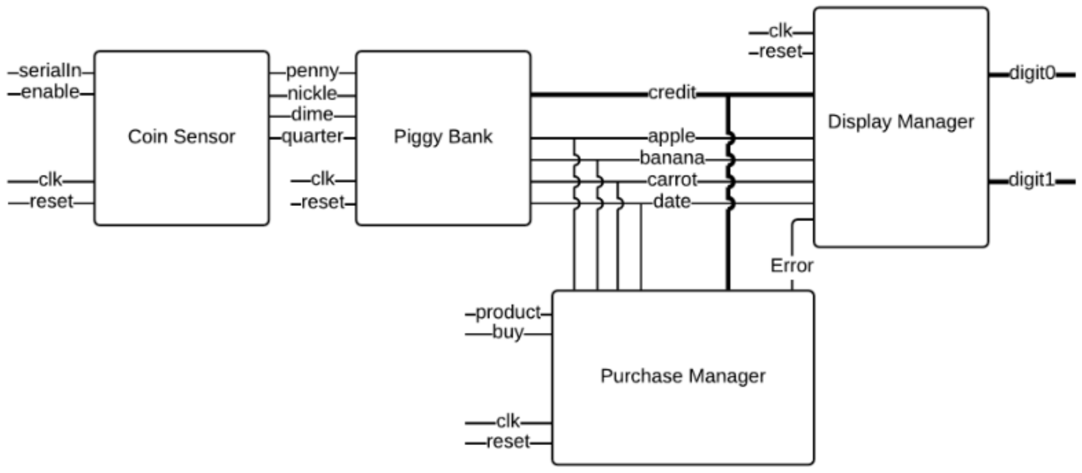

Figure 1: *Vending Machine Block Diagram*

Results:

The unit tests for the individual modules are used as tools to fix internal bugs within each of the units of the program. If the test bench for the vending machine works properly, it implies that the units all work properly and wired together properly. Therefore, this report only covers the results from the vending machine test bench.

To understand the waveform outputs of the test bench, you need to be able to read the values outputted by the digit1, and digit0 nets coming as output of the vending machine. Since digit1 and digit0 hold 7-bit values to be read as input to a seven-segment decoder, certain values correspond to hexadecimal values that are meant to be displayed. The table below shows the conversion between the hexadecimal values that are meant to be displayed and the hexadecimal code stored in the digit nets. For example, if the waveform reads digit1 as h01 and digit0 as h12, the value stored in credit is h02 or in decimal, 2.

HexDisplay	0	1	2	3	4	5	6	7
HexDigit	h01	h4f	h12	h06	h4c	h24	h20	h0f

HexDisplay	8	9	a	b	c	d	e	f
HexDigit	h00	h04	h08	h60	h31	h42	h30	h38

Table 1: Table to convert the displayed hex value and the hex value code of digit1 and digit0

To save on the number of tests that need to be created and to be briefer in this report, some of the corner cases, such as checking to make sure inputting the all the different coins into the piggy bank doesn't cause an overflow, are only tested with one input. Within the code, these blocks are replicated almost identically, so if one works, they should all work.

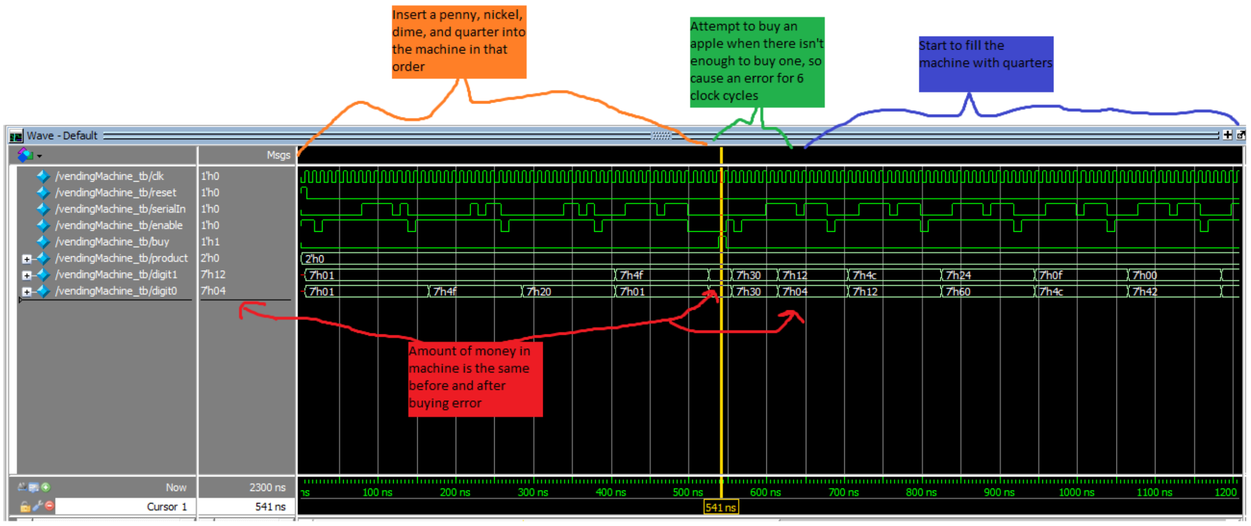
Insert a penny, nickel, dime, and quarter into the machine in that order
Attempt to buy an apple when there isn't enough to buy one, so cause an error for 6 clock cycles
Start to fill the machine with quarters
Amount of money in machine is the same before and after buying error
Wave - Default
Msgs
/vendingMachine_tb/clk 1'h0
/vendingMachine_tb/reset 1'h0
/vendingMachine_tb/serialIn 1'h0
/vendingMachine_tb/enable 1'h0
/vendingMachine_tb/buy 1'h1
/vendingMachine_tb/product 2'h0
/vendingMachine_tb/digit1 7'h12
/vendingMachine_tb/digit0 7'h04
2'h0
7'h01 7'h4f 7'h30 7'h12 7'h4c 7'h24 7'h0f 7'h00
7'h01 7'h4f 7'h20 7'h01 7'h30 7'h04 7'h12 7'h60 7'h4c 7'h42
Now 2300 ns
Cursor 1 541 ns
100 ns 200 ns 300 ns 400 ns 500 ns 600 ns 700 ns 800 ns 900 ns 1000 ns 1100 ns 1200
541 ns

Figure 2: First half of the vending machine test bench wave form

In figure 2, the first half of the vending machine test bench wave form is displayed. While not labeled, the reset appears on the first clock edge to initialize the circuit, and could be used later to reset all values relevant values to 0. Contained in the orange portion of the figure is the insertion of different coins into the coin sensor in the order penny, nickel, dime, and then quarter. If you were to decode the digit1 and digit0 seen on the graph, the credit values go from 0 to 1 to 6 to 16 to 41, which is expected from the coin insertion order.

The green portion of the figure shows an attempt at buying product 0, an apple, when not enough money has been inserted into the machine, this causes the digit outputs to change to values that correspond to EE being shown for 6 clock cycles. As seen in the red portion of the figure, this attempt at buying a product doesn't affect the value stored in credit in the machine.

In order to show that the machine maxes out at 255 credit, the blue portion of the figure is me incrementing the credit by 25 by inserting quarters. However, the credits don't max out until figure 3.

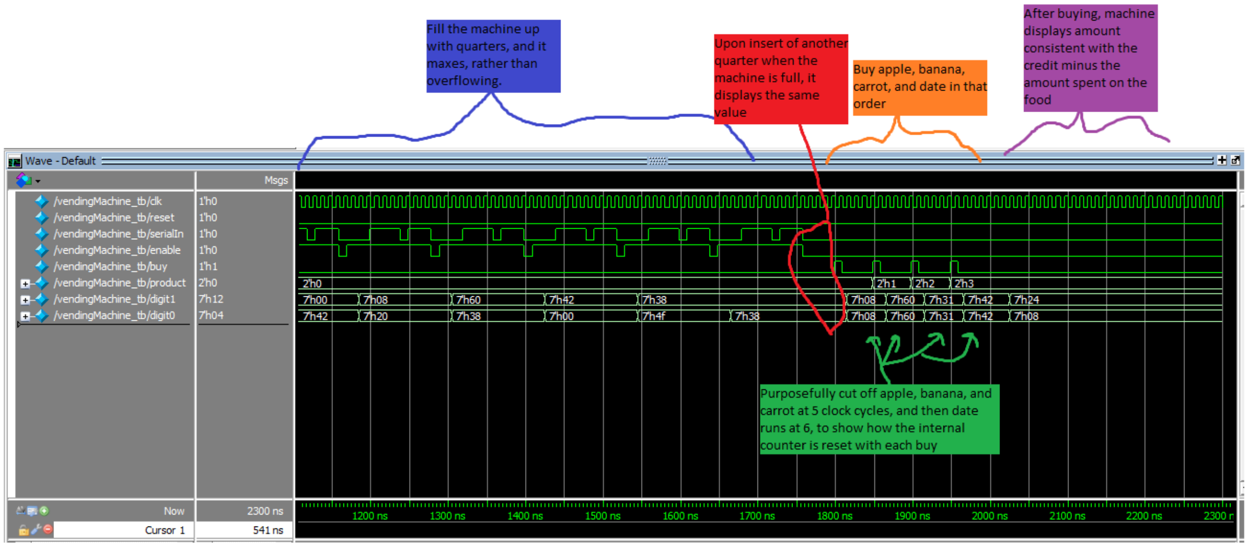
Fill the machine up with quarters, and it maxes, rather than overflowing.
Upon insert of another quarter when the machine is full, it displays the same value
Buy apple, banana, carrot, and date in that order
After buying, machine displays amount consistent with the credit minus the amount spent on the food
Purposefully cut off apple, banana, and carrot at 5 clock cycles, and then date runs at 6, to show how the internal counter is reset with each buy
Wave - Default
Msgs
/vendingMachine_tb/clk 1'h0
/vendingMachine_tb/reset 1'h0
/vendingMachine_tb/serialIn 1'h0
/vendingMachine_tb/enable 1'h0
/vendingMachine_tb/buy 1'h1
/vendingMachine_tb/product 2'h0
/vendingMachine_tb/digit1 7'h12
/vendingMachine_tb/digit0 7'h04
2'h0 2'h1 2'h2 2'h3
7'h00 7'h08 7'h60 7'h42 7'h38 7'h08 7'h60 7'h31 7'h42 7'h24
7'h42 7'h20 7'h38 7'h00 7'h4f 7'h38 7'h08 7'h60 7'h31 7'h42 7'h08
Now 2300 ns
Cursor 1 541 ns
1200 ns 1300 ns 1400 ns 1500 ns 1600 ns 1700 ns 1800 ns 1900 ns 2000 ns 2100 ns 2200 ns 2300 r

Figure 3: The second half of the vending machine test bench

The blue portion of figure 3 shows the rest of the incrementing of the credit counter by inserting quarters. The last bit of the blue portion shows digit1/digit0 increase from h38/h4f to h38/h38. This corresponds to the credit counter storing hF1 and increasing to hFF. This means that although enough money was inserted to the counter to overflow, the value stopped at the max of 255 or hFF. Testing whether this works for pennies, nickels, and dimes is pedantic and overzealous, because they were programmed in the same fashion within 6 lines of code each, so by testing 1 you basically test them all. Also since each module is tested individually, you can assume that such corner cases are extensively tested in the individual modules. Furthermore, the red portion of the graph shows that inserting an extra quarter after the max has been reached doesn't change the value of the counter. Since it doesn't change for the quarter, it is assumed to not change for the other coins as well.

The orange portion of figure 3 shows each product being bought and their corresponding display outputs being held for several clock cycles. As seen with the green portion, I timed the tests such that for the apple, banana, and carrot display outputs, they would be shown only for 5 clock cycles, before being cut off by the next product being bought. For the date, I let the counter run out, so 'DD' should be displayed for a total of 6 clock cycles.

The purple portion shows that the after the items were bought, the total value stored in the credit counter was 255-(75+40+30+20) = 90 or hFF → h5a or h38/h38 → h24/h08, which are the correct values. This value is then stored in the credit for the rest of the machines existence, because it is no longer being interacted with.

<u>Conclusion:</u>

The purpose of this lab was to create several test benches to test the internal modules of the vending machine extensively, and then create a test bench for the vending machine to make sure all the modules work together properly. This way of splitting up a large project into several smaller projects allows for quicker testing by allowing you to more accurately pinpoint the cause of a bug.