CMPE 260
Digital System Design II
Laboratory Exercise 4: Vending Machine
Revised: 06-22-2016

This exercise develops a vending machine controller. The objective of this exercise is to become familiar with the use of state machines in hardware and how they can be applied to real world problems. A controller is written and tested in VHDL, then put into use on an FPGA.

Background

The actions taken by a computer as a result of inputs often depend on previous inputs. When this is the case, a state machine is a good choice, as it can make use of hardware such as D flip flops for faster execution. However, given their popularity, the conversion of the logic is performed by the synthesis tools, leaving the engineer to focus on the design of the state machine.

Prelab Work

- Design a state machine that is capable of performing the tasks required by the coin controller. Use as few states as necessary. Draw the state diagram. **Hint:** At least one wait state is required.
- Read the given files from myCourses and understand their functionality.

Program Specification

The design must meet the following specifications and use all components listed at least once. Some will be used multiple times.

Coin Controller

The coin controller has the following features:

- Rules:
  - Process sensitivity lists, with the exception of the state machine, are limited to `clk` and `rst` **only**.
  - Signals must be std_logic, std_logic_vector, or enumerated (for states).
  - All if statements must have an else clause.
  - All case statements must have an others clause.

  The purpose of these rules is to ensure that the state machine cannot enter any unexpected states, and if it does, it will recover without requiring a reset.

- Inputs (Active high and one bit unless noted):
  - `clk`: Vending machine clock. Components should be rising edge triggered.
  - `rst`: Resets the vending machine, including the total deposited. Synchronous, active low.
  - `Qp`: A quarter has been deposited.
  - `Dp`: A dime has been deposited.
  - `Np`: A nickel has been deposited.
  - `soda_req`: A soda is being requested.
  - `soda_price`(4 bits): The code for the soda that the user wants. See Table 1.
- Outputs:
  - `drop_soda`: The requested soda should be dropped. Also indicates that the `amt_dep` was enough.
  - `amt_err`: The `amt_dep` was not enough and an error should be shown to the user.
  - `amt_dep`: The amount of money that the user has put into the machine.

Notes

- The `amt_dep` should not be cleared out after every transaction. Instead, any extra money should be carried over towards the purchasing of the next soda.

Table 1: `soda_price` to monetary amount conversion

Code	Price
0000	$0.55
0001	$0.85
0010	$0.95
0011	$1.25
0100	$1.35
0101	$1.50
0110	$2.25
0111	$2.50
1000	$3.00
1001-1111	Reserved

Seven Segment Decoder

The seven segment decoder is responsible for producing the pulses necessary to display the `amt_dep` on the seven segment display.

- Inputs:
  - `BCD` (12 bits): A three digit number in BCD.
- Outputs:

- `hund_disp_n` (7 bits): Binary representing the segments that should be on or off for the hundreds digit. Since the display has a common anode, segments are active low.
- `tens_disp_n` (7 bits): Same as the hund_disp_n, but for the tens place.
- `ones_disp_n` (7 bits): Same as the hund_disp_n, but for the ones place.

Notes

- It would be a good idea to create either an entity or function that is capable of converting a four bit vector in BCD to the seven segment seven bit vector.

Vending Machine Controller

This module will tie the coin controller to the seven segment decoder, as shown in Figure 1.

- Inputs:
  - Same as the coin controller's.
- Outputs:
  - All of those from seven segment decoder.
  - All of those from coin controller with the exception of `amt_dep`.
  - There should be a grand total of five.

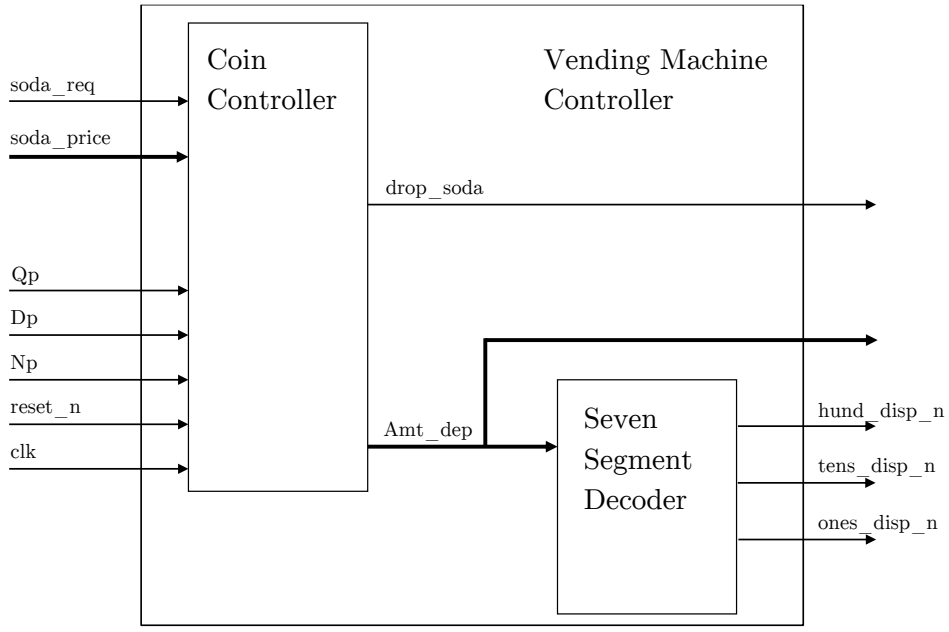

Figure 1: Vending Machine Controller Block Diagram

Given Code

Download all mentioned files from myCourses. The purpose of this section is to explain how to use them. For implementation details, read the files.

Usr_interaction.vhd

This is the top level of the vending machine and thus the one that will be instantiated in the test bench. It is responsible for connecting several components including the vending machine controller, coin_rx, and seven_seg_disp. Because of this, it is very important that the given port descriptions are used.

bin_bcd.vhd

The provides a function the will convert a binary vector to a BCD vector. To use, see Listing 1.

Listing 1: Usage for bin_bcd function

```
use work.bin_bcd.all;
...
-- Call function
bcd_vector <= bin_bcd(binary_vector);
```

coin_rx.vhd

This component will handle the debouncing of the buttons to ensure that two coins are not counted when only one was inserted. As a part of this, it outputs a signal for each received coin (via `Qp`, `Dp`, and `Np`) that is significantly longer than a single clock cycle. This must be accounted for when designing the coin controller.

seven_seg_disp.vhd

The seven segment display is not as simple as turning on or off an LED. This module simplifies away those issues by taking in the vectors from the seven segment decoder and outputting the necessary signals for the display. It is important to note that seeing changes in the signals from this module during simulation will take a very long time. See the testing section for more details.

Test Bench

One test bench is required, however a second may be a good idea.

Required Test Bench

This test bench should instantiate the Usr_interaction component for complete testing of the vending machine. Some scenarios to test:

- Resetting the vending machine.
- Selecting a soda.
- Inserting coins.
- Requesting a soda with too little money.

- Requesting a soda with exactly the right amount.
- Requesting a soda with too much money.

No assert statements are required. Note that simulating this test bench will take a long time, as the display takes approximately 21 ms to show all four values with a 100 MHz clock. However, the display must still be tested.

Optional Test Bench

To get around the long simulation time in the previous test bench, several solutions are viable. One is simply running the simulation for less time. However, if the test bench contains wait statements as to show all displayed values, the end of the test will not be performed. An alternative would be to make a second test bench without the wait statements. Another, more elegant solution would be to wrap the problematic wait statements in if statements and then using a constant that could be easily changed to toggle the longer waits.

Lab Procedure

1. Create a new directory and Xilinx ISE project for this exercise.
2. Download coin_rx.vhd, Usr_interaction.vhd, bin_bcd.vhd, and seven_seg_disp.vhd from myCourses and add them to the project.
3. Write a properly commented and properly formatted VHDL program according to the preceding specifications.
4. Test the design using a behavioral simulation. Be sure to simulate long enough that the display signals are verifiable.
5. Synthesize the design and retest it using a post route simulation. Once again, the display should be verified.
6. Prepare the design for hardware, following the procedure laid out in Exercise One. Use Nexys_3_IO.pdf from myCourses to help plan the layout. Pay careful attention to the ports of Usr_interaction, especially the coins, as they have both inputs and outputs. The outputs should be connected to LEDs, while the inputs should be attached to buttons.
7. Program the board and verify that it is working.
8. Demonstrate steps 4, 5, and 7 to the instructor, who will sign the grading sheet.
9. Screen shot the appropriate waveforms for the report, making sure that they will be readable. Put the `amt_dep` signal on the waveforms as well, since the seven segment signals do not have to be present.
10. Submit the source code and report online to the appropriate myCourses dropbox.

Lab Report

Write a report that meets the rules of professional technical writing and follows the lab report format on myCourses. Additional items that must be included:

- A state diagram, computer drawn and not taken from ModelSim
- Area used, which is reported via number of occupied slices, FFs used, and LUTs used.

- Timing results (best case achievable). No code modification should be required since a clock was used.
- Marked simulation results.