CS6760 – DDV – Verilog Assignment 1

Deadline for Submission: 23:59hrs on the 24th February 2016 (3 weeks)

Submission Guidelines:

- Please submit your Verilog design and the test-bench files in a .tar.gz compressed format.
- The compressed file name should include the group number of the team. For eg. Group10.tar.gz or Group2.tar.gz
- Late submissions will face penalty.
- Plagiarism will face penalty.
- A viva/demo will be conducted for each group.

1. BEVERAGE VENDING MACHINE

Problem Statement:

Design a vending machine where a user can order a tea, cold-water or coffee after inserting an amount Rs. 20/-, Rs. 20/- and Rs. 30/-, respectively, for each of these items. We assume that the amount of money that can be inserted at a time is Rs. 10/-. So, if the user wants to enter higher amount (multiple of Rs. 10/-), then (s)he has to enter it multiple number of times. The user is also provided with a return money option so that at the end of his/her transaction (s)he can take the remaining money (if more was inserted). The user can make multiple transactions (and get various items any number of times) until (s)he wants to take the return money, which basically points the end of his/her operation. However, it is also assumed that at any point of time the inserted or the remaining money shall not exceed Rs. 30/-, since the user is acquainted with the fact that maximum valued item price is Rs. 30/-.

Design your own functionalities in the test-bench for simulating the components as well as the overall system. Also, please make valid/necessary assumptions (if you require).

Inputs:

1. mi: input indicating Rs.10/- has been inserted
2. ti: input indicating that user wants tea
3. wi: input indicating that user wants water
4. ci: input indicating that user wants coffee
(5) \( rr \): input indicating that user wants the remaining money (end of his/her transaction)

Outputs:

(1) \( to \): output indicating that tea has been served to the user
(2) \( wo \): output indicating that water has been served to the user
(3) \( co \): output indicating that coffee has been served to the user
(4) \( rro \): output indicating the returned money
   a. \( rro = 0 \), if no money is returned
   b. \( rro = 1 \), if Rs. 10/- is returned
   c. \( rro = 2 \), if Rs. 20/- is returned
2. TRAFFIC LIGHT CONTROLLER

Problem Statement:

A little used farm road intersects a multi-lane highway; a traffic light controls the traffic at the intersection. The light controller is implemented to maximize the time the highway light remains green. The main module ties together a timer, a sensor, a farm light control and a highway control sub-modules. The basic block diagram is shown in the following:

The timer sub-module implements a timer that outputs “short” and "long" timeouts. The highway light stays green for at least “long” time. Any time after “long” time, if there is a car waiting on the farm road, then the farm light turns green. The farm light remains green until there are no more cars on the farm-road, or until the long timer expires. The yellow light for both directions stays yellow for “short” time. Note that only a single timer is used for both the farm road and highway controllers. In theory, this could lead to conflicts; as implemented, such conflicts are avoided. From the START state, the timer produces the signal “short” after a nondeterministic amount of time. The signal “short” remains asserted until the timer is reset (via the signal “start”). From the SHORT state, the timer produces the signal “long” after a nondeterministic amount of time. The signal “long” remains asserted until the timer is reset. Notice that the use of non-determinism in the description of the timer models an infinite number of actual implementations, each with a different set-up for the “short” and “long” periods.

The farm light stays RED until it is enabled by the highway control. At this point, it resets the timer, and moves to GREEN. It stays in GREEN until there are no cars, or the long timer expires. At this point, it moves to YELLOW and resets the timer. It stays in YELLOW until the short timer expires. At this point, it moves to RED and enables the highway controller.

The highway light stays RED until it is enabled by the farm control. At this point, it resets the timer, and moves to GREEN. It stays in GREEN until there are cars on the farm road and the long timer expires. At this point, it moves to YELLOW and resets the timer. It stays in YELLOW until the short timer expires. At this point, it moves to RED and enables the farm controller.

There is a single sensor that detects the presence of a car in either direction of the farm road. At each clock tick, it non-deterministically reports that a car is present or not.
Design your own functionalities in the test-bench for simulating the components as well as the overall system. Also, please make valid/necessary assumptions (if you require).
3. PRIORITY CACHE ACCESS LOGIC

Problem Statement:

Your goal is to develop a cache memory access logic that allows storage and retrieval by four different devices, M1, M2, M3 and M4. The architectural interface of this block is shown in the above Figure. The description of the interface signals are as follows.

1. **Request lines.** The block receives four request lines, namely r1, r2, r3 and r4 respectively from M1, M2, M3 and M4.

2. **Grant lines.** The block arbitrates between the two high-priority request lines to assert g1 or g2, and between the low-priority request lines to assert g3 or g4.

3. **Cache ready lines.** The line d1 indicates that the page requested by the high-priority device is available in the cache. If there is a cache hit, then d1 is asserted within two cycles. Otherwise, d1 is asserted after the page has been fetched from memory. Similarly, the line d2 is asserted when the page requested by M3 or M4 becomes available in the cache.

4. **Data and address lines.** The data and address lines are used to float the cache address and store / retrieve the data.

Design Requirements:

There are many architectural requirements of this block, but for simplicity we will choose the following property, which defines the notion of priority.

*M1 and M2 have higher priority over M3 and M4.* A page requested by M1 or M2 is either served two cycles after the request (when we have a cache hit), or is served in some future cycle (when we have a cache miss). In the later case, *no page should be served to the low priority devices in the intermediate cycles.*

Design your own functionalities in the test-bench for simulating the components as well as the overall system. Also, please make valid/necessary assumptions (if you require).
4. ELEVATOR CONTROLLER SYSTEM

Problem Statement:

Write a Verilog design to describe the elevator system for a 4-floor building. The design includes modules both for the physical system (reservation buttons, cabin, door), and for the controller. It also shows the connection between modules. The objective of the exercise is to complete the program, and write a testbench to simulate and check that the given requirements are satisfied. You will have to formalize the transition relation for the existing variables, possibly introduce additional variables and definitions, write testbench for individual modules as well as the overall system, and make sure that the design satisfies them.

Components of Elevator Controller System:

**Elevator Door**

The cabin is also equipped with a door, that can be either open or closed. The door can receive either open, close or nop commands from the controller, and it responds opening, closing, or preserving the current state.

Design Requirements:
- The controller can issue an open command only if the door is closed.
- The controller can issue a close command only if the door is open.

**Elevator Cabin**

The cabin can be at any floor between 1 and 4. It is equipped with an engine that has a direction of motion, that can be either standing, up or down. The engine can receive one of the following commands: nop, in which case it does not change status; stop, in which case it becomes standing; up (down), in which case it goes up (down).

Design Requirements:
- The controller can issue a stop command only if direction is up or down
- The controller can issue a move command only if direction is standing
- The cabin can move up only if the floor is not 4
- The cabin can move down only if the floor is not 1

**Elevator Button**

For each floor there is a button to request service, that can be pressed. A pressed button stays pressed unless reset by the controller. A button that is not pressed can become pressed nondeterministically.

Design Requirement: The controller must not reset a button that is not pressed.

**Elevator Controller**
The controller takes in input (as sensory signals) the floor and the direction of motion of the cabin, the status of the door, and the status of the four buttons. It decides the controls to the engine, to the door and to the buttons. The controller must also satisfy the following requirements.

Design Requirements:
- No button can reach a state where it remains pressed forever.
- No pressed button can be reset until the cabin stops at the corresponding floor and opens the door.
- A button must be reset as soon as the cabin stops at the corresponding floor with the door open.
- The cabin can move only when the door is closed.
- If no button is pressed, the controller must issue no commands and the cabin must be standing.

Design your own functionalities in the test-bench for simulating the components as well as the overall system. Also, please make valid/necessary assumptions (if you require).
An Automated Teller Machine (ATMs) is a computerized electronic device that helps the customers of a financial institution to access financial transactions in a public place without any help from others. It is a combination of both hardware (card-swipe, keypad, display, etc) and software (operating software. In this assignment you will deal with creating a basic Moore machine for the hardware controller of the ATM.

**Problem Statement:**

The controller starts by reading in the id of the card-swiped. This will be a 16-bit number input to the controller. If this card id is not present in the database then the controller should abandon the session and eject the card. On a valid card the controller requests for a 4-digit (16-bit) pin. This again is verified against the database of the card details. Once accepted, the controller will ask the user for the mode of transaction desired:

- Withdrawal – Request the amount to be withdrawn
- Deposit – Request the amount to be deposited
- Transfer – Amount to transfer and account number (10-digit : 40 bits)

If withdrawal is chosen then the appropriate cash needs to be sent out to the customer. If deposit is chosen the cash needs to be inserted into the controller and verified if it matches. If transfer is chosen then the account number is verified if it exists. If the account number is wrong then the card is ejected and session is closed. If the account is verified then a successful transaction is done.

At the end of a successful transaction the customer is given an option if he wishes to perform further transactions or not. If not, then the card is ejected and session is closed. Also during any point of the transaction if the cancel button is pressed, the card the ejected and session is closed.

**Basic Inputs:**

- Card id (16-bit)
- Pin (4-digit)
- Amount to withdraw/ deposit/ transfer (4-digits)
- Cash deposited to the controller (4 digits)
- Transfer account number (10 digits)

**Basic Outputs**

- Transaction Successful
- Output cash for withdrawal
- Card Ejected/ Session Over
- Transaction Canceled