CE2003 Laboratory 1
Simple Vending Machine

You are to design a finite state machine controller for a vending machine. The vending machine dispenses healthy muesli bars. It accepts only 50 cent and one dollar coins and a muesli bar costs exactly one dollar. If an excess amount is entered, for example 50 cents followed by one dollar, the transaction is rejected and all coins are returned. The FSM will also refund the inserted amount if the cancel input is asserted in any appropriate state. If the coins are returned (by the coin mechanism upon it getting a *money_return* signal from the FSM), the FSM will immediately return to the default state (the reset state). The FSM machine should vend the product as soon as a sufficient amount ($1) has been inserted and wait for a reset to return to the default state. The reset would normally be provided by the vending mechanism after it has dispensed the product, however in this example, we will do this manually.

The FSM has 3 inputs (*fifty*, *dollar* and *cancel*) in addition to *clk* and *rst*, and 4 outputs (*st*, *insert_coin*, *money_return* and *dispense*). An input is represented by a single cycle pulse on the corresponding input.

Task 1 (Pre-lab)

Sketch the state diagram that represents the above behaviour. You should use a Moore machine, where the outputs depend only on the current state. The most straightforward way is to think of the states as representing the amount inserted so far. The end state representing product vending should not have any transitions. We also assume only one input is asserted at any point in time. You should have 4 states in total (one of which is the vending end state). Verify your state diagram is identical to the one given in Fig. 2.

Using a text editor, implement the FSM in Verilog. Call it *lab1_FSM*. Your module should have five inputs and 4 outputs (including the state). See top_FSM.v.

1. Declare the module and the port list. Declare current state (*st*) with sufficient wordlength.
2. Use the parameter keyword to define names for the four states and their assignment, as:

   parameter INIT=0, S50c=1, VEND=2, RETURN=3;
3. Declare the next state (*nst*) variable of sufficient wordlength.
4. Implement a synchronous always block to take the value of next state and assign it to the current state at the *clk* rising edge, and add suitable reset behaviour (Note: *rst* is active low).
5. Implement the state transition and output logic. You may use a single combinational always block with a case statement. The case statement switches on the current state, and depending on the inputs will set a value for next state.

   When you have mutually exclusive inputs (as with a coin mechanism), where two inputs are not expected to be active (e.g. pressed) at the same time, it is better to use parallel ifs, as:

   if (a) nst = STA;
   if (b) nst = STB;

   rather than nested ifs, as the priority in the nested version consumes extra logic.

   if (a) nst = STA;
   else if (b) nst = STB;

Task 2 (Implement the Design)

Start a new project in Xilinx ISE, as follows:

1. Start ISE and create a new project targeting the Spartan 6 XC6SLX45-CSG324-3 FPGA.
2. Add the Verilog module *lab1_FSM* (Project | Add Source). Ensure you include the timescale directive at the top of the Verilog file (`timescale 1ns / 1ps)

   OR if you have not completed Task 1, create a new Verilog Module, called *lab1_FSM*, (Project | New Source) with the required inputs and outputs, and then add the required functionality (as in Task 1). This will automatically include the timescale directive.
3. Correct any syntax errors.

Task 3 (FSM Testbench)

Rather than load a design straight onto the FPGA, it is prudent to test it in a simulator first. Here we will build a Verilog testbench for the vending machine.

1. Create a new Verilog Test Fixture called *lab1_FSM_tb* (Project | New Source), and associate *lab1_FSM* with it. Click Finish. You should see a bare testbench with your module instantiated, **reg** and **wire** signals declared, and the signals set to initial values.
2. Add a clock **always** block to toggle every 5 timesteps (after the **initial** block).
   always #5 clk = ~clk;
3. Inside the **initial** block, where it says "// Add stimulus here ",add the following testbench code to initialise the inputs and test different functions of the state machine. This code sets reset to 1 after 10 timesteps. This will de-assert the reset, since the reset on the ATLYS board is active low. It also tests for different functionality, including: vending, cancelling a transaction and money return. Since the clock has a 10 timestep period, you should wait that long (#10) before issuing new sets of inputs.

```
// Add stimulus here
clk = 0; rst = 0; fifty = 0; dollar = 0; cancel = 0;
#10 rst = 1;                // to INIT (0) state
#10 fifty = 1;              // to S50c (1) state
#10 fifty = 0;
#10 fifty = 1;              // to VEND (2) state
#10 fifty = 0;
#20 rst = 0;                // RESET, to INIT (0) state
#10 rst = 1;
#10 dollar = 1;             // to  VEND (2) state
#10 dollar = 0;
#20 rst = 0;                // RESET, to INIT (0) state
#10 rst = 1;
#10 fifty = 1;              // to S50c (1) state
#10 fifty = 0;
#10 dollar = 1;             // to RETURN (3) state
#10 dollar = 0;             // to INIT (0) state
#20 fifty = 1;              // to S50c (1) state
#10 fifty = 0;
#10 cancel = 1;             // to RETURN (3) state
#10 cancel = 0;             // to INIT (0) state
```

   Note: You only need to assert or de-assert signals that change.

   After another 20 timesteps, add a **$finish()** statement to end the simulation.

At the top of your design hierarchy (LHS Top window), select the Simulation view and then select your testbench module. Then in the task window (directly below) expand the ISim Simulator and perform a Behavioral Check Syntax. If there are no errors or warnings, run the simulation. Add the next state (nst) instance to your simulation window (from the object name window for the uut instance), rerun the simulation and check that you obtain the expected values by looking at the wave window. You can also add longer delays to the input sequence if you like, but ensure each set of values is valid at (or before) a rising clock edge. You could also put in an invalid input like multiple high values, just to see what happens.

Task 4 (Implement on the FPGA)

Implement the FSM on the ATLYS board as shown in Fig. 1 (below). You are given the top level module (*top_FSM.v*) which instantiates 2 extra modules (*clockgen.v*, which slows the system clock from 100MHz to about 0.4Hz; and *seven_seg.v*, which displays the current state in 7-segment form). You will also need a .UCF file to map the top level module inputs and outputs to the ATLYS FPGA board. The file *top_FSM.ucf* is provided which already has the *clk*, *rst* and *seg* signals mapped. You will need to add *fifty*, *dollar* and *cancel* (mapped to the 3 horizontal push button switches near the RESET switch), *insert_coin*, *dispense* and *money_return* (mapped to the LEDs LD0 to LD2), and *sClk* (mapped to LED LD7). You will need to map both the location (LOC) and the I/O standard (IOSTANDARD).

In file *seven_seg.v*, edit the seat number (seat_Hi and seat_Lo) to reflect the number of your bench. It is currently set at 25. You may be asked to add an offset to your seat number.

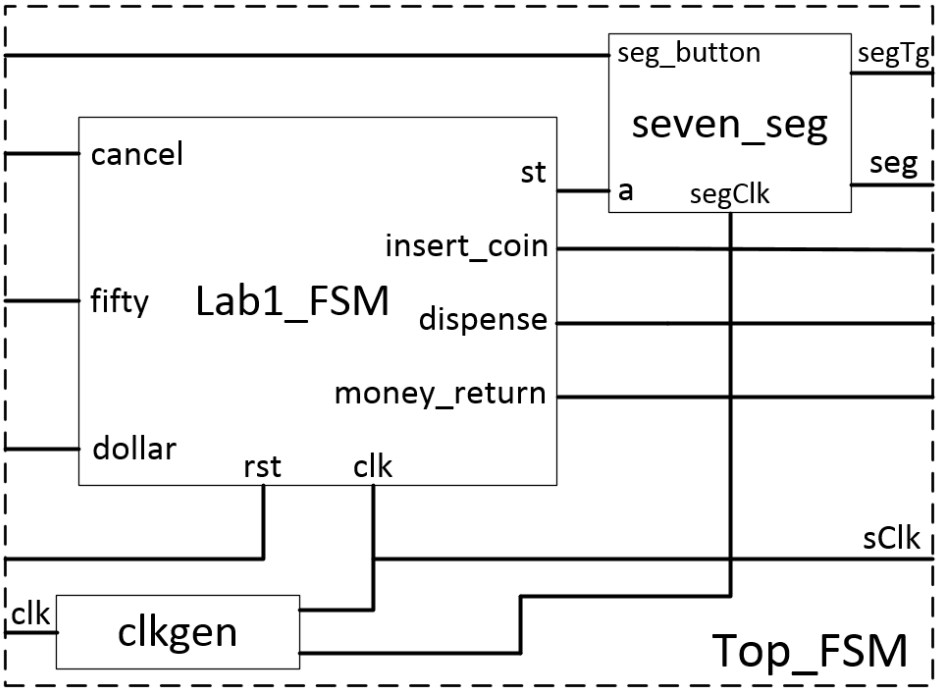

Fig. 1

Synthesise, Place and route and Generate the bitstream (using Xilinx ISE) and program the FPGA (using the Diligent Adept package). Verify that the FPGA implementation works as expected. You should also try some invalid inputs (such as pressing the fifty and dollar buttons at the same time while in state 0 or pressing the fifty and cancel buttons at the same time while in state 1) just to see what happens.

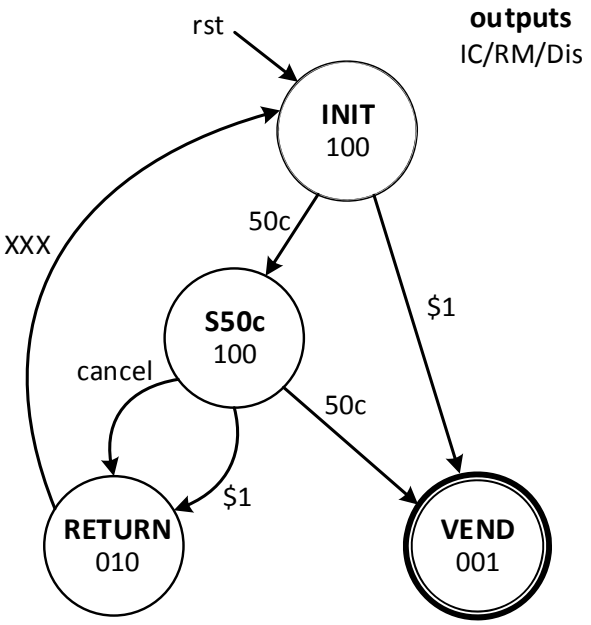

Fig. 2