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 high).
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 Vivado, as follows:

1. Start Vivado and create a new RTL project (Lab1) targeting the Artix 7 xc7a35tcpg236-1 FPGA.
2. Add the Verilog module *lab1_FSM* (Add Sources OR +, then select add or create design sources, then add). 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* with the required inputs and outputs, and then add the required functionality as in Task 1).
3. Correct any syntax errors.
4. Add the constraints file, *Lab1.xdc*, (Add Sources OR +, Add or create constraints)

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* (Add Sources OR ➕). You should see a bare testbench. You will need to instantiate your module, name the instance DUT, and then declare the **reg** and **wire** signals. Add an empty initial block.
2. Add a clock **always** block to toggle every 5 timesteps (after the **initial** block).

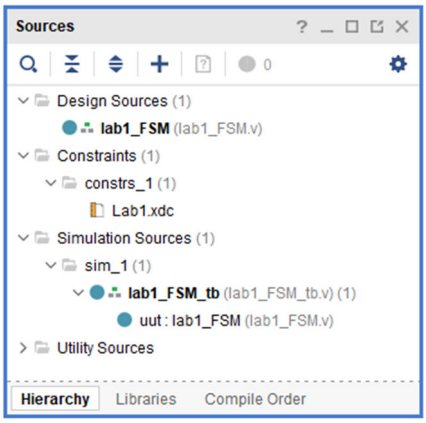
   always #5 clk = ~clk;
3. Inside the **initial** block, add the following testbench code to initialise the inputs and test different functions of the state machine. This code sets reset to 0 (de-asserted) after 10 timesteps. It also tests for different functionality, including: vending, cancelling a transaction 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 = 1; fifty = 0; dollar = 0; cancel = 0;
#10 rst = 0;                // 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 = 1;                // RESET, to INIT (0) state
#10 rst = 0;
#10 dollar = 1;             // to  VEND (2) state
#10 dollar = 0;
#20 rst = 1;                // RESET, to INIT (0) state
#10 rst = 0;
#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.

Check that the sources hierarchy (in the PROJECT MANAGER, Sources window) is similar to the figure below. If it is not, fix or check with the lab tutor.



In the flow navigator window, select “Run simulation”, then Behavioural. Open the waveform window and add the next state (*nst*) instance to your simulation window (in Scope, expand the lab1_FSM_tb instance and then click on DUT. In the objects window drag and drop n*st[1:0]* into the name field of the waveform window). Restart and rerun the simulation and check that you obtain the expected values by looking at the wave window. Try other inputs, including invalid ones and see what happens.

Task 4 (Implement on the FPGA)

Implement the FSM on the Basys3 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). You will also need the *Lab1.xdc* file to map the top-level module inputs and outputs to the Basys3 FPGA board. *Lab1.xdc* already has the *clk*, *rst* and *sel* signals mapped. You will need to add *fifty*, *dollar* and *cancel* to the 3 horizontal push button switches (*rst* and *sel* are mapped to the top and bottom buttons, respectively). You will also need to map *insert_coin*, *dispense* and *money_return* (to the LEDs: LD2 to LD0), Note that *sClk* has already been mapped to LED LD7 for you. You need to do this for both the location (PACKAGE_PIN) and the I/O standard (IOSTANDARD).

In file *seven_seg.v*, edit the bench number, *benchNo_Hi* and *benchNo*, (above the **endmodule** statement) to reflect **your** bench number. It is currently set at 95. The 7-segment outputs have been mapped for you.

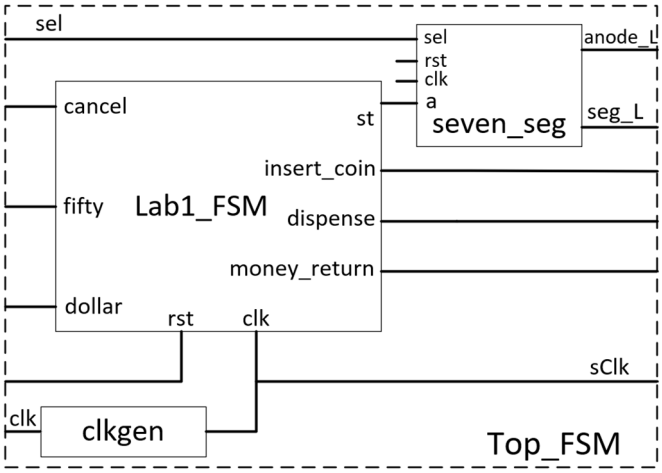


Fig. 1

Synthesise, Place and route and Generate the bitstream (checking and fixing any applicable errors and warnings at each step). Check that the Basys 3 board connected to the computer via a USB cable with the red power LED lit. Open the hardware Manager. Open Target and select Auto Connect. Verify the Digilent device is shown in the Hardware window. Then select Program Device and click Program. Verify that the FPGA implementation works as expected. **NOTE: You may have to press the buttons for some time due to the slow clock. They need a rising edge of sClk.** 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.

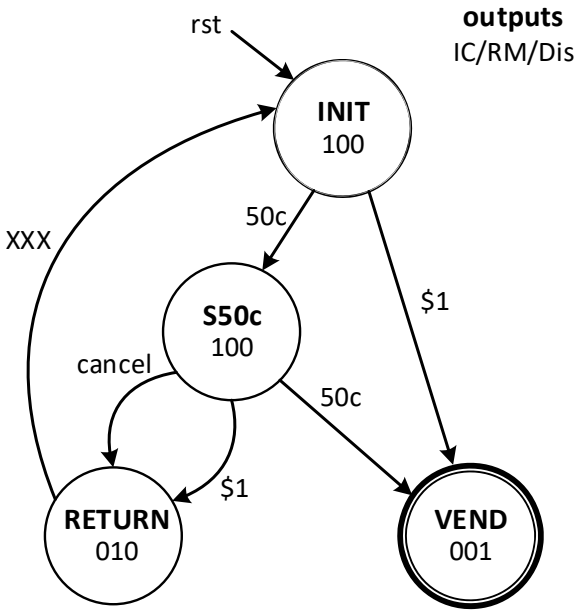


Fig. 2