CE2003 Laboratory 2

Memory Primitives

In this experiment, we will instantiate a LUT-based memory. We will also look at the synthesis tools to better understand what the synthesis tools have done.

We will then simulate the module using the ISE simulator, before implementing the design onto the ATLYS FPGA platform. In the implementation, we will use the slider switches and LEDs to store and show values. We will also use the push button switches for the various control signals. To load a value into an internal register, we first set the value on the slider switches then press the control button. That way we can use the same set of switches for both address and data, differentiated by the control switch.

Task 1

Create a new project and a new Verilog source module. It should have single bit *clk*, *rst*, *write_en*, *save_data* and *show_reg* inputs, along with an 8-bit *d_in* input and an 8-bit *d_out* output.

Declare an internal 8-bit register, *data_reg*.

Next declare an internal 64 x 8-bit RAM. To declare a 64 element array of 8-bits use the following statement (Note that we often number memory contents in conventional order):

```
reg [7:0] ramdata [0:63];
```

Since we will be using the same switches to set the address and data, we need to create an internal 6-input address wire, *addr*. Connect this to the lower 6 bits of the *d_in* signal.

In an **always** block, capture the following behaviour:

1. When (the active low) reset (*rst*) is asserted, the internal register is reset to zero, else when the *save_data* input is asserted, the internal register stores the value of *d_in*.
2. When the *write_en* input is asserted, the register value is stored into the memory location corresponding to the address on the address wire, i.e. ramdata[addr] <= data_reg;

Finally, use a conditional assignment to assign the read output (*ramdata*[*addr*]) to *d_out* when the *show_reg* signal is high and to assign the internal register *data_reg* to *d_out* when it is low.

A block diagram of the expected functionality is shown in figure 1.

Note: At this stage you should not include any clock divider or debounce circuits.

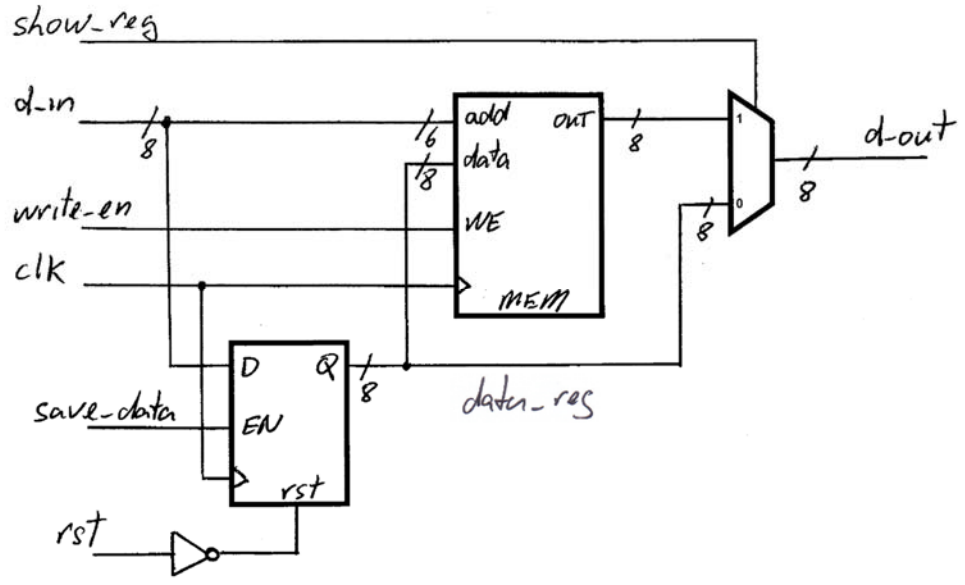

Figure 1. Block diagram of memory circuit

Synthesise the design and check the synthesis report. What elements have been inferred? How many LUTs and registers are used, and what for? Further down the report, under Timing Details, what frequency is expected?

Under the ``Synthesize - XST'' task, double click View RTL Schematic. Starting with the top-level module, double-click to see how it has been built. Can you identify each of the blocks? Compare the implementation to that of Figure 1. If there is a difference, you have probably implemented it incorrectly and you should then discuss the correct implementation with the Laboratory tutor.

Task 2

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 memory device.

1. Create a new Verilog Test Fixture (Project | New Source), and associate the memory module designed in Task 1 with it. 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).
3. Inside the **initial** block, where it says "// Add stimulus here ", set reset to 1 after 10 timesteps. This will de-assert the reset (since the reset on the ATLYS board is active low) before the rising edge of the clock.

Then add some test conditions. These should be added inside the initial block below the line that de-asserts the reset signal. Since the clock has a 10 timestep period, you should wait that long (#10) before issuing a new set of inputs. Use the following set of test conditions:

```
#10 rst = 1;
#10 d_in = 8'h15;
#10 save_data = 1;
#10 save_data = 0; d_in = 8'h01;
#10 write_en = 1;
#10 write_en = 0;
#10 d_in = 8'hA3;
#10 save_data = 1;
#10 save_data = 0; d_in = 8'h02;
#10 write_en = 1;
#10 write_en = 0;
#10 d_in = 8'h87;
#10 save_data = 1;
#10 save_data = 0;
#10 d_in = 8'h01;
#10 show_reg = 1;
#10 d_in = 8'h01; show_reg = 0;
#10;
```

Add a **$finish()** statement at the end of the stimulus, else the Xilinx ISE simulator (ISim) will keep running.

At the top of your design hierarchy, select the Simulation view and then select your testbench module. Then in the task window expand the ISim Simulator and perform a Behavioral Check Syntax. If there are no errors or warnings, run the simulation. Note that the waveform window is behind the source code window. Add the *data_reg*[7:0] and *ramdata*[0:63,7:0] instances from the object name window for the uut instance to the simulation, 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.

Does the circuit generate the expected waveform? *d_out* should be 0, 15, A3, 87, 15, 87. If it doesn't operate as expected, correct the design and re-simulate.

Do <u>NOT</u> close the simulator window. It is needed for assessment later in the lab.

Task 3 (Implement on the FPGA)

Now build a new top-level module with the same inputs and outputs as in task 2, but with two additional outputs, *seg_Tg_out* and [6:0] *seg*, derived from the *clkGen_7seg* module. Add instances of your memory module, the clock generator module (*clkGen_7seg.v*) and two debounce modules (*debounce.v*). Connect the debounce modules to the *save_data* and *write_en* inputs, using the slow clock (*o_clk*). Add pin constraints to connect the module I/O to the correct circuits on the ATLYS FPGA board. You need to connect the eight switches, three buttons, and the eight LEDs. You should use a text UCF file (as done in lab 1). For the three push button switches, connect your top level input which performs the save data function to button P4, the write enable function to F5, and the show register function to F6. If you use the Lab1 UCF template (in the .zip file), the *seg_Tg_out* and *seg* signals are already defined. BUT, remember to remove any unwanted signals.

> Note that you could also use PlanAhead (Tools | PlanAhead | I/O Pin Planning - Pre Synthesis) if you'd like to explore the tools further.

Again, as in Lab 1, edit the seat number (seat_Hi and seat_Lo) to reflect the number of your bench in the *clkGen_7seg.v* file. It is currently set at 25. You may be asked to add an offset to your seat number.

Then, synthesise the circuit and look at the resource usage. Can you spot the extra resources and how they correspond to the extra blocks you added? Complete the design implementation. Under Place and Route, double-click View/Edit Routed Design. If you enable Sites, Switch Boxes, Components, and Routes in the toolbar, and zoom to the shaded area, you will be able to see the final implementation on the FPGA, and explore the circuit arrangement.

Finally, test your design on the ATLYS board. The LEDS should show the current value of *data_reg*. You can use the button you assigned to the show register function (F6) to display the memory contents at the address, shown on the slider switches, on the LEDs. Try storing and retrieving multiple values to the memory.

Inform your lab supervisor once you reach this point.

You need to show the simulator window from Task 2 <u>AND</u> the design operating on the ATLYS board.