CE2003 Laboratory 2
Memory Primitives

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

We will then simulate the module before implementing the design onto the Basys 3 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 RTL project (called Lab2) targeting the Artix-7 xc7a35tcpg236-1 FPGA. **Remember to use a local drive location NOT a network drive.** Add a new Verilog source module (called Lab2_top). 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 two internal 8-bit signals, *reg_d* and *mem_d*, as per Figure 1.

Next, go to the IP Catalog (in the Flow Navigator window under PROJECT MANAGER OR from the Window menu bar). While you are here, quickly open some of the items and see the range of IP that is available.

We want a 64 x 8-bit distributed (LUT-based) RAM. Expand the “Memories & Storage Elements” item, then “RAMs & ROMs” and double-click on the “Distributed Memory Generator”. This will open the generator in a Customize IP window. Change the component name to “Lab2_mem”. In the “memory config” tab, leave the depth as the default (64) but change the Data Width to 8. Select Single port RAM. Click the “Port config” tab and verify that the input and output are both unregistered and that there is no pipelining. Leave the defaults in the “RST & initialization” tab. Click OK, then in the “Generate Output Products” window click “Generate”. Close the IP Catalog window. **Note: ignore any Out-of-Context Module Runs warnings.**

At the bottom of the Sources window, click “IP Sources”. Expand the Lab2_mem item and then the Instantiation Template. Open Lab2_mem.veo. Scroll down to where it says “Begin cut here …”, and copy the instantiation template. Open your top module (Lab2_top.v) and paste the template above the **endmodule** statement. Change the instance name to something a bit simpler (such as U1) and edit the port connections to match Figure 1. Note that the memory address (a) is 6-bits, so use *d_in[5:0]* for the memory address.

Then in an **always** block, reset the internal register (*reg_d*) to zero when (the active high) reset (*rst*) is asserted, else when the *save_data* input is asserted, the internal register stores the value of *d_in*.

Finally, use a conditional assignment to assign the memory output (*mem_d*) to *d_out* when the *show_reg* signal is high and to assign the internal register *reg_d* 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 additional Verilog modules.

Once your design is free of syntax errors, synthesise the design and check the utilization synthesis report (from reports in the bottom window). What elements have been inferred? How many LUTs and registers are used, and what for? Where is the memory and what resources does it use?

Under RTL ANALYSIS, select “Open Elaborated Design”, then select “Schematic”. Verify that the schematic resembles that of Figure 1.

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 (called Lab2_top_tb) for the memory device.

1. Create a new Verilog simulation source (+ or add sources). Open this file and instantiate your top level module (lab2_top.v), declaring the inputs to Lab2_top as **reg** and outputs as **wire.**

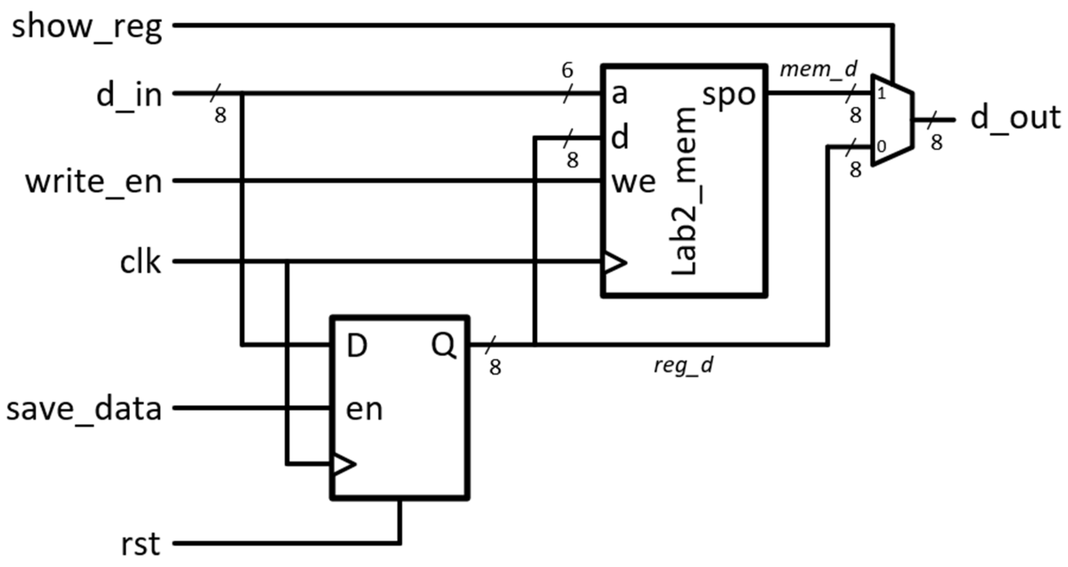

Figure 1. Block diagram of memory circuit

2. Add a clock **always** block to toggle every 5 timesteps (after the **initial** block).
3. Add an **initial** block, setting all inputs to '0', except *rst* which should be set to '1'.
4. Then add some test conditions. Since the clock has a 10 timestep period, you should wait that long (#10) before issuing a new set of inputs. Use the set of test conditions in Figure 2.

```
#10 rst = 0;
#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 $finish();
```

Figure 2. The test conditions

Run the simulation. Note that the waveform window closed because of the **$finish** statement, and you need to reopen it. Add the *reg_d* and *mem_d* instances from the object name window for the uut instance to the simulation and rerun the simulation (using the Restart and Run All buttons just below the menu bar). Check that the *d_out* output is as expected. For this set of stimuli, ***d_out* should be 0, 15, A3, 87, 15, 87**. If it doesn't operate as expected, correct the design and re-simulate. Note: you may need to restart and run the simulation a couple of times to populate both the *reg_d* and *mem_d* values.

Do NOT close the simulator window. It is needed for assessment later in the lab.

Task 3 (Implement on the FPGA)

A new top-level module (Lab2_imp.v) has been provided. It has the same inputs as in task 1, but the *d_out* output has been replaced by an additional *sel* input and three additional outputs, sClk, [3:0] *anode_L* and [6:0] *seg_L*, like in Lab 1. Add an instance of your Lab2_top memory module below the comment in Lab2_imp.v. You should connect the module's *d_out* to the *tmp_data* wire. Note that the clock generator module (*clkgen.v*) and the seven segment driver (*seven_seg.v*) have already been added.

Add the constraints file Lab2.xdc. You will need to add connections to the lower 8 slider switches (SW7 to SW0) to connect to *d_in* and to the 3 horizontal buttons (btnL, btnC and btnR) to connect to *show_reg, write_en* and *save_data*, respectively. The connections to *rst* and *sel* have already been made to btnU and btnD, respectively. The sClk connection to Led[7] (V14) has also already been made.

The push button connections are shown in Figure 3.

Again, as in Lab 1, edit *seven_seg.v* to reflect the number of your bench (benchNo_Hi and benchNo_Lo are found at the bottom of the file, just above the **endmodule** statement). The bench number is currently set at 95. You may be asked to add an offset to your seat number.

Then, synthesise, implement and generate the bitstream for your circuit. Open the Hardware manager and connect to the target. Program the Basys 3 board. Try storing and retrieving multiple values to the memory. Note: you will need to press and hold the buttons until the sClk led flashes on, then release the button.

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.

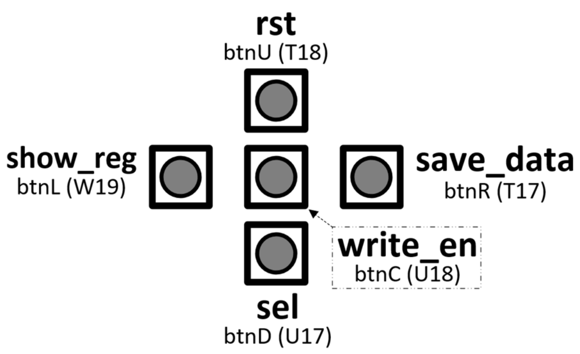

Figure 3.