Lab 7: Memory Bus

UC Davis Physics 116B
Rev 2/26/2020

In this lab, you will synthesize a simple memory bus, which will let you load data into a block of registers using the DIP switches, and then loop over them, displaying their contents.

Required Files

You will need the following files, which can be downloaded from the Canvas site if they are not already on the computer.

- `FPGA Files/Alchitry-pins.txt`: pin definitions
- `FPGA Files/Lab 7 Files/nibble2seg.v`: module to convert a 4-bit nibble to a 7 segment display.
- `FPGA Files/Lab 7 Files/memory-testbench.v`: Test bench file for your memory module.
- Your `slowcounter.v` and `binary2display.v` files from the previous two labs.

Overview

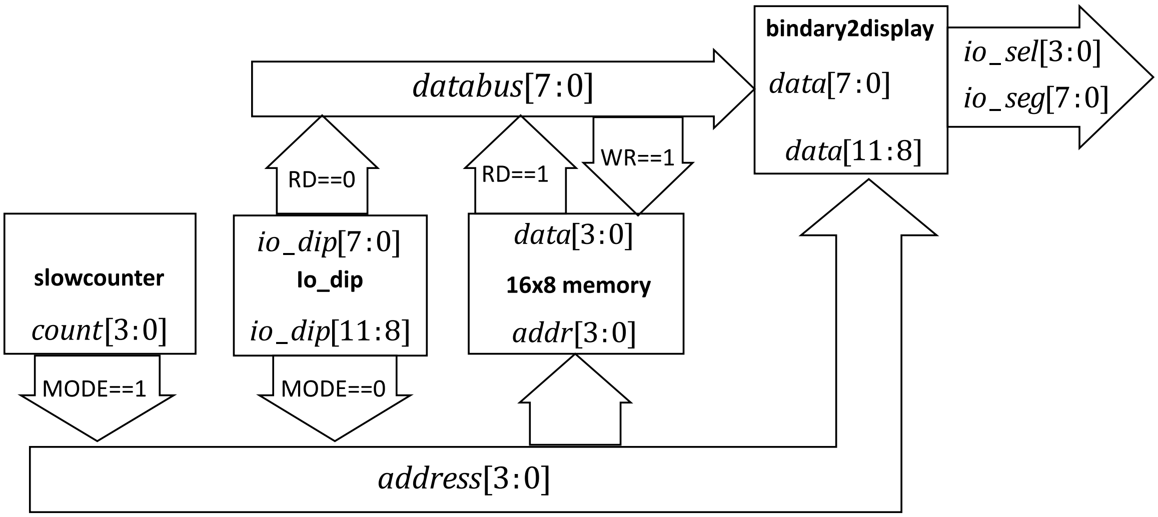


Figure 1: Strucure of the bus for this lab.

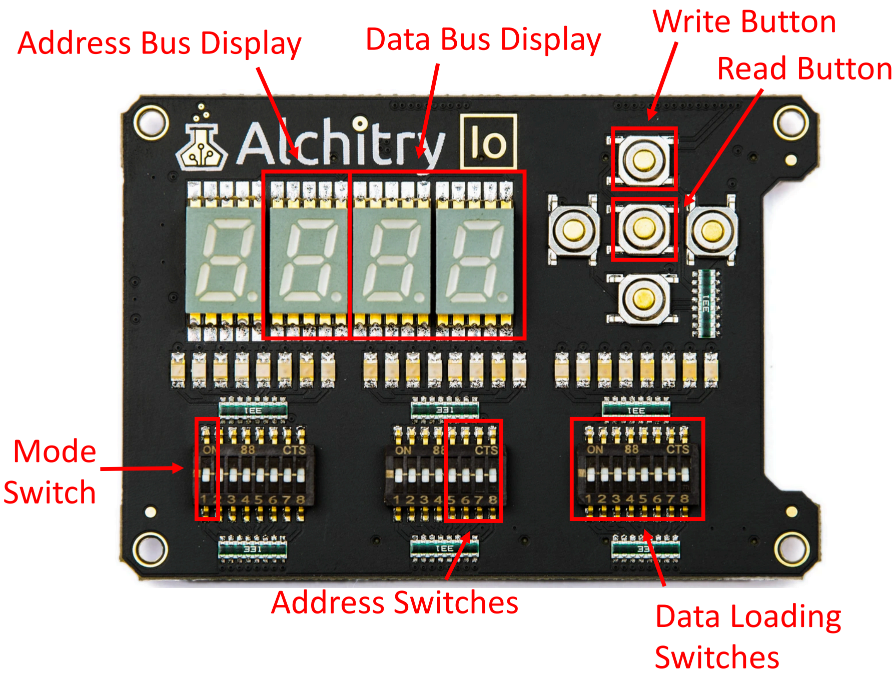


Figure 2: Usage of IO board resources for this lab.

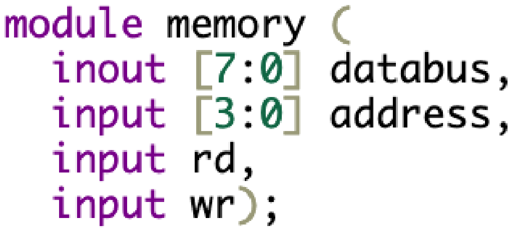
Figure 1 shows an overview of the system we're building. An 8-bit data bus will be used to write data to a 16 location memory and then to display the results in hex on the digital display. The display outputs and control inputs on the Alchitry-Io board are shown in Figure 2.

The design will operate in two modes, which are selected with DIP switch `io_dip[23]`:

- **Mode 0 (write):** In this mode, DIP switches `io_dip[11:8]` will select the 4-bit memory address. If no pushbuttons are pressed, then DIP switches `io_dip[7:0]` will drive the data bus. If the "Write Button" (`io_button[0]`) is pressed, the eight bits of data on the data bus will be loaded into the selected memory address. When the "Read Button" (`io_button[1]`) is pressed, the data at the memory location selected by the address bits will drive the data bus.

- **Mode 1 (read):** In this mode, the 4 address bits will increment at 1 Hz, driven by the lowest 4 bits of "slowcounter", and the data at that location will drive the memory bus; that is, in this mode, the RD bit will always be asserted.

In both modes, the data on the data bus will be displayed in hex on the lowest two digits and the address will be dislayed on the third. The fourth should be tied to zero.



```
module memory (
  inout [7:0] databus,
  input [3:0] address,
  input rd,
  input wr);
```

Figure 3: Header for memory.v module. Note that the databus port is bidirectional (type “inout”).

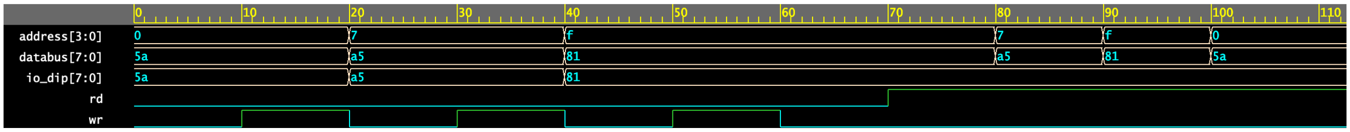


Figure 4: Timeline for simulated memory module.

Lab Activities

Writing and Tessting the Memory Module

Write a module with the header shown in Figure 3. It should have a memory register dimensioned to hold 16 8-bit words, and should behave as follows:

- If `wr==1`, the data bits on `databus[7:0]` will be loaded into the memory location selected by `address[3:0]`
- if `rd==1`, the data bits at the location selected by `address[3:0]` will be asserted to `databus[7:0]`, Otherwise, these bits will be set high-Z by this module.

The read function is most simply realized with a conditional assign statment. It would be reasonable to implement the write function with

```
always @(posedge wr)
   // memory setting command
```

however, the Vivado compiler will recognize this as clocked, synchronous logic and generate a fatal error because the pushbutton isn’t tied to a dedicated clock line input. There are workarounds to reduce this into a warning, but it’s better to just write the logic like this

```
always @(wr)
   if(wr==1}
     // memory setting command
```

Simulate this module in EDA Playground using the `memory-testbench.v` test bench and verify that you get the time line shown in Figure 4. Note that the test bench code will probably give you some good hints for the next part.

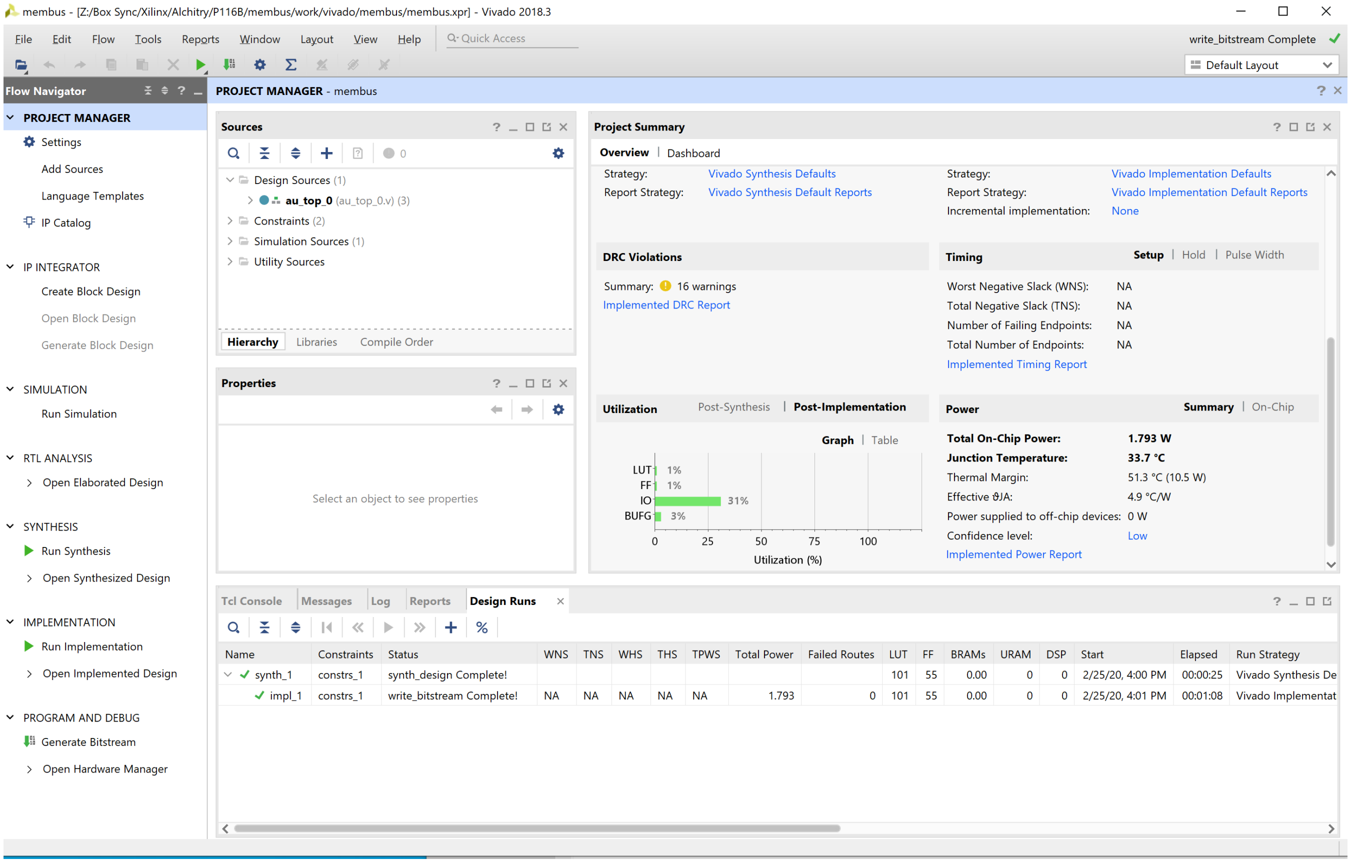
Implementing the Design

Import the `nibble2seg`, `binary2display`, and `slowcounter` modules from the previous labs. Write an `au_top` module to implement the design described in the overview. Note that you'll need to include the inputs from the `io_button[1:0]` and add their locations from "Alchitry-pins.txt" to the constraint file.

Compile and load the configuration. Verify that in Mode 0, you can write random data to at least 8 different memory locations by settin the address bits and pressing the Write button, and then read it back by selecting the address and pressing the Read button.

Then use `io_dip[23]` to switch to Mode 1, and verify that the configuration cycles through the memory location and displays the values you have loaded (hint: tie the RD bit to the OR of `io_dip[23]` and `io_button[1]`).

Looking Under the Hood



Strategy: Vivado Synthesis Defaults
Report Strategy: Vivado Synthesis Default Reports

Strategy: Vivado Implementation Defaults
Report Strategy: Vivado Implementation Default Reports
Incremental implementation: None

Summary: 16 warnings

Worst Negative Slack (WNS): NA
Total Negative Slack (TNS): NA
Number of Failing Endpoints: NA
Total Number of Endpoints: NA

Total On-Chip Power: 1.793 W
Junction Temperature: 33.7 °C
Thermal Margin: 51.3 °C (10.5 W)
Effective ϑJA: 4.9 °C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
✓ synth_1	constrs_1	synth_design Complete!								101	55	0.00	0	0	2/25/20, 4:00 PM	00:00:25	Vivado Synthesis De
✓ impl_1	constrs_1	write_bitstream Complete!	NA	NA	NA	NA	NA	1.793	0	101	55	0.00	0	0	2/25/20, 4:01 PM	00:01:08	Vivado Implementat

Figure 5: Vivado IDE desktop.

Using the Alchitry Labs program simplifies our lab exercises by shielding you from the complexity of the Vivado design tools, but it's worth at least taking a look at some of the more advanced

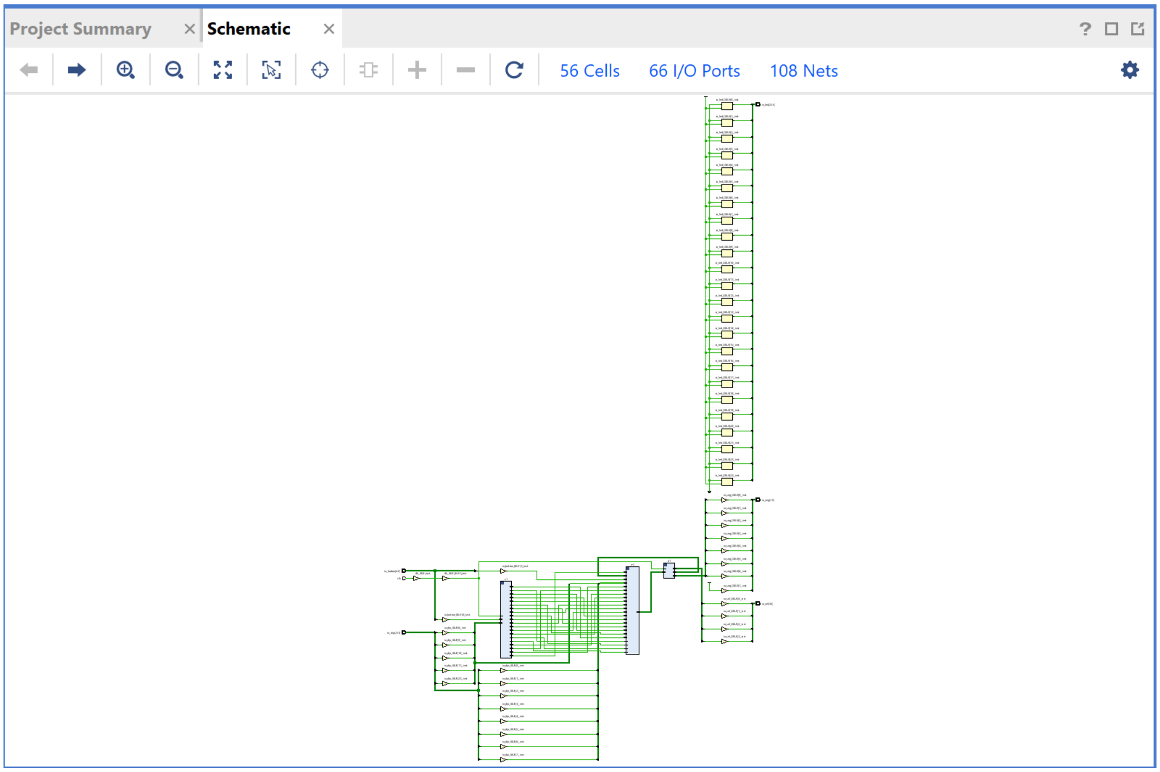


Figure 6: Design schematic from Vivado.

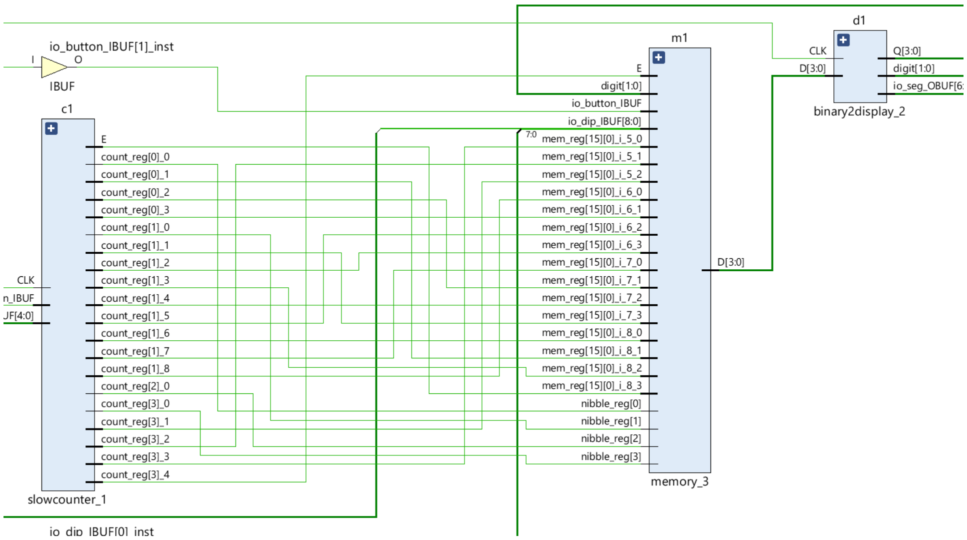


Figure 7: Zoom in on design schematic from Vivado.

features of the whole program. As explained in the first FPGA lab, the way Alchitry Labs works is by building a Vivado project and then running it in the background. You can access this project by going to your project directory, usually in `Documents\Alchitry\(project name)`, then going to `work\vivado\(project name)` and clicking on the Vivado project file `(project name).xpr`.

This will bring up the Vivado desktop, shown in Figure 5. Expand the “Synthesis” menu at the left, and click on “Schematic” . This will bring up a schematic of your design, which should be similar to Figure 6. If you use the zoom button to zoom in, you should be able to zoom in to the three instances iin your design, as shown in Figure 7. Double clock in the instances and verify that you understand how they were implemented in your design.

Lab Report

For your lab report, include your final code, as well as the time line from the simulation. You should also include a screen shot of the schematic of at least one of your instances that you accessed using the Vivado desktop.