

Adding Instruction Memory and Program Counter to Your Computer

1 Objective and Overview

In this lab you will add instruction memory to your lab computer. You will also add a Program Counter (PC) to address the instruction memory. We will only investigate straight line code for this experiment. Branching will be added in lab 8.

You must complete Lab 6 before you start this experiment.

2 Instruction Memory

In this lab, you will add instruction memory to the computer you completed in Lab 6. You will need to add instruction memory as described below and a Program Counter (PC) to address the Instruction Memory (IM). Your instruction memory should be 16 bits wide by 256 entries. An instruction for the lab computer is 16 bits wide, and 16 bits is defined as a word in the IM. The IM is *word addressable*. Your top level design will connect the `BTN1` button on the PYNQ board to all clock inputs, including the PC, memory and the register file. `BTN0` is the reset button for the register file and PC. It also resets the *output* from the instruction memory, but does not touch the memory. DM does not have a reset. Figure 1 shows the top level design for this lab experiment.

2.1 Prelab

Download the prelab assignment from the course webpage. Follow your TA's instructions for submitting the prelab.

2.2 Entering Your Design

2.2.1 Designing the top level

Get the top level Verilog file, `pdatapath_top_lab7.v` from the course webpage. This Verilog code shows how all of your components are connected. You should have regfile, alu and instruction decode components already designed. Get the debounce circuit, `debounce.v` from the course webpage. You will need to generate instruction memory, data memory and VIO to add to this experiment. You also will need to add the Verilog code that describes the program counter as described below.

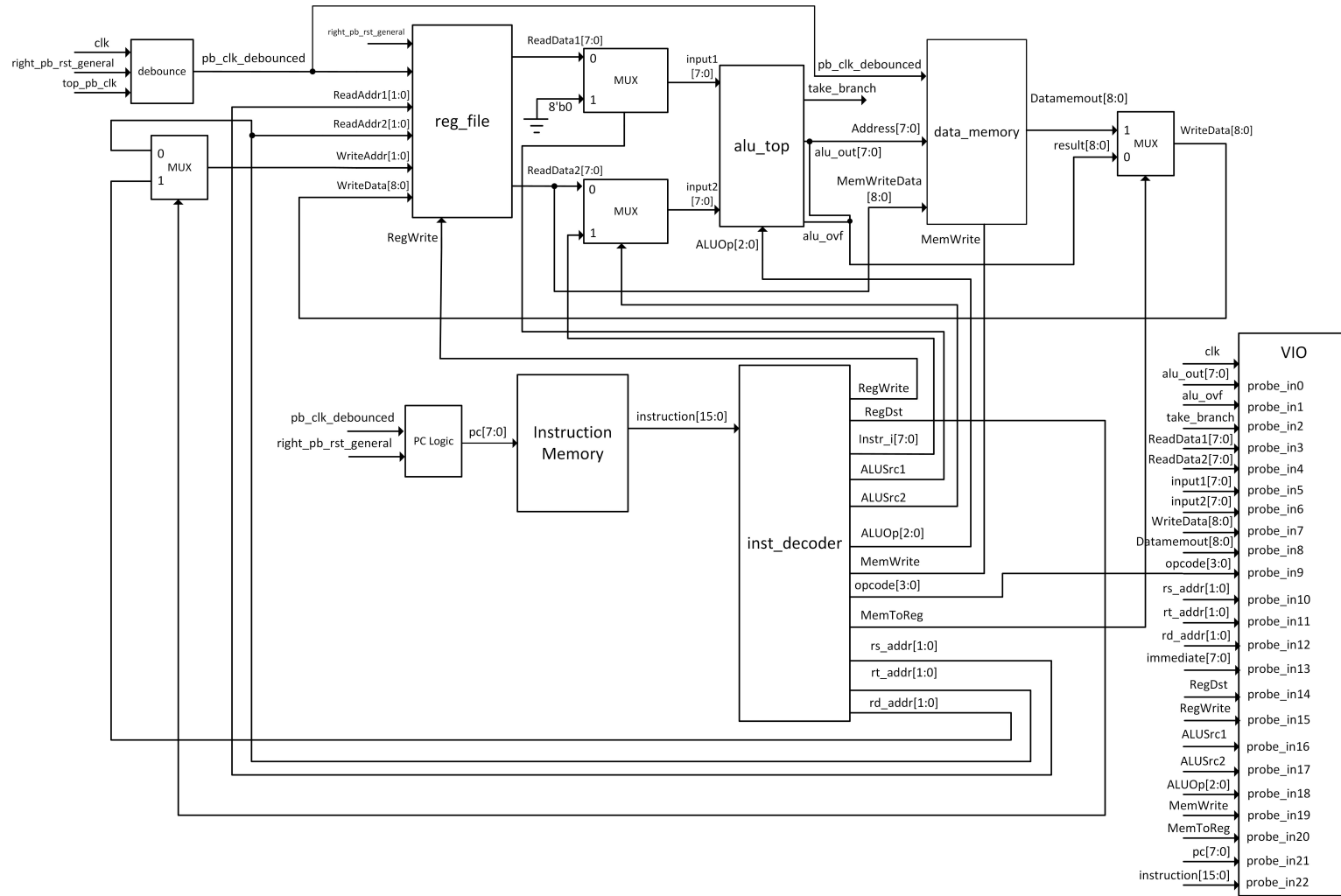


Figure 1: Computer with Instruction Memory and Program Counter

2.2.2 Generating Memory Contents using the Assembler

Get the handout from Canvas that explains how to use the assembler for the lab computer. The format of the output of the assembler matches the format of the *.coe file you need to initialize memory. All you need to do is to rename the generated file for your design.

2.2.3 Instruction Memory

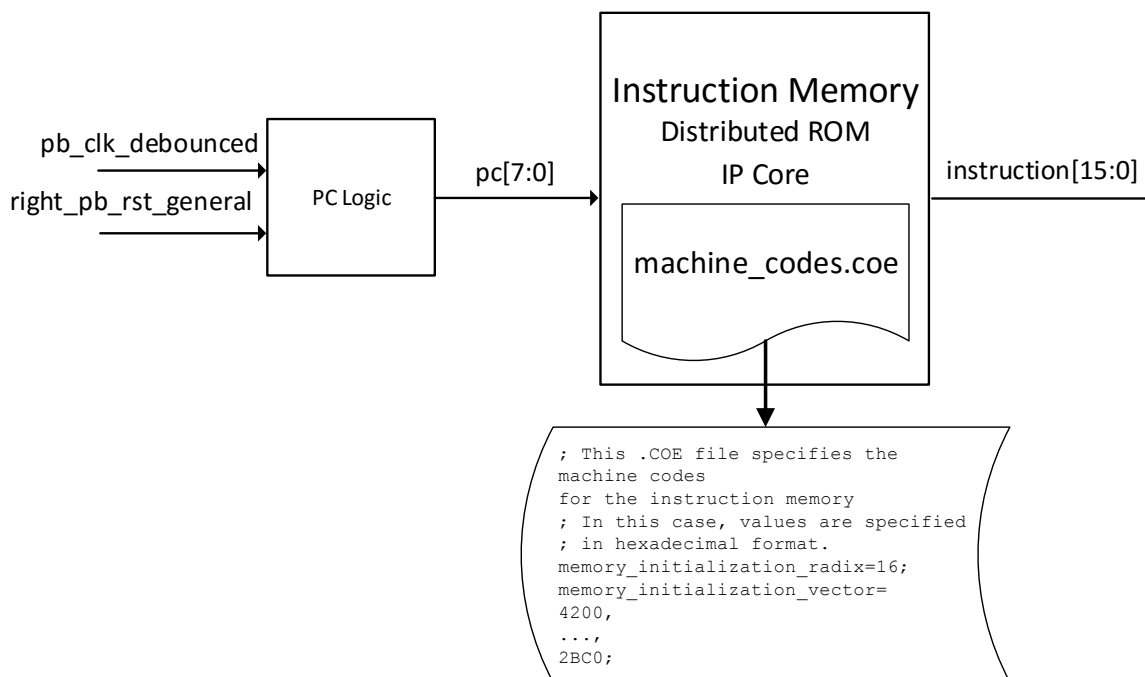


Figure 2: Instruction Memory

For implementing Instruction Memory, add a Distributed ROM IP Core to the design and configure it. You can find this IP core in the IP catalog. To add Instruction Memory to your design in Vivado, follow these steps.

Open the IP catalog by clicking on “IP Catalog” in the Flow Navigator Pane under the Project Manager section. You should see the IP catalog window open. There you can find Distributed Memory Generator in the Memories & Storage Elements category under the RAMs & ROMs section, or you can look it up by typing Distributed Memory Generator in the search bar. After that, double click on the core name (Distributed Memory Generator) to open the configuration window. In the memory config tab of the configuration window, change the Depth to 256 and the Data Width to 16 to match the IM requirements. Set the memory Type to “ROM”. Change the name of the component to `instr_mem` by writing the name in the component name field. Under the Port config tab, check that both inputs and outputs are **Non Registered**. Under the RST & Initialization tab, in the Load COE File section browse for the `mycode.coe` file which you have generated using the assembler. By pressing “Edit” you can validate that you have imported the correct file. Leave all other options as they are, then press OK and then Generate to add the core to your design. Find the instantiation template in the IP Sources tab under: `instr_mem->Instantiation Template->instr_mem.v eo`.

Check that this template matches the interface for the memory in the `datapath_top.v` module. The only interface signals should be the address port `a` and the output `spo` which will be the instruction from that address.

2.2.4 Program Counter

Add the Verilog code for the program counter to your top level design. The PC is an 8 bit counter that is reset with the reset input and incremented with the push button input.

2.2.5 Completing the top level

To complete the top level design, will need to generate VIO to add to this experiment. Note that this VIO module has several new inputs and no outputs. For this lab experiment, you are using VIO to see what is happening in your hardware. All the control logic is being generated by you in hardware.

Add the constraint file for your design. The constraint file for this design is named: `pdatapath_lab7.xdc`

2.3 Testing in Hardware

2.3.1 Implementing the Design with Vivado Synthesizer

Click on Generate Bitstream under the Program and Debug section in the Flow Navigator pane to run the synthesis process. Vivado IDE will run the Synthesis and Implementation processes automatically.

If the synthesis process runs correctly, continue by programming the FPGA board. Follow the same steps you have for previous labs.

2.3.2 Programming the FPGA with Hardware Manager

Program your hardware the way you have in previous labs. Open the hardware manager, open the target device, and program the PYNQ.

Now you should see the `hw_vio_1` window open. Click on the + sign and add all of the probes by selecting and adding them. Now you are ready to test your design in hardware.

2.3.3 Testing Your Design

This experiment uses the same buttons as in experiment 6. There is one reset, named `rst_general` in Verilog, to reset the PC, register file, and instruction memory output port. On the board, it is connected to the right push button (BTN0). The upper push button (BTN1) allows you to single step your design. You need to press BTN1 whenever you want to store data or update the PC.

Use VIO to test your design. Use the instructions you developed in the prelab. Keep screen shots to show that you have done this.

Follow the instructions from your TA for submitting and demonstrating that your lab works.