CSCE 230 Project Part IV

Adding I/O and ARM-like Conditional Execution — Due December 4th

In general, you are allowed to use block diagrams, VHDL, or a mix of both to complete any of these assignments. Each will come with its own advantages/disadvantages. Block diagrams are easier to realize but allow more simple mistakes, e.g. wire connections. VHDL may be harder to implement but you will make less simple mistakes. If you copy any code from other sources, internet/books, you must cite your source or receive a zero for the project. Each team member must work together and should only turn in one assignment/report/check off by the TA.

Overview

Your objective for this fourth part of the project is to add to the already completed processor, extensions to the lw and sw instructions to handle input/output with the help of additional components and information provided to you. You will also extend the processor deign to handle an ARM-like conditional execution of instructions, which is described in pages 11-12 of the Project Overview document, as well as in section D.9 of the textbook. The design process will follow the steps below:

1. Block-level datapath design (or VHDL design)
2. Design of individual Components
3. Datapath integration
4. Control Unit Design
5. System integration and validation

Steps 1-3

Now that you already have a basic processor with load and store capability, only a little bit of logic is needed to connect the I/O memory. You will be given IO_MemoryInterface.bdf as one of the I/O files to extend lw and sw to deal with I/O. **In addition to the mem_write control signal, the upper four bits of the memory addresses are used to determine which I/O device is being addressed.**. The remaining 12 bits of the address will be zero. The upper four bits are assigned to the I/O devices as follows:

1000 – Slider Switches (SW)

0100 – Push Buttons (KEY)

0010 – 7-segment display (HEX)

0001 – Green LEDS

Example: If I want to store to the Green LEDS, I would use the following code:

```
ldw   r2, 20(r0)    \\Address 20 in memory holds the value 0x001000, which is the address for LEDS
addi  r3, r0, 15    \\Load r3 with a value to store on the leds
stw   r3, 0(r2)     \\Store r3 on the leds, the first four leds should light up
```

Since the mem_write control signal will be 1 for sw instructions regardless of whether you want to write to memory or perform an output operation, additional logic is needed for the mem_write input to the data memory. In essence, if any of the upper four bits of the address are 1 and mem_write is 1, then the input to your IO memory interface should be 1. If all of the upper four bits are 0 and mem_write is 1, then the mem_write input to your Main memory interface should be 1. In all other cases it should be 0 for both of them.

Similarly, y_select will be 01 for lw instructions, so the upper four bits of the address should be used to select which memory output (data memory or I/O memory) will be an input to mux_Y. A multiplexor would allow you to choose between these two data values, with the select signal being determined by the upper four bits of the address.

In addition to clock and reset, you need to add inputs and outputs for each of the four I/O devices and assign them to pins on the board. To do this, right-click on a pin and select Locate→Locate in Pin Planner. Refer to the tables in Pin_Connections_FPGA_Board.pdf that is included in this parts files. Note, that if you map your reset input to a push button, the push button has a value of 1 when not pressed and 0 when pressed. Also, the Hex display operates in a similar fashion. A 1 turns a bit off, while a zero will turn it on. Make sure to connect your clock to the 50mHz option.

For conditional execution you should already have a way of storing the conditional flags with your PS register. The only thing that should change is when you want to update the PS register. You can use the flag_enable to determine when to update

your PS. This flag should be set based on the input of the S bit. Do this in your control unit in stage 3 when the value of the flags is updated. **You only want to update the flags/PS register during stage 3 of the conditional instruction**

Step 4

The control unit already has inputs for cond and the four flags. The easiest way to implement conditional execution is to add another internal signal (like the wmfc signal) to represent whether or not the instruction should be executed. You should check the condition and the flags during the down cycle of the clock, not the upcycle. Add an if statement before the if(rising_edge(clock)) to check for falling _edge(clock). In that if statement, you want to check the flags to match the condition passed in, and change the enable signal you created. Using if and else statements, you should check through each Cond and see if the correct flags are set. If the flags are set correctly, then the instruction should execute normally. If not, then an disable signal should be turned on. You should check the value of this disable signal in stages 3-5 before you do any decoding in them.**Note that not all instructions can execute conditionally. Also note, that stage 1 and 2 should always run**

You may have been wondering about the S bit. I already mentioned it above, but this is what you will use to determine if you want to update your PS. You will have to add code in your control unit that sets the flag_enable bit if the S bit is set. This should be done for R and D type instructions, except JR. Note that this control flag should be treated like a mem_write or rf_write signal. It should only be on for 1 stage, and then turned off during the next stage.

Step 5

A **MAJOR** component of each part in the project is the last step. **Make sure you reserve sufficient time for this step** as your validation results will help us understand the success of your implementation.

You will also test your implementation thoroughly for correctness and timing performance. Also create a program that uses the I/O and download your processor to your board and see that it works as expected. To program the board, make sure all your pins are assigned, then go to Tools→Programmer, then make sure that your board is listed under hardware and click Start.

Assignment

As described in the above sections, modify your datapath to include the I/O memory interface and the additional logic for it to correctly function, and add to the control unit to execute instructions conditionally.

Tasks you must perform

- Download the provided component(s) from Piazza
- Look over and develop an understanding any provided component(s)
- Add the I/O memory interface to your datapath.
- Add to the control unit to implement conditional execution.
- Hand-assemble instruction to test I/O and instructions that have conditions to execute.
- Create a test script(s) (.do file(s)) and run a simulation(s) of the processor to make sure the I/O works and that instructions can be executed conditionally.

Testing and simulation

Before verifying your enhanced processor with the two extensions made in this part by a test program, consider doing basic component-level testing of the modified parts of the design. Then do a functional simulation of the test programs for comprehensively testing all the instructions implemented thus far. As before, add all inputs and outputs and look for errors as your test program runs.

In your timing simulation, strive to test your processor for correct operation at the highest possible clock speed. As you processor grows more complex, it is very likely that you will have to slow down this clock speed of the system to produce the correct results by giving signals enough time to propagate. The test cases for the timing simulation should be the same as in the functional simulation.

Grading and TA check-off

The grading breaks down from the table below:

Points	Part
15	Design File (.qar)
10	Simulation
20	Check off and demonstration
30	Technical report

The design file, simulation, and report are due by Midnight of December 4th. The Check off is due by the time the last TA leaves on December 4th. If you are unable to finish this part in time, **still hand in what you have done so far for partial points**. Late submissions will get zero points so make sure to meet the deadlines. Each part of this project is incremental so being late on a deadline will make you late on the next one.

Check off

The check off for this section will include the demonstration of conditional I/O. An easy way to get the check off is to set up the following program in your .mif file and demonstrate it to a lab TA.:

```
        ldw   r2, 20(r0)      //Load the pushButtons
        ldw   r3, 21(r0)      //Load the green leds
        addi  r5, r0, 4     //Will be used in bit masking
        addi  r6, r0, 12      //Something to display if pushed
LOOP:     ldw   r4, 0(r2)     //Load the status of the pushbuttons
        stw   r5, 0(r3)     //Something to display if not pushed
        cmp   r5, r4        //Want to know if button 3 is pushed
        beq   LEDS
        br    LOOP
LEDS:     stw   r6, 0(r3)
        ldw   r4, 0(r2)     //Load the status of the pushbuttons
        cmp   r5, r4        //Want to know if button 3 is pushed
        beq   LEDS
        br    LOOP
```

This needs to be demonstrated on the board. You will not receive credit for the check off if your processor can not show conditional execution on the board.

Submit

You must submit the files electronically to webhandin by 11:59PM on December 4th. The naming convention should be as follows:

- "Project-team-number"_Part4.qar for the project archive file. The simulation file should be located inside of the archive.
- "Project-team-number"_Part4.pdf for the project report. This report should follow the best practices of writing technical reports.

Hints

A few hints to help you along.

1. Reset is active high, so if you connect key0 to your reset input, make sure to invert it so that your processor actually runs.
2. You can assign pins to a bus input. You do not need an individual node input for each pin.
3. You can test your I/O in simulation by forcing the value of the keys or the push buttons in your .do file
4. Test each I/O device individually
5. Get one condition to work before adding the rest
6. In the .mif file, leave the first address space empty like before.
7. Get I/O to work in simulation before trying it on the board. When testing in modelsim, you will have to force the Switches or Keys input in your .do file, otherwise the input would be unknown.
8. When assigning pins to inputs, be sure to assign the clock to the 50mHz clock on the board, and the reset to key0. When reset is assigned to key0, it will reset your system if the button is not pressed. Make sure to invert the signal that gets taken in from the board.