27.4.2016

EE540 Spring 2016, Assignment4, Due 11.5.2016

Microprocessor Design: Datapath, Control Unit & Memory

The purpose of this assignment is to assemble and simulate the pipelined datapath of your microprocessor and to design the controller of your microprocessor. At the end, you should run a simulation of a test program containing couple instructions to show that your microprocessor works.

Datapath

The datapath of your microprocessor stores, accesses, and computes the operands of your instructions, and interfaces with the external modules. A good starting point for this exercise would to be to look at the data flow for each of your instructions and make sure that you have all the components necessary to do the desired computations. By now, you have designed most of the components needed to achieve the desired functionality. You may need to design muxes. You must decide whether to locate modules on or off of the datapath. For example, you may decide to put the instruction register (IR) and processor status register (PSR) on the datapath. If so, you should design those and incorporate them into your datapath. You must also commit to multiplexor-based design. A natural next step is to assemble all of the modules together and simulate for functional and timing correctness. This requires that you present correct control signals to all of the control points in the simulation. At this point, you can either force these input signals yourself or implement the control unit and then simulate both the datapath and the control unit together. However, in the latter case, debugging would be harder. You are required to make a list of all the control signals and explain their purposes. With the control signals identified, it will be easier to simulate the functional correctness of the datapath.

Control Unit

The first step in designing the controller is to identify and define the control lines, which you did while building your datapath. The controller for the baseline architecture is simplified by the fact that all instructions are one word long. In the simplest implementation of the microcontroller, decoding can be done with no state machines. If state machines are used for jump and branch, or other instructions (e.g. interrupt processing) that require two cycles in the execute stage, the state machines can be trivially simple. Implementing the control in a pipelined machine that has a separate decoding stage means having the necessary logic to set the control bits to their respective values in each stage for each instruction. This is accomplished by decoding the OPCODE from the instruction register and the condition bits (in case of a branch or jump) to create the control bits which are stored in the Control Register. These control bits dictate the function of the *EXECUTE* stage in the next cycle. Simple control is also required for the *FETCH* stage, the details of which depend upon the timing of memory access and write back. In our baseline architecture, *DECODE* and *EXECUTE* are in the same stage (to avoid data dependencies). The *DECODE* is so simple and fast that it does not add much delay to the *EXECUTE* stage. You can implement the control (decode) logic without a control register, but be certain that any lines that control writing to data memory or register files are hazard-free.

Your data memory interface (and sometimes your program memory interface) will have associated control signals in addition to data and address buses. These control signals should also be driven by your control unit. You may also need a few flip-flops to save certain control bits for one more clock cycle; for example, if you write back results into the register file on the positive edge of the clock after the completion of the execute stage, you may need to latch the Write_Enable signal on the falling edge of the clock so that you do not use the control signal value of the next instruction.

Memory

Before you start on your datapath and control unit, it is important that you know how you are going to implement the instruction and data memory. For this purpose we are going to use the block RAMs inside the Xilinx FPGA chip. The Spartan6 FPGAs of the Nexys3 boards in our labs incorporate 576 Kbits of block RAM memory. For the memory part of our microprocessor, we are going to use these block RAMs of the Spartan6 FPGA. The Block RAM primitives available on the Spartan6 FPGA can be used for building efficient on-chip memory structures. All reads and writes to block RAMs take a single clock cycle. While the memories can be made dual-ported, we will only use single-port memories.

To instantiate block RAM memories, include the spblockram.v module shown below as a new source in your project:

```
module spblockram (clk, we, a, di, do);
 input clk;
 input we;
 input [4:0] a;
 input [15:0] di;
 output [15:0] do;
 reg [15:0] RAM[0:31];
 reg [4:0] read_addr;

 always @(posedge clk) begin
  if (we == 1'b1)
        RAM[a] <= di;
  read_addr <= a;
 end

 assign do = RAM[read_addr];
endmodule
```

When synthesized, the module shown above becomes a memory with 32 words, each of width 16-bits. The code can be modified to build memories of different sizes by changing the size of the address vector a (i.e. size [4:0] in the code above).

Later, you will be given a test code (test.bin) that includes instructions for your microprocessor. The test code will have the binary data that you should put into your instruction memory. Once you receive this binary data, you can put them into appropriate addresses of your memory code to simulate your design.

Assignment:

Datapath

Simulate the entire datapath in Isim for functional verification. You can either force control variables and register file select signals by yourself or simulate both the datapath and the control unit together. You will have many control signals. Run at least one simulation each for all instructions (both Reg-Reg and Reg-Immediate). Pay careful attention to the write-back portion of your "execute" pipeline stage. List and describe all of the control signals for the datapath. Your description should include functional specifics for your individual components. You may include a truth-table if you like.

Control Unit

Write Verilog codes for your controller. Simulate this Verilog description of your controller along with your datapath and memories prior to synthesis to ensure correct operation (*Hint: partitioning your controller into smaller modules can be helpful in monitoring the quality of the synthesized result*). After testing the Verilog model, synthesize it in Xilinx ISE. You should set the timing constraints based on your knowledge of the timing of your datapath and any estimates you may have for associated peripheral circuits including memories.

Deliverables

Run an Isim simulation of the entire microprocessor. You are required to show proper operation for the binary test code (test.bin) that will be given to you in assignment5. You should also demonstrate a few examples of each available arithmetic/logic operation, the ability to change program flow (branch, jump and jal) as well as important cases such as system reset for cases that are not included in the given binary test code.

Your reports (*.doc or *.pdf) should include the snapshots for all simulation results, a summary of the synthesis results (do not simply copy/paste the *.syr file, present the results in tabular form if possible), schematics of the synthesized circuits, and your comments. Important considerations that went into the design, and extra features, if any, should also be documented. Your reports should not contain your Verilog codes, which will be submitted as individual files. Before emailing your homework, zip all relevant Verilog files including testbenches, synthesis output files (*.syr only), and your report under a folder named as: *hw4_yourlastname_yourfirstname*. Only the following file types should be in the unzipped folder: *.doc (or *.docx or *.pdf), *.v, *.syr.

Return the project files and execution of the binary instructions for testing (test.bin). You must verify the controller along with the datapath and memories (not standalone). Note that test.bin only tests required instructions. You may have additional instructions, interrupts or special operating modes that are not part of the basic requirements. While you should design your controller to support any of these extras, you do not have to demonstrate their proper operation for this assignment. Return complete schematics including the controller, datapath, memories and any i/o pads on external memory interfaces.