EE 533 Lab #6
Pipelined Datapath on NetFPGA

Instructor: Young Cho - youngcho@isi.edu

Part 1 – Extending Synchronous Adder into ALU

You must extend your synchronous 8-bit full adder into a synchronous Arithmetic Logic Unit (ALU) that is at least 32-bit wide. You must include add, subtract, bitwise AND, bitwise OR, bitwise XNOR, compare, and logical shift functions. Other useful network-related functions may be substring comparison and shift-then-compare. You may use any combination of schematics, Verilog, Core IP, and previous projects. Just do not copy someone else's work. If we find out you copied someone else's work, you will receive zero for the entire laboratory.

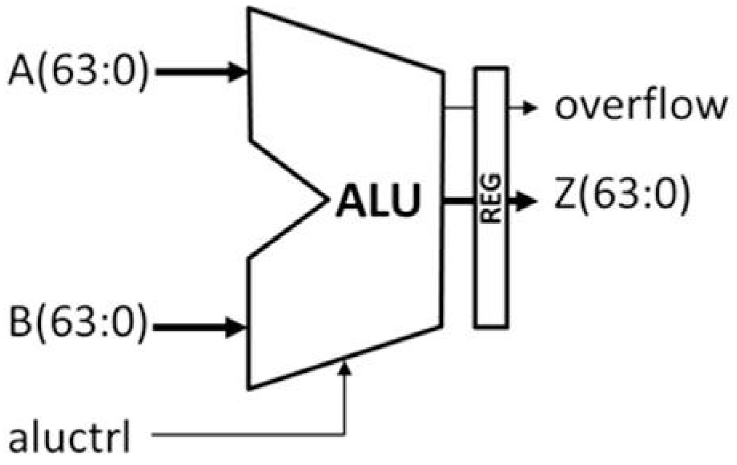

Figure 1: 64-bit Arithmetic Logic Unit

Make a table for the ALU control and what each mode represents (this table should be unique for each team). Simulate your circuit and verify the correctness of the design. Take screenshots of your simulation results with a description and the table, then include them as part of your write-up.

Part 2 – Building Register File and Memories

You are to build a register file for your processor. It must have at least 4 registers that are 32-bit wide (64-bit is recommended since the NetFPGA datapath is 64-bit wide). A typical register file can be constructed with two read ports and one write-back port as shown in the following figure.

Simulate your design and verify the correct functionality of your work. Include screenshots of your simulation results and explanations in the report.

Use Core IP to generate a BRAM-based dual-port synchronous memory of 64-bit wide data I/O with 256 entries. Check to make sure that you are using a single BRAM. This will be the data memory for your processor for now. Also, generate a BRAM-based single-port synchronous memory of 32-bit width and 512 depth or 16-bit by 1024 depth (dependent on your ISA). Check to make sure that you are using a single BRAM. This will be the instruction memory for your processor. Test your memory modules for correct functionality. You need not include any screenshots for these units.

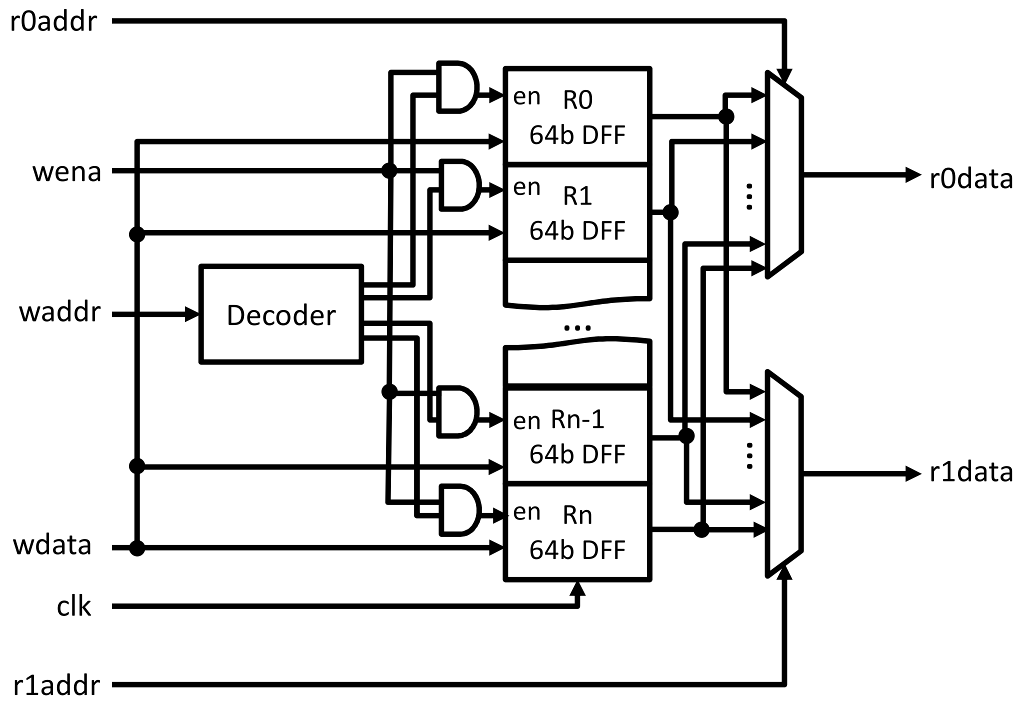

Figure 2: 64-bit Register file with two read ports and 1 write-back port

Part 3 – Building Pipeline Datapath

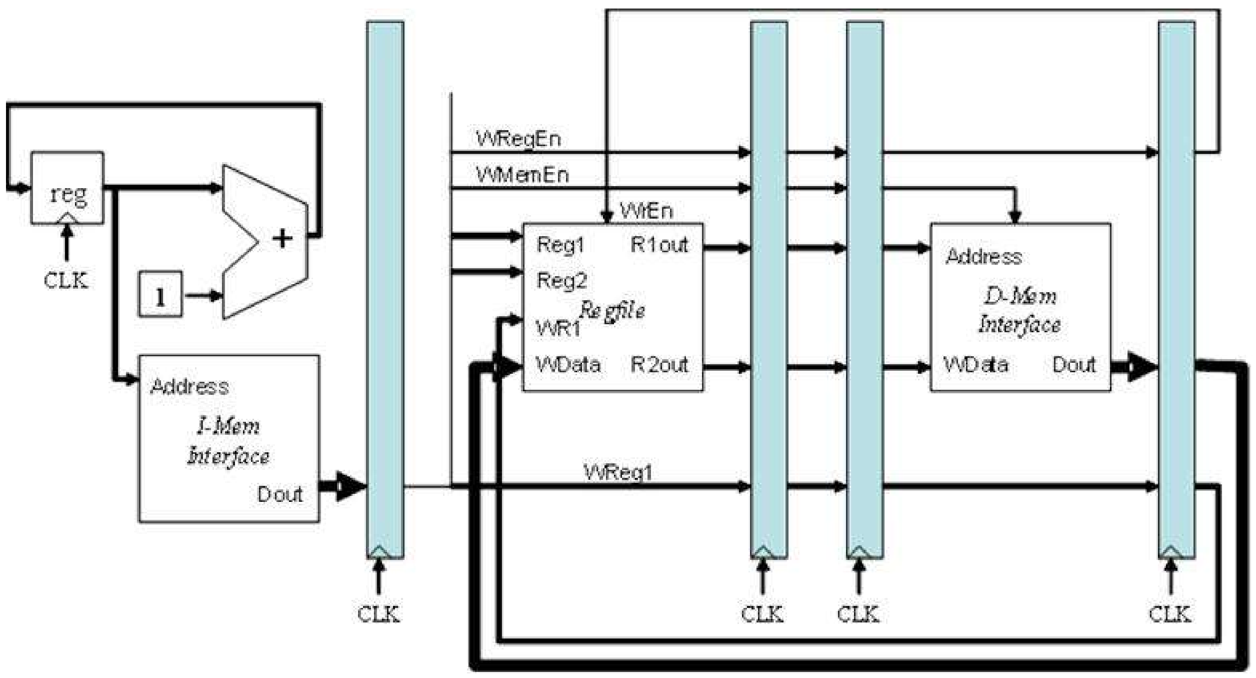

Figure 3: Skeleton Pipelined Datapath

The datapath is a framework on which your pipelined processor will be built. The following figure represents a datapath that you may build to test your memory interface and register files without making a controller. Eventually, you will add the ALU from part 1 and Multiplexers to make a full pipelined processor, but we will not do that for now. For this datapath to function correctly, you need simple and dumb Instructions. These instructions can be loaded onto the memory before the processor executes.

Instruction Format

WMemEn (1bit)	WRegEn (1bit)	Reg1 (3bits)	Reg2 (3bits)	WReg1 (3bits)	Unused (rest)

Create a small program in bit vectors that would first load the data from D-Mem into one of the registers and then load it to another register. Then, store the value of Reg2 to the address value in Reg1. Following can be an example of your dumb-dumb D-Mem content and a test program:

Data Memory

Addr	Value
0	4
1	X
2	X
3	X
4	100

Instruction Memory

Addr	WME	WRE	REG1	REG2	WREG1	Comments
0	0	1	000	XXX	002	//Load Address 0 (Reg0=0) of D-Mem to Reg2
1	0	1	000	XXX	003	//Load Address 0 (Reg0=0) of D-Mem to Reg3
2	0	0	XXX	XXX	XXX	//Nop
3	0	0	XXX	XXX	XXX	//Nop - value 4 written into Reg2 by now
4	0	0	XXX	XXX	XXX	//Nop - value 4 written into Reg3 by now
5	1	0	002	003	XXX	//Store 4 into Mem address 4

Preload the memory modules with the above values and simulate the execution of the pipeline using the tools. Fill the rest of the I-memory with 0s to prevent the datapath from corrupting your results. Take screenshots of your simulation results with a description, then include them as part of your write-up.

Part 4 – Integrating the Pipeline into NetFPGA

You will place your design in the user_datapath, and all of your interactions with your datapath should be through the software/hardware register interface. As with the Mini-IDS lab, you should start with the reference router design and connect the input and output pins of the user_datapath.v isolating the processor from the router design.

You need to design an interface to program your data memory and instruction memory using software/hardware register. You are free to do whatever you want to make this happen. One way to achieve this is using address map. Your I-memory and D-memory can be assigned address spaces (i.e., I-memory could be address 0-511 and D-memory could span address 512-768). Your interface may have registers for command, address, and data. Then whenever you assign some values to address and data registers along with a command for write, your interface module then can write the data to the corresponding memory. You should also build an interface for reading the values from the memory. This interface can be used to verify the correctness of your design.

Include a description of your interface module and the block diagrams in your report. In addition to your report, you are to demonstrate the working of this datapath in your YouTube demo video

<u>Submission and Demonstration</u>

- Draw a high-level design of the datapath. The figure should depict the communication between the components.
- Include the following in your report:
    - Screen Capture of Schematics
    - Generated Verilog Files
    - GitHub records and descriptions per team member
- A major part of the assignment will be in demonstration of your system. Please complete the process of going through your design in your YouTube demo video.