Design HW II

Prof. Mani Srivastava

EEM16/CSM51A - Logic Design of Digital Systems

NOTE: This homework is to be done individually using LogiSim.

SUBMISSION PROCEDURE: You need to submit your solution in electronic form at http://nesl.ee.ucla.edu/courses/eem16/2013w/submissions . You would need to use the user id and password handed out in the first lecture.

NO LATE SUBMISSION. Only exceptions would be for medical emergencies based on a certified note from a doctor.

Following table to be filled by the course staff only:

Problem	Maximum Score	Your Score	Comments
ALU			
Correctness + Quality	15		
REG			
Correctness + Quality	10		
DATAPATH			
Correctness + Quality	15		
TESTBENCH_DP			
Design	5		
Testing	10		
Submission of required material	5		
Total	**60**		

Now that hopefully the previous design homework made you comfortable with Logisim, in this and the next two design homework you would design a very simple 16-bit computer - one that is more like a calculator really. The central processing unit (CPU) in computers have two main portions: the “Datapath” where the computation operations are done and variables stored, and the “Controller” which orchestrates the order in which the operations are done. This is a very simplistic view, but roughly right.

In this design homework we will make a simple “Datapath”. Your Datapath in turn will have two main entities: an “Arithmetic Logic Unit” (ALU) which is a logic block capable of doing a variety of arithmetic and logic operations, and a “Register Bank” which stores variables and from which ALU can gets it operands and into which it can deposit the results.

For sake of preciseness of specification, this homework has a lot of verbiage, but the design itself is not hard.

You’d design the following circuits, all in a single Logisim project named *dhw2*. For us to grade your homework with our test suite it is extremely important that your design adhere strictly to the names and orders of input and output pins for each circuit. Moreover, all input pins of a circuit must be on the left side in the detailed circuit schematic (i.e. not just on the block symbol of the circuit but the actual detailed circuit), from top to bottom in the order listed. Likewise, all output pins of a circuit must be on the right side, from top to bottom in the order listed in the detailed circuit schematic.

1. A circuit named *ALU* as per the following description:

You may use any Logisim modules for this circuit.

Inputs:

DINA[15:0], DINB[15:0], and OPCODE[3:0]

Important note: These input pins must be on the left side of your circuit in the preceding order from top to bottom. I.e. the 16-bit pin for DINA[15:0] should be at the top of the left edge, then the 16-bit pin for DINB[15:0], and finally the 4-bit pin for OPCODE[3:0].

Outputs:

DOUT[15:0], FZER, FNEG, and FOVF

Important note: These output pins must be on the right side of your circuit in the preceding order from top to bottom.

Functionality Description:

The OPCODE[3:0] specified what operation is being done by the ALU as per the following table. For arithmetic operations (NEG, ADD, SUB, MUL) the operands are to be considered as 2’s complement numbers.

FZER = 1 if DOUT[15:0] is all 0; otherwise 0

FNEG = 1 if DOUT[15] is 1, thus indicating a negative number

FOVF = 1 if ~~the most recent~~ ADD, SUB, NEG or MUL resulted in an overflow or underflow in 2's complement form. ~~Note that this implies that your circuit needs to remember a logic value, and so you should consider using a RS latch here.~~ It is 0 otherwise.

OPCODE[3:0]	Name	Operation
0000	INV	DOUT[15:0] = ! DINA[15:0] (i.e. bitwise logical not)
0001	AND	DOUT[15:0] = DINA[15:0] & DINB[15:0] (i.e. bitwise logical and)
0010	OR	DOUT[15:0] = DINA[15:0] | DINB[15:0] (i.e. bitwise logical or)
0011	XOR	DOUT[15:0] = DINA[15:0] xor DINB[15:0] (i.e. bitwise logical xor)
0100	NEG	DOUT[15:0] = - DINA[15:0] (i.e. 2's complement)
0101	ADD	DOUT[15:0] = DINA[15:0] + DINB[15:0] (i.e. 2's complement add)
0110	SUB	DOUT[15:0] = DINA[15:0] - DINB[15:0] (i.e. 2's complement subtract)
0111	MUL	DOUT[15:0] = DINA[15:0] * DINB[15:0]
1000	LSHFTL	DOUT[15:0] = DINA[15:0] << DINB[3:0] i.e. bits of DINA[15:0] shifted left by DINB[3:0] places with 0's filled in from the LSB end
1001	LSHFTR	DOUT[15:0] = DINA[15:0] >> DINB[3:0] i.e. bits of DINA[15:0] shifted right by DINB[3:0] places with 0's filled in from the MSB end
1010	ASHFTR	DOUT[15:0] = bits of DINA[15:0] shifted right by DINB[3:0] places with DINA[15] filled in from the MSB end
1011	UNUSED (i.e. DON'T CARE)	
1100		
1101		
1110		
1111		

2. A circuit named *REG* as per the following description:

This circuit is in effect a small memory into which one can store values. It has 14 locations addresses as 0x1 to 0xE. You may use any Logisim modules for this circuit except that the only modules that you may use from Logisim's Memory library are D Flip-Flop and Register.

Inputs:

DINA[15:0], SELOUTA[3:0], SELOUTB[3:0], SELIN[3:0], WR, CLK

Important note: These input pins must be on the left side of your circuit in the preceding order from top to bottom.

Outputs:

DOUTA[15:0], DOUTB[15:0]

Important note: These output pins must be on the right side of your circuit in the preceding order from top to bottom.

Functionality Description:

SELOUTA[3:0], SELOUTB[3:0] and SELIN[3:0] can only take values from 0x1 to 0xE.

DOUTA[15:0] = value stored at the location with address SELOUTA[3:0]

DOUTB[15:0] = value stored at the location with address SELOUTB[3:0]

At a rising edge on CLK and if WR=1 and SELIN[3:0] is in the range 0x1 to 0xE then the circuit should store the value on DINA[15:0] into location with address SELIN[3:0].

3. A circuit named *DATAPATH* as per the following description:

This circuit uses one instance each of ALU and REG to create a complete datapath in which one can do computation.

Inputs:

DIN[15:0], INSTR[15:0], CLK

Important note: These input pins must be on the left side of your circuit in the preceding order from top to bottom.

Outputs:

DOUT[15:0], FZER, FNEG, and FOVF

Important note: These output pins must be on the right side of your circuit in the preceding order from top to bottom.

Functionality Description:

The basic idea is that INSTR[15:0] is an instruction to the datapath, with the top four bits of instruction, i.e. INSTR[15:12], indicating precisely what the datapath has to do at the next clock tick. For the most part it mimics what the ALU has to do, except that now we also has to specify where the values for ALU operands are coming from and where would the result go.

If INSTR[15:12] corresponds to a valid values for ALU's OPCODE[3:0] input, (i.e. 0000 through 0010) the datapath should at the next clock tick do the corresponding ALU operation. The following describes where the operands come from and where the results of the ALU operation get stored.

- INSTR[11:8] would specify the register from which ALU gets its first operand (i.e. the value on DINA[15:0] of ALU).
    a. If INSTR[11:8] = 0x0 then the first operand to ALU should be 0x0000.
    b. If INSTR[11:8] = 0xF then the first operand to ALU should be the value at DIN[15:0] of the datapath.
    c. Otherwise the first operand to ALU should be value from REG at address given by INSTR[11:8]
- Likewise INSTR[7:4] would specify the register from which ALU gets its second operand (i.e. the value on DINB[15:0] of ALU).
    d. If INSTR[7:4] = 0x0 then the second operand to ALU should be 0x0001.
    e. If INSTR[7:4] = 0xF then the second operand to ALU should be the value at DIN[15:0] of the datapath.
    f. Otherwise the second operand to ALU should be value from REG at address given by INSTR[7:4]
- Finally, INSTR[3:0] specify whether to store the results from the ALU in any register, and if so, where.
    g. If INSTR[3:0] = 0x0 or 0xF then the output is not stored anywhere
    h. Otherwise the ALU output is stored in the register bank REG at address given by INSTR[3:0]

If INSTR[15:12] corresponds to values other than valid values for ALU's OPCODE[3:0] input then the datapath should not affect any register values.

Lastly,

DOUT[15:0] should always be the same as the output of the ALU.

FZER, FNEG, and FOVF should equal the corresponding outputs from the ALU.

4. A circuit named *TESTBENCH_DP* as per the following description:

This is where you'd test your datapath. Make an instance of the datapath, and hook it up as follows:

- Connect DOUT[15:0] to four hexadecimal 7-segment displays
- Connect FZER, FNEG, and FOVF to three LEDs, which should light up when the signals are 1.
- Connect DIN[15:0] and INSTR[15:0] to input pins that you can set by poking.
- Connect CLK to a clock source
- Disable clock tick in Logisim so that you can then manually step the clock tick by tick.

Now using this testbench circuit test your datapath for some simple sequence of instructions.

For example, consider what happens if we set

INSTR[15:0] = 0x1002 and DIN[15:0] is set to some arbitrary value.

Well, INSTR[15:12] = "0001", which means we're asking to do AND. Also, INSTR[11:8] is "0000", which means the first operand to ALU is 0x0000. Similarly, INSTR[7:4] is "0000", which means the first operand to ALU is 0x0001. So the ALU will produce bitwise AND of 0x0000 and 0x0001, which is 0x0000. Finally, INSTR[3:0] is "0010", which means that the result of ALU are to be put into location with address 2 in the register bank. So after the clock edge, the preceding instruction will put the value 0x0000 into register #2. You'd also see 0x0000 at the output on the seven segment displays. Also, FZER will be 1 while FNEG, and FOVF will both be 0.

Likewise INSTR[15:0] = 0x2003 will do an OR of 0x0000 and 0x0001, and put the result in register #3. You'd also see 0x0001 at the output on the seven segment displays. Also, FZER, FNEG, and FOVF will all be 0.

Then let us say we do INSTR[15:0] = 0x5331. Well, now INSTR[15:12] = "0101", which indicates an ADD. It will add values from registers #3 (which we had set to 1) to that from #3 (which we had set to 1), and put the result in register #1. So after the clock edge register #1 will have the value 0x0002, and we'd see 0x0002 at the output on the seven segment displays. Also, FZER, FNEG, and FOVF will all be 0.

Think of other similar sequences of instructions to test that your circuit works correctly for various operations. You are free to add additional components to the testbench to test your design more easily according to your linking. For example, may be you can use the keyboard input module in Logisim to make it easier to enter a sequence of instructions, or perhaps use a ROM to play out a set of instructions from a file.

What to submit?

Please follow these instructions carefully.

1. Save your design with the name *dhw2*. This will create a file called *dhw2.circ.*

2. Additionally create a single PDF file named *dhw2.pdf* with the following

- Your name and student id on the top of the first page
- Color screen capture of Logisim showing schematics of each of the circuits.

3. Use the free screen video capture software at http://www.techsmith.com/jing.html to create a short video (< 5 min, which is the limit of the free softwate) of you showing Logisim working with the circuit on some sample inputs. Include a URL to the video in the PDF.

Note that we will grade your submission by also testing on some sample sequences of our own. So it is extremely important to stick exactly with the names of circuits and input/output signals.

Put both *dhw2.circ* and *dhw2.pdf* (and any other additional file you deem necessary) into a single folder with the name *dhw2_yourlastname_yourfirstname* (e.g. *dhw2_smith_john*), and then zip that folder to get a file named *dhw2_yourlastname_yourfirstname.zip.* Upload this file by logging in at http://nesl.ee.ucla.edu/courses/eem16/2013w/submissions using the password and id that were given in the first lecture.