

**University of California**  
**Department of Electrical and Computer Engineering**  
**ECE 156B**  
**Synthesis & CAD**

**Homework 4**

**DUE: In Class, Tuesday, Mar 9**

**If you have not used your 1 week extension, you can turn in HW 4 on Tuesday, Mar 16 by 5pm in my office**

*A Simple Processor*

Turn in the following (hardcopy in class or to my office, email to lylee@ece.ucsb.edu):

- RTL Verilog version of your processor (hardcopy + email)
- Waveform of RTL simulation (hardcopy only)
- Design Compiler area and timing reports (hardcopy + email)
- Final Gate Verilog version of your processor generated by DC (hardcopy + email)
- Waveform of Gate simulation (hardcopy only)

Note: This specification is different from the simple processor you did in ECE 156A

For this homework, you are to design, simulate, and synthesize a simple processor. We will also introduce timing, and so you may want to let your processor perform faster by introducing a pipeline (use no more than a 4 stage pipeline – Decode, Read Registers, Compute, Store). If you do decide to include a pipeline, be sure you do not have any data hazards as learned in ECE 154.

HW 4 will be worth 30 points. 20 points will be distributed to correctness of your code and simulations. The last 10 points will be based on your circuit size and circuit performance. This will make HW 0 worth 5% of your total grade, HW 1-3 each worth 10% of your total grade, and HW 4 worth 15% of your total grade. The midterm is worth 20% and the final 30%.

**Processor specifications:**

- All storage elements must use the positive edge of clock
- There are 8 registers indexed by 3 bits (ie. register 000, register 001, ..., register 111)
- The data width is 8 bits, all integers are unsigned
- An 8 bit data\_out signal which is don't care except during Access Register and Conditions
- A 1 bit overflow and 1 bit underflow signal, 1 when overflow/underflow happens, 0 otherwise
- The instruction width is 13 bits, with the leftmost bit being bit 12
  - Opcode is in bits 12-11 (Opcode 01 means 0 in bit 12, and 1 in bit 11)
- The functions your processor needs to perform are defined as follows:
  - Load Immediate
    - Opcode: 00
    - Bits 10-8: register to load into
    - Bits 7-0: data to be loaded
  - Access Register
    - Opcode: 11
    - Bits 2-0: register to access, output register contents to data\_out

- Operations
  - Opcode: 01
  - Bits 10-9: 00 for addition, 01 for subtraction, 10 for bit-wise or, 11 for and
  - Bits 8-6: target register
  - Bits 5-3: source A register
  - Bits 2-0: source B register
  - target = A (operation) B
- Conditions
  - Opcode: 10
  - Bits 10-9: 00 for ==, 01 for <, 10 for >, 11 for !=
  - Bits 8-6: target register (00000000 if false, 00000001 if true)
  - Bits 5-3: source A register
  - Bits 2-0: source B register
  - target = A (condition) B

When writing your test bench, be sure to comment what instruction you are executing, and what your expected result (if any) in data\_out and ovr/und will be. Be sure to exercise each calculation and register as well. Finally, your test bench needs to account for register hold time and setup time, so don't change your values directly at the clock edge (something a lot of students have had trouble with in their previous homework).

### Timing:

When synthesizing in Design Compiler, before you optimize/map, specify a clock signal to your clk as you learned how to do in the tutorial. Each gate in class.db has a delay, so the timing report will provide you with the longest timing path in the design.

Point	Incr	Path
clock (input port clock) (rise edge)	0.00	0.00
input external delay	0.00	0.00 r
inst[3] (in)	0.00	0.00 r
U1160/Z (IVI)	0.88	0.88 f
U977/Z (NR3)	8.04	8.92 r
U1239/Z (AO1P)	0.35	9.27 f
U995/Z (ND2I)	0.77	10.04 r
add_28/A[0] (processor_DW01_add_9_0)	0.00	10.04 r
add_28/U44/Z (ND2I)	0.28	10.32 f
add_28/U72/Z (IVI)	0.25	10.57 r
... (continues)		

This report will tell you the data arrival time and your clock cycle. 5 points of this homework will be based on how fast your data arrives. In the tutorial, you learned how to specify a tighter clock constraint and let Design Compiler optimize it for timing by increasing area usage. Another option you can try is to spread your logic out over pipeline stages, but note this will also increase your area as you have to introduce storage elements for data that passes between stages.

The last 5 points will be based on your area, so do not just optimize for timing without concern for area. For example, an excellent timing but really bad area might result in 5 points for timing and 1 point for area, while lessening the area at the cost of worse timing might result in 4 points for timing and 4 points for area.