Lec-6-HW-3-ALUarithmetic

Part 1. Reading, PP, Chp 2:
2.1 (Bits and datatypes)
2.2 (signed and unsigned integers)
2.3 (2's complement)
2.4 (positional notation)
2.5 (int. add/sub, signed/unsigned overflow)
2.6 (logic: AND, OR, NOT, XOR)
2.7 (bit vectors, floating point, ascii, hex)

Part 1. Problems, PP, Chp 2:
2.2 (#bits for 26 char)
2.4 (unsigned range of n-bit)
2.5 (5-bit 2's comp., signed mag., and 1's comp.)
2.6 (6-bit 2's comp.)
2.8 (8-bit and n-bit 2's comp. ranges)
2.9 (bits per decimal digit in fp. format)
2.10ab (convert 2's comp. to dec.)
2.11ac (convert dec. to 2's comp.)
2.13 (convert k-bit 2's comp. to 8-bit)
2.15 (what op == shift right?)
2.17ab (i-bit + k-bit 2's comp.)
2.18ab (i-bit + k-bit unsigned)
2.20 (4-bit 2's comp. overflow)
2.34b (AND-OR-NOT)
2.36 (bit masking: BUSYNESS vector)
2.37 (alg. for detecting 4-bit 2's comp. overflow)
2.39a (dec. to fp format)
2.40ab (fp format to dec.)
2.44 (convert bin. to ascii)
2.45ab (convert bin. to hex.)
2.49ab (add in hex. notation)
2.50a (logic in hex. notation)
2.52 [only col. 1](32-bit hex. to unsigned, 1's-2's comp., fp, ascii)
2.56 (define 8-bit fp format)

Part 2. In your LC3, implement the ALU, muxes, and the "plus1" circuit.

(1.) Implement muxes using basic gates (AND, NOT, OR). There are several muxes in the design, but we will only concern ourselves with the PCMUX, and the muxes in the RegFile and in the ALU.

The PCMUX is a 16-bit, 4-input, 1-output mux: inputs and output are 16-bits each; there is a 2-bit select input. It is mux16_4x1 in parts.jelib.

RegFile has two copies of mux16_8x1, one for the SR1out, the other for the SR2out. It is implemented using two mux16_4x1s and a mux16_2x1.

The ALU has a single mux16_4x1.

There are only two parts to modify: mux16_2x1 and mux16_4x1 in parts.jelib. You are to remove the verilog code and blackboxes from these cells and build each part using basic gates. (See notes below for building arrays of elements in Electric.)

(2.) Implement function units in the ALU and the plus1 using basic gates. The ALU contains three 16-bit functional subparts: AND16, ADD16, and NOT16. AND and NOT bitwise operations are very simple.

For ADD16, you will implement a 16-bit full adder. To do that, create a 1-bit full adder, FA1, from basic gates. Then use that as a base element to implement ADD16.

The plus1 part can simply be an ADD16 with appropriate inputs; eg., one 16-bit input will always be 0 and the low-bit carry-in will always be 1.

(3) EXTRA-CREDIT: plus1 can be made using many fewer gates than using ADD16 to implement it. Analyze its boolean logic and implement plus1 directly. Use logic circuit minimization techniques we have learned.

NOTES

(1) Arrays of gates/devices. It is easy to create a 16-bit circuit from 1-bit elements using Electric's Edit.Array command. First, build a 1-bit element; eg., a 1-bit AND with short wires attached to inputs and outputs. Name the wires as bus elements; eg., inA[0], inB[0], and out[0]. Now, select the entire 1-bit device by dragging a select-frame over it. Then, use Edit.Array, and fill in the appropriate number of repetitions of the device (16 in this case). Decide if you want them laid out vertically or horizontally. Recall that Edit.undo is ctl-z in MS Windows and Apple-z in Mac. The wires will be automagically numbered Ain[0], Ain[1], and so forth. All that's left to do is check that the cell's bus ports are still working. If you were careful not to delete the ports and you used the same naming scheme for wires/busses used originally, they should be ok. (But, when you have the same signal going to all instances in your array, as in MUX selects, you will have to manually rename them since they will be numbered as if they form a bus.) If you have to create an entire cell from scratch, remember to set all port's input/output characteristics appropriately, and do View.MakeIconView. You can edit the icon view, "{ic}", to move text around and resize it, add text to inputs and outputs, and move the location of ports. To move ports you may have to Edit.Rotate them and their attached wire/bus (select the pin, not the wire/bus for moving; select both for rotation.)

(2) WARNING: In editing pre-existing cells, be careful not to delete the exports when removing the old implementation. When you edit or delete cells, either schematic or icon cells, you may find your connections to its icon in higher-level cells break, or your exports disappear or get automagically renamed in the schematic. Visually check connections: (1) in your schematic cell, select the export name or its icon's export crosshair, all connected wire/bus nets will be highlighted; (2) in a higher-level cell, select the icon's export crosshairs. In the worst case, you may need to delete the old icon from the cells in which there are instances (including its own schematic cell), make a new icon view, and put the new icon in where the old one was and reconnect it. If you are only editing an existing schematic, you can delete the icon in the cell's schematic without deleting its icon view "{ic}" and then recreate exports that match the cell's icon view's exports.