COMSM1302 Lab Sheet 2

This lab sheet covers content from week 2 lectures, focusing on boolean arithmetic and relevant circuit implementation. On successful completion of the lab, students should be able to:

1. Create subcircuits within Logisim and use them as components within other circuit designs.
2. Assemble circuits that can handle multiple bit inputs and outputs, using splitters.
3. Design circuits that can carry out arithmetic operations, in particular addition and subtraction.
4. Use multiplexers and demultiplexers within a circuit design to make computational decisions based on relevant inputs.

You'll need the Logisim software to implement gates and circuits, and may need to install Java first if running Logisim on your own computer. Details on how to download and run Logisim can be found on the unit page and if you need help with using Logisim, try referring to the Logisim user guide.

You will also need to borrow a NAND board to try physical implementations, which you can find details of on the unit page. If you get stuck using NAND boards, try watching the NAND board explanation or have a look through the GitHub documentation.

Half adder

Create a 1-bit half adder circuit design on Logisim, as defined by the given truth table, using any gates with 2 or fewer inputs. Then, using this as a subcircuit, assemble a 4-bit incrementer that accepts a single input 4-bit signal and outputs a single 4-bit signal.

A	B	C_out	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Hint: A splitter can be used to separate a multi-bit signal into 1-bit signals or vice versa. To build the incrementer, you'll need 2 splitters (each with 4 fan out signals) as well as a constant 1, both found in the Wiring library on Logisim.

Full adder

Create a 1-bit full adder circuit design on Logisim, as defined by the given truth table, using any gates with 2 or fewer inputs. Then, using this as a subcircuit, assemble a 4-bit adder that accepts two 4-bit input signals and outputs a single 4-bit signal.

Hint: A splitter can be used to separate a multi-bit signal into 1-bit signals or vice versa. To build the 4-bit adder, you'll need 3 splitters (each with 4 fan out signals) found in the Wiring library on Logisim.

Challenge: Can you adapt your 4-bit full adder so it can subtract as well as add? **Hint:** You'll need a multiplexer and an understanding of 2's complement!

C_in	A	B	C_out	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

NAND adder

Using principles covered in the NAND lecture, assemble an implementation of each of the following circuits on Logisim (using only 2-input NAND gates) then test on a NAND board:

- 1-bit half adder
- 1-bit full adder

As an *optional* task, try chaining your NAND board full adder up with another student's! They will need to be connected to the same power supply, using a USB splitter provided in the lab.

Challenge: Can you create a 3-bit bitwise incrementer just using 2-input NAND gates? Test out your design on a NAND board!

Plexers

Assemble a multiplexer with a 2-bit control input C with the following behaviour, where A and B are both 4-bit inputs:

- If C is 0, output A AND B
- If C is 1, output A OR B
- If C is 2, output A XOR B
- If C is 3, output A NOR B

Hint: You will need to ensure you have changed the numbers of bits in the attributes of each component you use and it's a good idea to ensure the *three-state?* attribute of each of your input pins is No.

Plexers also have an enable input which, when not 0, enables the plexer to function. Add a demultiplexer with a 1-bit control input D and 1-bit input E to your design so that your multiplexer is only enabled when the demultiplexer's control input D is 0 and input E is 1.

Arithmetic Logic Unit

Build a Hack ALU on Logisim, that includes 6 multiplexers with the following 1-bit control bits:

1. **zx**: if zx == 0 then x = x, else x = 0
2. **nx**: if nx == 0 then x = x, else x = ¬x
3. **zy**: if zy == 0 then y = y, else y = 0
4. **ny**: if ny == 0 then y = y, else y = ¬y
5. **f**: if f == 0 then out = x&y, else out = x+y
6. **no**: if no == 0 then out = out, else out = ¬out

The signals x, y, and *out* should be 4-bits and you should use your 4-bit full adder as a subcircuit within this design. Test your design is correct using the given table, which lists **some** combinations of values of the control bits.

Then, connect 1-bit comparison outputs to your ALU design, that will report the following:

1. **zr**: if out == 0 then zr = 1, else zr = 0
2. **ng**: if out < 0 then ng = 1, else ng = 0

zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

Challenge: Can you explain why each of these series of calculations produces the output defined in this table?