COMS12200 lab. worksheet: week #4

- We intend this worksheet to be attempted in the associated lab. session, which represents a central form of help and feedback for this unit.
- The worksheet is not *directly* assessed. Rather, it simply provides guided, practical exploration of the material covered in the lecture(s): the only requirement is that you archive your work (complete or otherwise) in a portfolio via the appropriate component at

  https://wwwa.fen.bris.ac.uk/COMS12200/

  This forms the basis for assessment during a viva at the end of each teaching block, by acting as evidence that you have engaged with and understand the material.
- The deadline for submission to your portfolio is the end of the associated teaching block (i.e., in time for the viva): there is no requirement to complete the worksheet in the lab. itself (some questions require too much work to do so), but there is an emphasis on *you* to catch up with anything left incomplete.
- To accommodate the number of students registered on the unit, the single 3 hour timetabled lab. session is split into two $1\frac{1}{2}$ hour halves. You should attend *one* half only, selecting as follows:
  1. if you have a timetable clash that means you *must* attend one half or the other then do so, otherwise
  2. execute the following BASH command pipeline

     ```
     id -n -u | sha1sum | cut -c-40 | tr 'a-f' 'A-F' | dc -e '16i ? 2 % p'
     ```

     e.g., log into a lab. workstation and copy-and-paste it into a terminal window, then check the output: 0 means attend the first half, 1 means attend the second half.

Q1. This worksheet focuses on using Logisim[1], which is a Java-based application for designing and simulating the behaviour of circuits: the premise is that actively using transistors makes them easier to understand than just reading about them.

This question is not really a question: it is designed as an introduction to features in Logisim required later by this worksheet. To support further work throughout the unit as a whole, the emphasis is on *you* to explore wider documentation available at

http://sourceforge.net/projects/circuit/support/

as and when needed. The first step is to actually execute Logisim: you can *either*

a download the software from

  http://sourceforge.net/projects/circuit/

  and install it yourself (since Logisim is written in Java, this just means using the downloaded JAR file rather than any actual installation), *or*

b use a pre-downloaded version (which is preferably since it will save you ~ 7 MB of disk space) by simply issuing the command

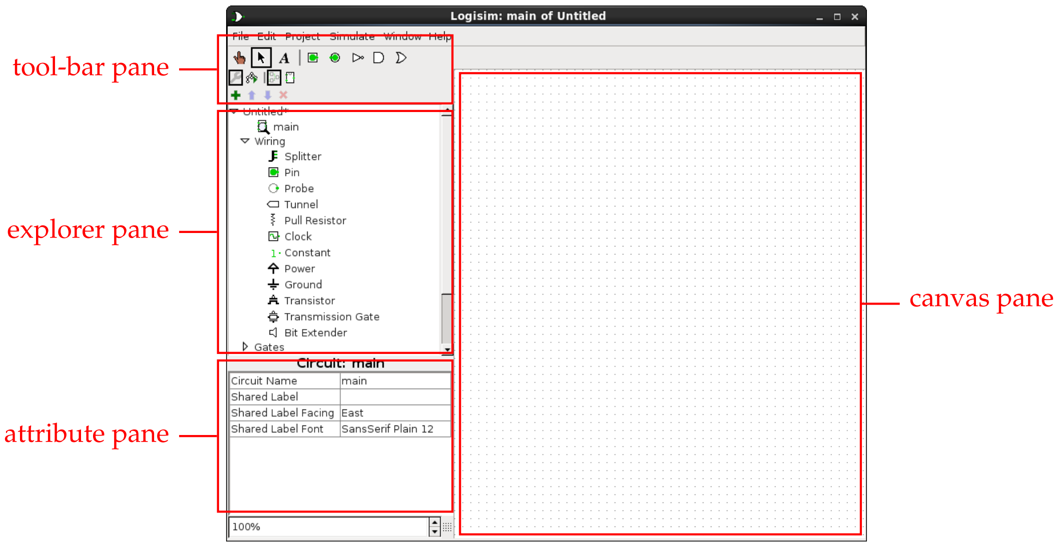
  ```
  java -jar ~page/local/generic/logisim-generic-2.7.1.jar
  ```

  from a BASH prompt (i.e., from a terminal window, and noting the tilde character before page which should be read as "the home directory of the user named page").

Once you do this, you should see a window similar to Figure 1a; notice that the Figure includes annotation explaining each part of the window. Briefly, these parts include the following:

- The canvas pane is where we will "draw" the circuit itself; this process is helped by a grid, which aligns component instances we place on the canvas.
- The tool-bar pane selects the way you interact with the canvas, e.g., what clicking on it does. It includes icons that allow you to
  - alter the state of input components (the "hand" tool),
  - place, select and alter components and wires (the "arrow" tool),

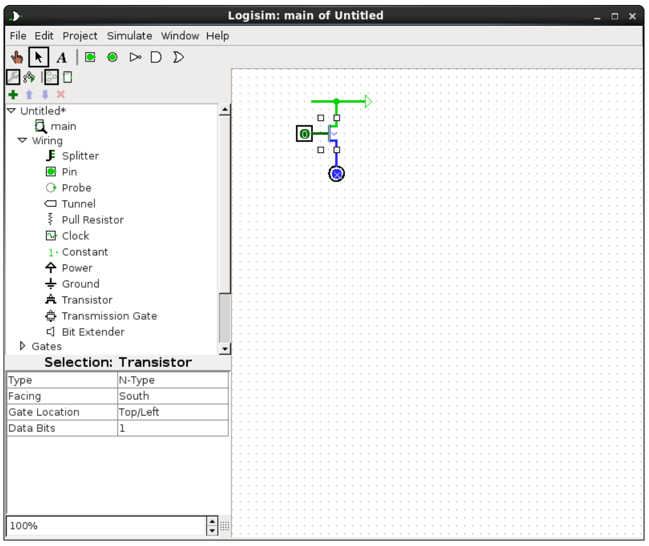
[1] We will focus exclusively on version 2.7.1 of Logisim, but other versions should work in a roughly similar way.



Circuit: main

Circuit Name	main
Shared Label	
Shared Label Facing	East
Shared Label Font	SansSerif Plain 12

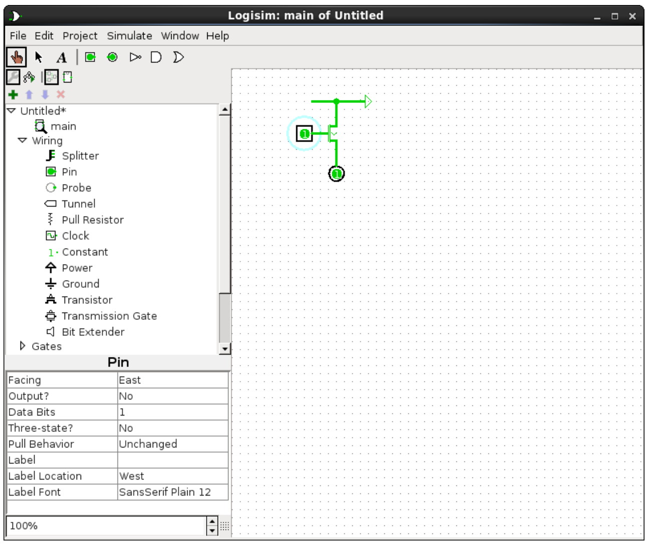
(a) *The initial Logisim window, annotated to show the major features.*



Selection: Transistor

Type	N-Type
Facing	South
Gate Location	Top/Left
Data Bits	1

(b) *A simple example circuit including a single transistors, one input and an output ...*



Pin

Facing	East
Output?	No
Data Bits	1
Three-state?	No
Pull Behavior	Unchanged
Label	
Label Location	West
Label Font	SansSerif Plain 12

(c) *... the same example, but with the input flipped from* 0 *to* 1.

Figure 1: *A (very) simple example circuit using Logisim.*

- place input components (the "square pin" tool), and
- place output components (the "round pin" tool).

- The explorer pane allows you to browse through and select from the components available: selecting a component allows placement of an instance on the canvas.
- Once a component instance on the canvas is selected, features that describe how it behaves are shown in the attribute pane; these can be altered in order to change this behaviour, and hence control aspects of the circuit on the canvas.

Figure 1b illustrates a simple example where a single MOSFET transistor has been placed on the canvas. Note that since the transistor is currently selected, the attribute pane shows it is an N-MOSFET and also south facing. This latter attribute is important, since it determines the direction a voltage level can flow: an arrow inside the component itself is used to visualised this on the canvas. The top terminal is connected to the V_{dd} voltage level using a wire; the bottom terminal is connected to an output pin, and the gate to an input pin. In this case, the downward, south-facing direction of the N-MOSFET transistor means voltage can flow from the top to bottom terminal when the gate is connected to 1. The gate is currently shown connected to 0 (which is illustrated on the canvas) meaning the output at the bottom terminal is X (or unknown). If the gate is flipped (using the "hand" tool to toggle the input pin) so it is connected to 1, we end up with Figure 1c. Now the top and bottom terminals are connected, so the output is 1.

Notice that as you change the input, causing Logisim to simulate the circuit and so produce the output, wires on the canvas will change colour to illustrate their value[2]. Also notice that input and output pins can be configured as 3-state (i.e., capable of taking the values 0, 1 or unknown) or not via the attributes pane, and may represent larger than 1-bit values if need be (although this is not useful here). Like transistors, the direction they face in is important since this dictates where the connection pin will be. Finally, keep in mind that Logisim has various tools to annotate your design; these include the "text" tool (next to the "arrow" tool) for placing arbitrary labels, and the label attribute of most components. Use of either is a good idea, both to make sure you remember how the design works *and* to support submission as part of your portfolio.

Simply as a first step, reproduce this example yourself: experiment with drawing wires, changing the transistor type, input values and so on.

Q2. In the lecture(s), we saw that NOT, NAND and NOR gates could be constructed using transistors, which are basically just switches. In more detail, the transistors are arranged to from a combination of

- a pull-up network of P-MOSFET transistors, that can set the output to 1 (by connecting it to the V_{dd} power rail), and
- a pull-down network of N-MOSFET transistors, that can set the output to 0 (by connecting it to the *GND* power rail).

Your challenge in this question is to apply this theory, using Logisim to implement *and* simulate (in order to test)

a a NOT gate with input x and output r,

b a NOR gate with inputs x and y and output r, and

c a NAND gate with inputs x and y and output r.

Make sure you save your work as you go so you can come back to it if need be; once you are confident each gate works correctly, try to package it as a Logisim sub-component[3] that you can easily reuse.

[2] The light green wire means 1, dark green means 0, blue means X or unknown (or disconnected, which basically matches the concept of a high impedance value we discussed in the lecture), red means E or error (or conflict), orange means a data type mismatch (e.g., a n-bit wire is connected to an input expected m bits where $n \neq m$, which is analogous to a type error in C), and black simply means this is a vector of wires (i.e., an n-bit rather than 1-bit value).

A red wire is clearly problematic: it means the simulator cannot tell which value the wire should take because it is driven simultaneously by two *different* values (maybe *GND and* V_{dd} for example). Likewise, the mismatch highlighted by an orange wire potentially needs to be fixed before simulation can be performed reliably.

[3] By default, there is one component in a Logisim called `main`; you can think of it roughly like the `main` function in a C program, in the sense it acts as the top-level (or entry point) in your design.

The `Project` menu allows management of this and other user-defined components. For example, the `Add Circuit` item allows addition of a new, named component to the project. Double-clicking on the new entry shown in the explorer pane displays a new canvas associated with the new component: by implementing a circuit on this canvas (labelling the input and output pins if you want) you can use it just like the transistors (or indeed any other component) used so far.

Q3. Exercise the universality of NAND and NOR: using your components from the question above, implement and simulate both forms of

a an OR gate, and

b an AND gate

both of which have inputs x and y and output r.

Q4. In the previous questions, you implemented 2-input, 1-output NOR, NAND, OR and AND gates. It is common, however, to require gates with *more* than 2 inputs: we might want to compute the NAND of x, y and z for example.

a Write out the truth table specifying the behaviour you expect for a 3-input NAND gate.

b A natural first attempt might use two 2-input NAND gates to compute

$$x \overline{\wedge} y \overline{\wedge} z,$$

but this is incorrect: prove this is the case, and explain why (versus a similar scenario using AND instead of NAND, for example).

c Irrespective of the above, first design, then implement and simulate a *correct* 3-input NAND gate in Logisim by *directly* using MOSFET transistors.

d If you needed an n-input, 1-output NAND gate for example, can you articulate the design strategy needed (i.e., the pattern that emerges in terms of how the transistors are arranged)?

Q5[+]. This is an extended rather than core question: it caters for differing backgrounds and abilities by supporting study of more advanced topics, but is therefore significantly more difficult and open-ended. As such, you should *only* attempt the associated tasks having completed all core questions (which satisfy the unit ILOs) first; even then, there is no shame in ignoring the question, or deferring work on it until later. Note that a solution will not *necessarily* be provided.

Throughout the lecture(s), we focussed on implementing logic gates by using transistors. The motivation for doing so is the dominance of this approach in current micro-electronics: it would be fairly unusual for you to own a device *not* based on some form of transistor.

However, it is important to see that Boolean algebra (and hence computation) is *independent* from this implementation technology. Put another way, we *could* realise the Boolean operators using a technology *other* than transistor-based logic gates; the result would be a computer, although perhaps not with the same characteristics we are used to. Two examples of "other" technologies are

- fluids, e.g.,

  https://www.youtube.com/watch?v=NKxFG21vw_I

  or

- pulleys, e.g.,

  http://vimeo.com/93042377

Based on this,

a try to assess how (and why) the characteristics of a computer based on these examples might differ from one that uses transistors (e.g., list any obvious advantages or disadvantages of each), then

b try to think of (or find) any other viable "other" technologies.