COMS12200 lab. worksheet: week #5

- 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. There is a set of questions available directly at

http://www.cs.bris.ac.uk/home/page/teaching/material/arch_old/sheet/exam-logic_minimise.q.pdf

or via the unit web-page: using pencil-and-paper, each asks you to translate a truth table into a Boolean expression, e.g., by using the Karnaugh map technique. There are *way* too many for the lab. session(s) alone, but in the longer term your challenge is simple: answer a few questions regularly (e.g., ten random questions a week) in order to practice and hence master this technique.

Q2. In the lecture(s), we discussed two building-block components:

a a 1-bit, 2-input **multiplexer** with inputs x, y (the choices) and c (the control signal) and output r (the result), and

b a 1-bit, **full-adder** with inputs x, y (the summands) and ci (the carry-in) and outputs s (the sum) and co (the carry-out).

For each component,

- use the built-in[1] logic gates provided by Logisim to produce a simulated implementation, then
- package your implementation as a Logisim sub-component to allow easy reuse, and
- translate the design into NAND-only form, and produce a physical implementation using your NAND board

testing the result at each step to ensure it yields the expected behaviour.

Q3. In the lecture(s) we covered the design of a **ripple-carry adder** which computes the sum $r = x + y$ for n-bit summands x and y; it does so by mirroring long-hand addition, using n full-adder instances linked by a carry chain.

[1] The suggestion to use the built-in logic gates is simply to limit dependencies between worksheets, and the volume of work required: it is important to keep in mind that you *could* use the components you developed for the worksheet in week #4 instead.

Logisim allows instances of AND and OR with more than two inputs, whereas doing so by hand is a little tedious and veers away from the worksheet scope. Take care however: the default number of inputs seems to be 5 (!), and undefined (i.e., unconnected) inputs might yield an unexpected output. You can use the attribute pane to change this and other features of each gate instance.

a Use your full-adder component from the previous question to implement a simulated 4-bit ripple-carry adder in Logisim.

b Although a full-*subtractor* was not covered in the lectures(s), such a component forms the design of a ripple-carry *subtractor* in the same way as above (but with components now linked by a borrow chain). First design and implement a a 1-bit full-subtractor, then implement an analogous simulated 4-bit ripple-carry subtractor in Logisim.

c One NAND board has enough components for only few 1-bit, full-adder instances; with n NAND boards however, you can implement a physical n-bit ripple-carry adder. Find $n - 1$ other students to work with, and combine your NAND boards to do so: what is the largest value of n you can manage to produce results for?

Q4. The following questions are more difficult to address using the NAND board. Instead, the emphasis in each is a *design* challenge: whereas each of the previous questions provided a set of steps to follow, this one demands that *you* design each component from scratch by applying theory from the lecture(s).

a To left-rotate an n-bit sequence x by a distance of one bit, the most-significant, $(n - 1)$-th bit of x is first removed; the bit is concatenated onto the least-significant end of the $n - 1$ remaining bits to form the n-bit result. More generally, left-rotation by a distance of d bits repeats this process d times. For example, consider the 28-bit literal

$$1001100110011001100110011001.$$

After left-rotating it by a distance of one bit, the result is

$$0011001100110011001100110011,$$

and left-rotating again by one bit (meaning a total distance of two bits) yields

$$0110011001100110011001100110.$$

Consider a component that has two inputs: x is a 28-bit sequence to be rotated, and y is an unsigned, 4-bit integer that controls the rotation distance. Specifically, the required distance is

$$d = \begin{cases} 1 & \text{if } y \in \{0, 1, 8, 15\} \\ 2 & \text{otherwise} \end{cases}$$

bits, where clearly d is a function of y. First design such a component, then implement and simulate your design in Logisim to verify it works correctly.

b In cryptography, an S-box (or "substitution box") component can be viewed as a Boolean function

$$f : \{0, 1\}^n \to \{0, 1\}^m.$$

That is, an n-bit input x is translated into (or substituted for) the m-bit output $r = f(x)$. Provided n and m are reasonably small, one can write out the entire function in the form of a look-up (or truth) table.

A concrete example is the S-box used by PRESENT[2], a block cipher. It sets $n = m = 4$, meaning the translation can be described by

x	$r = f(x)$
$0_{(16)}$	$C_{(16)}$
$1_{(16)}$	$5_{(16)}$
$2_{(16)}$	$6_{(16)}$
$3_{(16)}$	$B_{(16)}$
$4_{(16)}$	$9_{(16)}$
$5_{(16)}$	$0_{(16)}$
$6_{(16)}$	$A_{(16)}$
$7_{(16)}$	$D_{(16)}$

x	$r = f(x)$
$8_{(16)}$	$3_{(16)}$
$9_{(16)}$	$E_{(16)}$
$A_{(16)}$	$F_{(16)}$
$B_{(16)}$	$8_{(16)}$
$C_{(16)}$	$4_{(16)}$
$D_{(16)}$	$7_{(16)}$
$E_{(16)}$	$1_{(16)}$
$F_{(16)}$	$2_{(16)}$

noting that the entries are in hexadecimal.

i Using suitable Karnaugh map(s), derive a Boolean expression $r_i = f_i(x)$ for each bit i-th bit of the output.

ii PRESENT is a light-weight block cipher design, implying use within devices such as RFID tags where metrics such as area and power consumption are important: optimise your design above to minimise the overall gate count.

iii Implement and simulate your design in Logisim to verify it works correctly.

[2] http://www.iacr.org/archive/ches2007/47270450/47270450.pdf