COMS12200 lab. worksheet: week #7

- 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 lab. session shown in your timetable is split into two $1\frac{1}{2}$ hour halves. You should only attend *one* half, 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, whereas 1 means attend the second half.
- Keep in mind that hardware and software within the lab. is managed by the IT Services Zone E team. If you encounter a problem (e.g., a workstation that cannot be powered-on, an error when executing some software, some missing software, or you just cannot log into your account), *they* can help: either talk to them in room MVB-3.41, and/or submit a service request online via

  http://servicedesk.bristol.ac.uk

Like the worksheet from week #6, the Logisim project

http://www.cs.bris.ac.uk/home/page/teaching/material/arch_new/sheet/lab-7.q.circ

provides a 2-phase clock component. Download and use this project as a starting point, adding to it within each of the following questions.

Q1. Using the 2-phase clock component provided, produce a simulated implementation of the 4-bit controlled counter we discussed in the lecture(s): recall that this component was designed to replicate the behaviour of a C-style `for` loop. Adopting a step-by-step approach is sensible: first

- implement the data-path and test it by manually controlling each of the inputs, (e.g., ensure it can increment and compare i as required), then
- implement the control-path and test it by manually controlling each of the inputs (e.g., ensure it transisions between states correctly), then
- integrate the two halves, and make sure the result behaves as expected under control of a clock.

Either way, for both this and the next question, use any appropriate built-in[1] components provided by Logisim; package your implementation as a sub-component to allow easy reuse.

[1] The suggestion to use the built-in ripple-carry adder and register components, for example, 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 #5 instead. However, given said components are becoming more complex, there are some issues to beware of:

- The built-in multiplexer includes an input labelled `enable`, but you can ignore this (or just turn it off). The input labelled `select` is what we termed the control signal: Logisim uses a *single* control signal whose size (i.e., number of bits) is derived from the number of inputs being selected between.
- The built-in register is based on flip-flops (so is edge-triggered by default): to match the material we covered, turn this into a latch-based alternative (meaning it is level-triggered). Ignore the two inputs labelled `clear` and `enable`, since they just allow more complex control of the component than we covered: instead, focus on the input labelled `clock` which acts as the enable signal we termed *en*.
- The built-in comparator interprets both inputs as signed, two's-complement integers, but you can and should turn this into an unsigned comparison.

Q2. Given unsigned n-bit operands x and y, an $(n \times n)$-bit multiplication yields an unsigned $2n$-bit product $r = x \cdot y$. It can be useful to truncate r, leaving just the least-significant n-bit half (whose size then matches x and y): both the ARM and MIPS32 `mul` instructions do this for $n = 32$. For smaller operands, namely for $n = 8$, implement

a a simulated combinatorial tree multiplier, then

b a simulated iterative bit-serial multiplier

both of which should compute the truncated 8-bit product r from 8-bit x and y. For the latter, use the counter implementation from the previous question as a starting point: since the design is iterative, it should compute r under control of the counter and hence the associated control protocol.

As an aside, ensuring you understand the bit-serial approach before implementing it would be a good idea. Experimenting with a software implementation of the algorithm, e.g.,

```
uint8_t mul( uint8_t x, uint8_t y ) {
  uint8_t t = 0;

  for( int i = ( BITSOF( y ) - 1 ); i >= 0; i-- ) {
    t = t << 1;

    if( ( y >> i ) & 1 ) {
      t = t + x;
    }
  }

  return t;
}
```

might help in some cases: it allows you, for example, to inspect intermediate values and perhaps debug your Logisim implementation more easily. Note the `BITSOF` macro is as per the worksheet in week #3, i.e., here it is used to compute the number of bits used to represent `y`.

Q3[+]. 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.

In the lecture(s) we (briefly) covered the use of Booth recoding; this was presented at a fairly conceptual level, rather than via a concrete algorithm or circuit design. By first producing such a design, extend your existing multiplier implementation (either combinatorial and/or bit-serial) to support this optimisation.