- Remember to register your attendance using the UoB Check-In app. Either
  1. download, install, and use the native app[a] available for Android and iOS, or
  2. directly use the web-based app available at

     https://check-in.bristol.ac.uk

  noting the latter is also linked to via the `Attendance` menu item on the left-hand side of the Blackboard-based unit web-site.
- The hardware *and* software resources located in the MVB Linux lab(s). (e.g., MVB-1.15 or MVB-2.11) are managed by the Faculty IT Support Team, a subset of IT Services. If you encounter a problem (e.g., a workstation that fails to boot, an error when you try to use some software, or you just cannot log into your account), they can help: you can contact them, to report then resolve said problem, via

  https://www.bristol.ac.uk/it-support

- The lab. worksheet is written *assuming* you work in the lab. using UoB-managed and thus *supported* equipment. If you need or prefer to use your own equipment, however, various *unsupported*[b] alternatives available: for example, *you* could 1) manually install any software dependencies yourself, *or* 2) use the unit-specific Vagrant[c] box by following instructions at

  https://cs-uob.github.io/COMS10015/vm

- The questions are roughly classified as either C (for core questions, that *should* be attempted within the lab. slot), A (for additional questions, that *could* be attempted within the lab. slot), or R (for revision questions). Keep in mind that we only *expect* you to attempt the C-class questions: the other classes are provided *purely* for your benefit and/or interest, so there is no problem with nor penalty for totally ignoring them.
- There is an associated set of solutions is available, at least for the C-class questions. These solutions are there for you to learn from (e.g., to provide an explanation or hint, or illustrate a range of different solutions and/or trade-offs), rather than (purely) to judge your solution against; they often present *a* solution vs. *the* solution, meaning there might be many valid approaches to and solutions for a question.
- Keep in mind that various mechanisms exist to get support with and/or feedback on your work; these include both in-person (e.g., the lab. slot itself) *and* online (e.g., the unit forum, accessible via the unit web-site) instances.

[a] https://www.bristol.ac.uk/students/support/it/software-and-online-resources/registering-attendance

[b] The implication here is that such alternatives are provided in a best-effort attempt to help you: they are experimental, and so *no* guarantees about nor support for their use will be offered.

[c] https://www.vagrantup.com

COMS10015 lab. worksheet #9

During the period of time aligned with this lab. worksheet, there is an active (or open) coursework assignment for the unit. You could address this fact by dividing your time between them. However, our (strong) suggestion is to view the former as of secondary importance (or optional, basically), and instead focus on the latter: since it is credit bearing, the coursework assignment should be viewed as of primary importance. Put another way, focus exclusively on completing the latter before you invest any time at all in the former.

Before you start work, download (and, if need be, unarchive[a]) the file

https://assets.phoo.org/COMS10015_2024_TB-4/csdsp/sheet/lab-09_q.tar.gz

somewhere secure[b] in your file system; from here on, we assume `${ARCHIVE}` denotes a path to the resulting, unarchived content. The archive content is intended to act as a starting point for your work, and will be referred to in what follows. In common with lab. worksheet #4, note that the archive provides a user-defined a 2-phase clock generator component.

[a]For example, you could 1) use `tar`, e.g., by issuing the command `tar xvfz lab-09_q.tar.gz` in a terminal window, 2) use `ark` directly: use the `Activities` desktop menu item, search for and execute `ark`, use the `Archive`→`Open` menu item to open `lab-09_q.tar.gz`, then extract the contents via the `Extract` button, or 3) use `ark` indirectly: use the `Activities` desktop menu item, search for and execute `dolphin`, right-click on `lab-09_q.tar.gz`, select `Open with`, select `ark`, then extract the contents via the `Extract` button.

[b]For example, the `Private` sub-directory within your home directory (which, by default, cannot be read by another user).

§1. C-class, or core questions

▷ **Q1[C].** In the lecture slot(s), we studied the design of a component whose behaviour models the loop counter used by a C-style `for` loop. Assuming $n = 4$, use LogisimEvo to implement and simulate a variant of this design based on

a latches, *or*

b flip-flops.

The (strong) recommendation is to make use of built-in components provided by LogisimEvo, and adopt a step-by-step approach:

- implement the data-path, then test it by manually controlling all the inputs (e.g., ensure it can increment and compare i as required),
- implement the control-path, then test it by manually controlling all of the inputs (e.g., ensure it transitions between states correctly),
- integrate the two halves, then test the whole implementation using an automatically controlled clock,
- package your implementation as a sub-component to allow easy reuse.

▷ **Q2[C].** Given unsigned n-bit operands x and y, an $(n \times n)$-bit multiplication yields an unsigned $2n$-bit product $r = x \cdot y$. However, it can be useful to truncate r, leaving just the least-significant n-bit half (whose size then matches x and y). Assuming $n = 8$, use LogisimEvo to implement and simulate

a a combinatorial tree multiplier, then

b an iterative bit-serial multiplier

which can compute a 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. Exploring bit-serial multiplication using a C-based reference implementation *might* help before or during development of your LogisimEvo implementation: doing so might help to improve your understanding, for example, and also act as a means of generating a trace of intermediate values to compare against. Figure 1 provides such a reference implementation, noting that the `BITSOF` macro matches lab. worksheet #2: it computes the number of bits used to represent `y` without relying on the data-type.

```
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;
}
```

Figure 1: *A C-based software implementation of left-to-right, bit-serial multiplication.*

§2. R-class, or revision questions

▷ **Q3[R].** There is a set of questions available at

https://assets.phoo.org/COMS10015_2024_TB-4/csdsp/sheet/misc-revision_q.pdf

Using pencil-and-paper, each asks you to solve a problem relating to Boolean algebra. There are too many for the lab. session(s) alone, but, in the longer term, the idea is simple: attempt to answer the questions, applying theory covered in the lecture(s) to do so, as a means of revising and thereby *ensuring* you understand the material.

§3. A-class, or additional questions

▷ **Q4[A].** Within the lecture slot(s), the HP-35 calculator was used as an example to demonstrate of two points: 1) the application of techniques and components within a real-world (versus "toy") challenge, and 2) the conceptual relationship between this (or any) calculator and a micro-processor. This question has two options:

- you *could* attempt to design and implement a LogisimEvo-based HP-35 calculator yourself, *or*
- you *could* inspect and use the LogisimEvo-based HP-35 calculator implementation provided.

Note that the former option will take *much* longer to complete, and that you cannot assume enough information provided by the lecture slot(s) alone (meaning research of your own *will* be required). For either option, the overarching goal is to offer insight and understanding: doing so is difficult using an on-paper, theoretical presentation, but arguably easier via hands-on, practical exploration. More concretely, one idea is to evaluate an expression such as

$$(19 - 5) \cdot (1 + 2),$$

and confirm, on a component-by-component, step-by-step basis, that you understand how the implementation is doing so.

A Frequently Asked Questions (FAQs)

I'm confused by the recommended use of built-in components: what do you mean? First, *why*. The recommendation is simply to limit dependencies between lab. worksheets, plus the volume of work required: it is possible to rely on user-defined components instead, but doing so will probably complicates your solution in the sense you will naturally focus less on the central goal(s). Second, *which* and *how*:

- The built-in `Memory`→`Register` component can be useful as a way to store n-bit values. However, it is important to take care re. at least 3 points. First, the value of n is controlled by the data bits property. Second, note that this component is based on flip-flops, and so will be edge-triggered by default; changing the trigger property allows it to alternatively support, e.g., level-triggered latch. Third, this component has inputs and outputs which need explanation:
  - on the left-hand edge, there is a clock input: this represents what we referred to as enable or en,
  - on the left-hand edge, there is a Write Enable (WE) input: this should set (or simply fixed) to 1, otherwise the en input will be ignored,
  - on the bottom edge, there is a clear input: this allows the stored value to be cleared, or reset; this *can* be useful, but, equally, can be ignored if not.
- The built-in `Plexers`→`Multiplexer` component can be parameterised to select between m different n-bit operands: the data bits property directly controls n, whereas the select bits property indirectly controls m (in the sense it directly controls $\log_2 m$, which is the number of control signals required).
- The built-in `Arithmetic`→`Comparator` component can be useful as a way to compare two n-bit operands, e.g., to signal whether one is greater-than, equal-to, or less-than the other. However, keep in mind that it will interpret the operands as signed integers represented using two's-complement by default: the numeric type property allows this to be changed, e.g., to support unsigned comparison instead.
- The built-in `Input/Output`→`Button` component can be useful as a way to model, e.g., reset or rst: it acts in a similar way to an input pin, but is automatically "unpressed" after being "pressed" using the poke tool.