- 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 lab. 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 #8: Finite State Machines (FSMs)

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_2025_TB-4/csdsp/sheet/lab-08_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-08_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-08_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-08_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].** This question is a (or perhaps *the*) textbook example of FSM design: the scenario focuses on two sets of UK-style, i.e., "red, amber, and green", traffic lights:

- the traffic lights are at the intersection is between a main road and an access road,
- they should stop cars crashing into each other, displaying
  - green on main road and red on access road, then
  - amber on main road and red on access road, then
  - red on main road and amber on access road, then
  - red on main road and green on access road, then
  - red on main road and amber on access road, then
  - amber on main road and red on access road,

  and then cycle, and
- they should allow use of an emergency stop button, which
  - forces red on both main and access roads while pushed, then
  - reset the system into an initial start state when released.

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:

- enumerate the inputs and outputs from the controller, and the states it can be in,
- design an FSM (e.g., diagrammatically) that describes all valid transitions between states,
- translate the transition and output functions δ and ω into sets of Boolean expressions, then finally
- fill in the generic framework outlined in the lecture slot(s), and thereby implement and simulate your design in LogisimEvo.

§2. R-class, or revision questions

▷ **Q2[R].** There is a (large) set of questions available at

https://assets.phoo.org/COMS10015_2025_TB-4/csdsp/sheet/misc-revision_q.pdf

There are far too many for the lab. slot(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. Note that the set of questions is general-purpose, in the sense it covers *all* topics covered by the lecture(s): you will need to be selective, and only consider questions related to this lab. slot.

§3. A-class, or additional questions

▷ **Q3[A].** In terms of the functionality involved, the traffic light scenario above could be described as somewhat basic; as a example for learning from, it does not consider various real-world extensions and design constraints. Select a such an extension from the list below (or develop alternatives yourself), then improve your design and implementation to cater for it.

a. If temporary traffic lights are erected to support maintenance work, their timing is crucial wrt. traffic flow. Some roadworks in one lane often dictate use of traffic lights to control shared access to the other lane; their timing (e.g., the period a light is green) should clearly relate to the length of roadworks (e.g., a user-controlled value n).

b. Some advanced traffic light controllers support somewhat dynamic, rather than purely statically defined, behaviour. Imagine a car were to approach the main road with no car waiting on the access road, for example; to prevent the car waiting, the controller may detect such a situation (e.g., via a camera) and skip straight to green on the main road and red on the access road.

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.