COMS12200 Labs

Week 9: The Build-a-comp boards

Simon Hollis (simon@cs.bris.ac.uk) Document Version 2.0

Build-a-comp Lab format

This worksheet is intended to be attempted in the corresponding lab. session, where there will be help available and we can give feedback on your work.

- The work represented by the lab worksheets will accumulate to form a portfolio: whether complete or not, archive everything you do via https://wwwa.fen.bris.ac.uk/COMS12200/ since it will partly form the basis for assessment.
- You are not necessarily expected to finish all work within the lab. itself. However, the emphasis is on you to catch up with anything left unfinished. To accommodate the number of students registered on the unit, the 3 hour lab session is split into two 1.5 hour halves. You should only attend one half, making a selection based on the following:
    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 **ON SNOWY**

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

For example 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.

In build-a-comp labs, you will **work in pairs** to complete the worksheets for the first part of the course. In later labs, with more complex assignments, you will work in bigger groups.

Lab overview

This week's lab is mostly about familiarising ourselves with the new Build-a-comp boards and their simulator, Modulesim.

We'll look at the following components:

1. Input switches
2. Register (Latch)
3. Logic unit

The modules' behaviour will be described towards the end of this document, but for now let's look at the overall structure of the modules and our tasks for today.

1. The first task is to familiarise yourselves with the boards, how they are connected, how they are powered, what inputs and outputs mean and how they are represented.
2. The second task is to use a small number of modules to build your first functioning system. This system is a functional logic block.
3. The third task is to use the register block to provide inputs and outputs to the logical block.
4. Finally, we will see how the *Modulesim* software can be used to perform a software simulation of the modules, allowing you to experiment with circuits outside labs and to create more complex circuits than you may have the parts for.

High-level module overview

The Build-a-comp system is made up of many different modules, each with a different function. The modules can be connected together in an almost unlimited number of ways to build different functionalities. We will start small, but by the end of the course we will use them to build an entire computer.

Data inputs and outputs

Each module has data inputs and outputs; normally the data enters at the bottom edge of the module and leaves at the top edge. **Each data input and output carries 4 bits of data** and every data output has 4 **red LEDs** to indicate the state of the data. The least significant bit is in the right-most position, in accordance with normal binary display. The data signals are labelled `d0..d3`.

Control inputs and outputs

Each module has control inputs and outputs; these enter on the right hand edge and leave on the left-hand edge. This means that the 4-bit

modules can be connected together to perform operations on 8-bit, 12-bit or 16-bit data (we'll do this in later labs, but for today we'll concentrate on 4-bit data). The control signals are labelled `c0..c3`. Control signals are show using **blue LEDs**.

Default input and output values

Unless specified otherwise, unconnected data and control inputs **default to logic 0 (low)**. To change the input values, we need to connect some active module outputs to module inputs.

Power supply

The build-a-comp modules are designed in a way that means that, provided a single module is connected to a power supply, all the interconnected modules will be powered. In practice, this means that you need only connect one power connector to ensure all your modules are operational.

Power supply inputs are located on the **Input** modules (we'll see today) and the **Clock** modules (we'll see in another lab). So, to have a working Build-a-comp system, you will always need at least one Input or Clock module connected to your system.

Interconnection wires

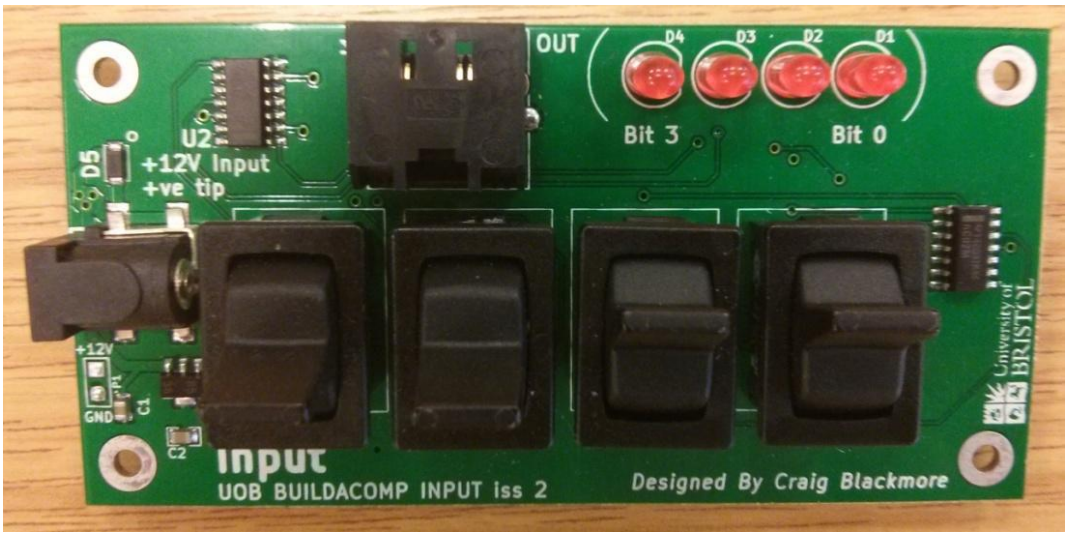
The Build-a-comp system uses standard RJ45 connectors and Category 3/5/5e/6 cables to carry data and control signals. You may know these already as the connectors used for the Ethernet networking protocol. However, the modules **ARE NOT COMPATIBLE WITH ETHERNET** connections, and connecting a module to e.g. your laptop may fry the Ethernet device or v.v. Therefore, **DO NOT CONNECT the modules to an Ethernet device, lest sparks fly!**

Task 1: Board familiarisation

Task 1.1: The Input module and power

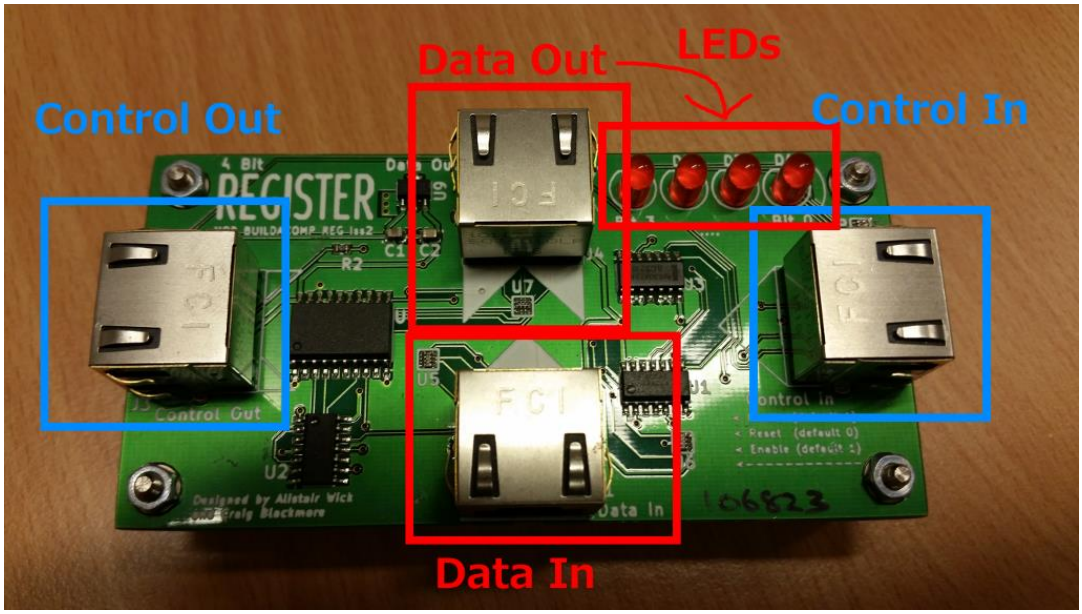
Take an input board from your kit and connect it to a power supply.

Play with the input switches and notice how changing them affects the Red LEDs, which visualise the output value.



Task 1.2: The Register Module

Next, take a register module from your kit. Look at it and try to identify the four connections, as shown in the picture below. Notice how the Red LEDs display the Data Out signal.



Task 2: Connecting them together

In this task, we'll get the simplest possible system working: an Input module connected to a register module, configured as an Output module.

This task involves the following steps:

1. Locate an Input and Register module from your kit.
2. Connect the Input's `Data out` to the `Data In` of the Register Module using a (short) Cat 5 cable. (leave the other connectors disconnected).
3. Connecting the Input module to the power supply.
4. Operate the switches on the Input module to see how they affect the input value being applied (LEDs) and the output from the Register module (output LEDs).

What do you notice? We will explore this behaviour in more detail in subsequent tasks.

Task 3: A logic processing system

Task 3.1

Now, we aim to perform some logic on the data values you can create. We'll use a Logic module to do this. It supports NOT, AND, OR and XOR and performs these binary operations separately on each of its four data inputs. i.e. it can perform four ANDs in the following way:

- Output `d0` = Input A.`d0` `AND` Input B.`d0`
- Output `d1` = Input A.`d1` `AND` Input B.`d1`
- Output `d2` = Input A.`d2` `AND` Input B.`d2`
- Output `d3` = Input A.`d3` `AND` Input B.`d3`

When using the `NOT` function, the Logic module only acts on Input A. Input B values are ignored.

The logical functionality if selected by applying a `control` input to the module. Two of the four possible signals are used, `c0` and `c1`. The following encoding is used:

- `c0 = 0 ; c1 = 0 => NOT`
- `c0 = 1 ; c1 = 0 => AND`
- `c0 = 0 ; c1 = 1 => OR`
- `c0 = 1 ; c1 = 1 => XOR`

This can be expressed more neatly in a table:

Control inputs (binary)	Function selected
xx00	NOT
xx01	AND
xx10	OR
xx11	XOR

Task procedure

1. Locate three Input modules and a Logic module.
2. Connect an Input to each of the Logic module's data inputs and on to its Control input.
3. Operate the switches to a) select a logical function; b) input some data values to be operated on. Observe the output of the Logic module and ensure that it corresponds to what you expected.
4. Experiment with all four of the logical functions.

Task 3.2

Now that we are capable of performing basic computation, let's learn how to store it!

In this part, we'll learn to capture the outputs from our logic block using a Register, to keep it safe from further changes.

In Task 1, we used the register as a module which outputs the same as its input. This is its default behaviour; however the register is made from 4x D-type latches which have ***enable*** inputs (`c2`) to determine whether or not the latches are *transparent* or *opaque*.

The latches also have a clock input (`c0`), which is also capable of gating the output of the latches and a reset input (`c1`), which zeroes the outputs.

Here's the truth table for the **`control in`** signals:

Clock (`c0`)	Reset (`c1`)	Enable (`c2`)	Output
0	0	0	<no change>
0	0	1	<no change>
0	1	0	0000
0	1	1	0000
1	0	0	<no change>
1	0	1	Output = Input
1	1	0	0000
1	1	1	0000

For this lab, I'd like you to fix the Clock input at `'1'` and experiment only with the `Enable` and `Reset` inputs. We will see the role of the Clock input in later labs.

Task procedure

1. Take the system from Task 2.
2. Add a Register module onto the output of the logic module.
3. Add an Input to the Control input of the Register.
4. Experiment with controlling the Register to capture the results of a logical computation. You may wish to try the following.
    a. Enable the Register (set to transparent)
    b. Configure the Logic module to perform a computation
    c. Set up the data inputs and see the result propagate from the Logic module to the Register
    d. Disable the register (opaque)
    e. Re-configure the data inputs to get a new result. Observe that the result appears at the Logic output, but not the Register output.
    f. Re-enable the Register and see the new value propagate to the Register.

After step (e), you will notice that your system is currently storing two separate results, one in the Logic output and one in the Register. This shows how the Register can be used to keep old results around. In later labs, we'll use this mechanism to provide intermediate results for re-computation.

Task 4: Simulation using Modulesim

This task helps you learn about the software simulator that has been developed to allow you to experiment with Build-a-comp modules both inside and outside labs, without need to have physical access to the module kit.

Using the simulator will allow you to work on the various worksheets before, during and after labs and allow you to maximise your time in labs by attempting problems in your own time before

Getting ModuleSim

The simulator was developed by Alistair Wick and is called Modulesim. It is available as a Java jar online via the unit webpage.

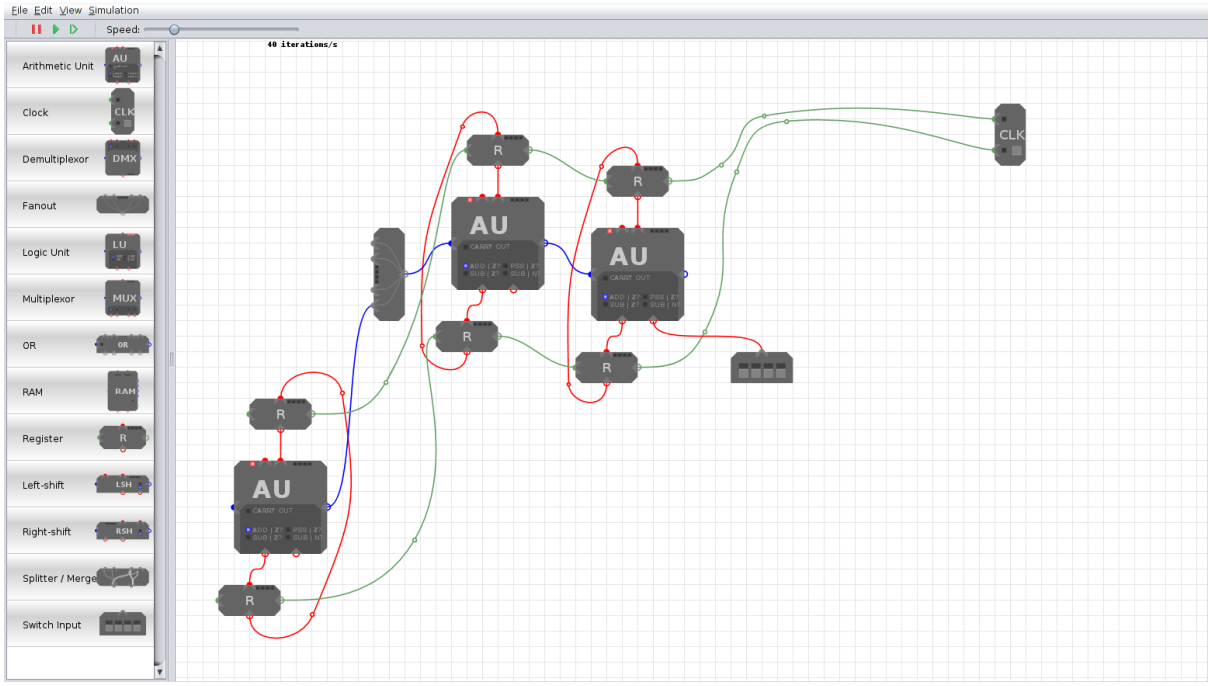
To download it, click the link from this lab on the website and download the file "`ModuleSim.jar`" into a folder on your local linux machine. Then `cd` into the download directory run the simulator with the following command:

```
java -jar ModuleSim.jar
```

Using ModuleSim

Using Modulesim is easy: just drag modules from the left-hand pane to the main window and connect them up with wires. The module layouts mirror the physical ones, with data inputs at the bottom, data outputs at the top, control inputs at the right and control outputs at the left. The LEDs can also be seen to light as they operate and Input module switches can be set by clicking on them.

The Modulesim interface also contains simulation speed controls and a pause button, so that you can explore by setting your preferred time-scale. Designs can be saved and restored.



Task procedure

1. Download and run Modulesim.
2. Create and simulate the circuits from Tasks 2, 3.1 & 3.2.
3. Check that their behaviour matches that of the real hardware.
4. Save the designs and upload them to your portfolio.

Congratulations! You have completed the first Build-a-comp lab!