COMS12200 Labs

Week 10: Build-a-comp arithmetic

Simon Hollis (simon@cs.bris.ac.uk) Document Version 1.0

Build-a-comp module 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.

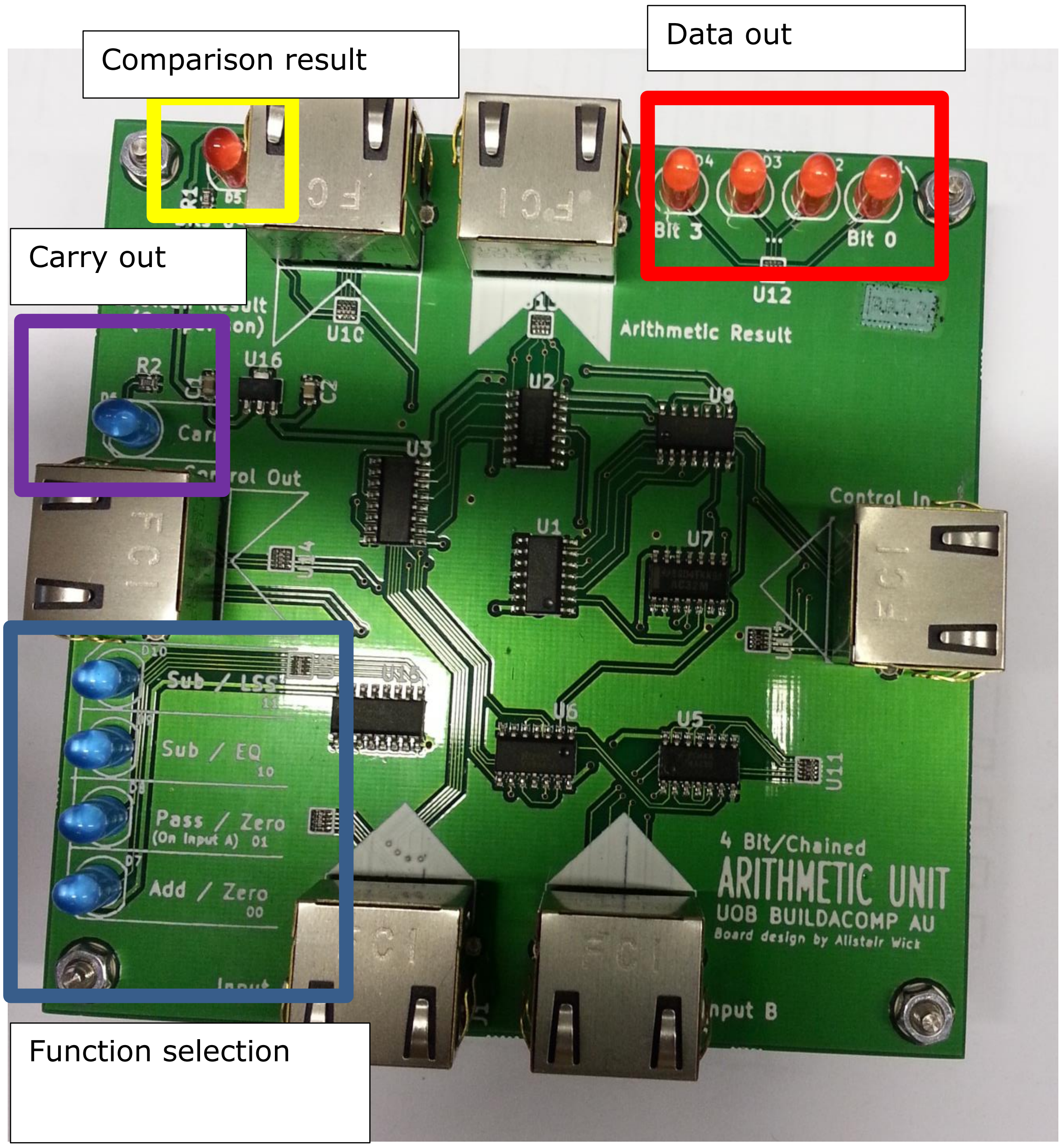
- The work represented by the lab worksheets will accumulate to form a portfolio: whether complete or not, archive everything you do via https://wwwa.fen.bris.ac.uk/COMS12200/ since it will partly form the basis for assessment.
- You are not necessarily expected to finish all work within the lab. itself. However, the emphasis is on you to catch up with anything left unfinished.
- 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.
- 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 session**, 9am-10:30am OR 10:30am-12pm. **The slot to which you have been allocated is visible on your SAFE progress page for COMS12200.**

Lab overview

In today's lab, we will concentrate on the ALU module. The ALU allows us to perform computations on data values and create results. We'll take the input values initially from input modules, then move on to storing them in registers. We'll also do the same for results and create a system with feedback, based on the looping counter you saw in the Week 7 lectures from Dan's part of the course.

Today’s tasks

Today, we will explore the Arithmetic module (ALU) and its arithmetic functions. Here is a brief guide to its most important features:



ALU module overview

Arithmetic

The arithmetic unit takes in two 4-bit inputs and outputs, *expressed in 2's complement form,* and performs one of the following operations on them:

1. Addition
2. Subtraction
3. Nothing (Input A [left] is passed through to the output)

The result is produced on the data out connector and is displayed on the red LEDs.

Comparisons

At the same time as performing the arithmetic, the module also performs a comparison test on its inputs or output. The comparison tests allow a user to determine the relative values of the inputs and output and are very useful in feeding into a conditional logic stage.

The result from comparisons is propagated onto the "Boolean Result (Comparison)" output. Either all bits will be zero (negative comparison), or all bits will be one (positive comparison).

Control signals

The ALU module responds to control signals as follows:

c2	*c1*	*c0*	**Operation**	**Comparison**
0	0	0	Addition (Input A + Input B)	Output = 0?
0	0	1	Pass through Input A	Output = 0?
1	1	0	Subtraction (Input A – Input B)	Input A = Input B?
1	1	1	Subtraction (Input A – Input B)	Input A < Input B?

Task 1

First, we will explore the arithmetic function of the ALU.

Task procedure

1. Recall 2’s complement form for expressing numbers.
2. Attach Input modules to both of the data inputs and the control input.
3. Try varying combinations of data input for each of the arithmetic functions. Use both positive and negative 4-bit numbers and verify that the data output corresponds to what you expect. If anything is unclear, consult a demonstrator.

Task 2

Now, we will explore the comparison function.

Task procedure

1. Keep the Input modules attached to the data and control inputs and add a Register module to the comparison output. Do not connect its control input, so that it remains transparent (enabled).
2. Explore the comparison function by applying data values for each comparison function and verifying that you understand the result of each comparison.
3. You will see that the comparison function lights either none or all of the register module LEDs. This corresponds to the fact that the comparison output is propagated on all four bits.

Carry in and carry out

You are familiar with the concept of a *carry* signal, when applied to adders. In our modules, the carry signal can be thought of as either an overflow signal, representing the fact that the result could not be stored in four bits, or as a representation for 2^5 (16), and a temporary '5th bit', created by the addition of two 4-bit values.

i.e. `carry out = (Arithmetic Result > 15?)`

The ALU module creates a carry output on all operations. The meaning is quite clear for addition, and it is always `0` when in pass-through mode, but it is even more interesting when the unit is in subtraction mode.

Task 3

In this task, we'll explore the carry signal and its meaning with relationship to addition.

Carry with addition procedure

1. Set the ALU to Add by setting `c0`, `c1` and `c2` inputs to `0`.
2. Input two numbers with a total less than 16. Observe that the carry out light is not lit (`carry out = 0`).
3. Input two numbers with a total greater than 16. Observe that the carry out light is lit (`carry out = 1`).
4. Set the two inputs so that they total 15 ***exactly***.
5. Set the `c2` input to `1`. Observe what happens to the `Arithmetic Result` and the `Carry out` signals. Why do you think that this has occurred?

At this point, you should realise that `c2` is in fact not a normal control input signal, as suggested in the previous table, but rather a `carry in` signal. Its value is added to that of the two inputs and the result of the three signals produces the ALU result.

i.e. `Arithmetic Result = Input A + Input B + carry in`

Carry with subtraction procedure

Now, we'll explore what happens with the carry during subtraction (haven't you always wondered what goes on there?!)

1. Set the ALU control inputs as follows: `c0 = 0, c1 = 1, c2 = 0`. This selects subtraction, with `carry in = 0`.
2. Set the two input values to `0`.
3. Observe the output. Can you explain it?

What you should see is that all output LEDs are lit. Recalling that we are working in 2's complement form, this means that the output from the ALU is -1. But, why?

See if you can explain the above by stopping here and thinking about it. When you are ready, turn to the next page for an explanation.

Off-by-one

We saw that the ALU, when set as above performs the following sum:

`0 - 0 = -1`

Why?

The answer lies in the 2's complement form and how subtraction operates.

The ALU performs subtraction in the following manner:

`Result = Input A + (NOT Input B)`

i.e. it first inverts the value to be subtracted, then adds it to the original value. This leads to a very simple hardware implementation of a subtraction unit.

However, there is a problem: the inversion leaves Input B in 1's complement form, and our arithmetic operates on 2's complement form, so the output generates an error.

The solution is to convert to 2's normal form, which we know we can do by simply adding one to 1's normal form following the inversion.

i.e. we need: `Result = Input A + ( (NOT Input B) + 1)`

In our system, the carry in gives us this extra `1`, and so needs to be asserted to make subtraction work.

2's complement subtraction procedure

1. Perform the same configuration as before, but set the `carry in (c2)` bit to `1`.
2. Observe the result. Is it as you originally expected now?
3. Remember that the ALU modules need `carry in = 1` on the right-hand control input if subtracting.

Carry chaining

Finally today, we will look at chaining ALUs via the carry signals.

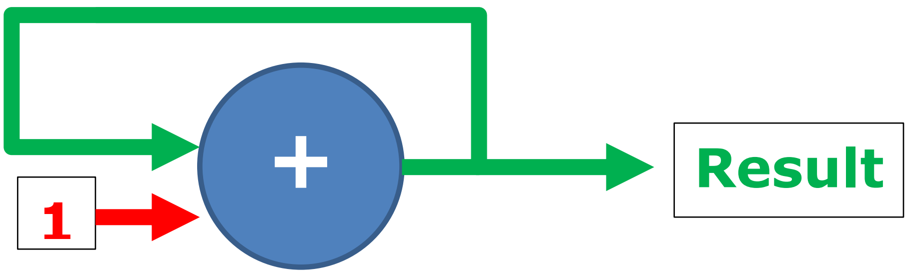
Our aim is to make larger adders and subtraction units than the small 4-bit adder that we have as a base module. The procedure is simple:

1. Take ***two*** ALU units, place them side-by-side and connect Input modules to all four data inputs.
2. Connect an Input module to the right-hand control input.
3. Chain the control ports together by connecting the carry out of the right-hand ALU to the control input of the left-hand one. This will cause the function selection to be sent to both ALUs and the carry out of the right-hand module to propagate as the carry in of the left-hand one.
4. Select an arithmetic function on the control input (`c0` & `c1`)
5. Configure the carry in as necessary for the operation (`c2`).
6. Input two 8-bit data values as 2x 4-bits (e.g. 8-bit Input A is made of the two Input A values, 4-bits of which is input on each ALU board; input B similarly).
7. Observe the 8-bit output by concatenating the left and right-hand Arithmetic results by eye.
8. Verify that the system operates as expected and gives you 8-bit addition and subtraction.
9. If you have any questions, consult a demonstrator.

Task 4: Building a looping counter

Now, it's time to build a useful arithmetic function: a looping counter.

The aim of this task is simple. We wish to produce a circuit that stores a value and repeatedly adds 1 to it. We also want to do this in a manner where the addition is controlled by a control signal.



1. Let's try the obvious way: take an ALU unit, two Input modules and connect them up as follows:
    a. Input module->ALU control input
    b. Input module->ALU Input A. Set value to +1.
    c. ALU Arithmetic Result->ALU Input B (loop with a cable)
2. Use the control inputs to select addition (`0000`)
3. What happens? Can you explain it?

What happened?

The outputs of the ALU unit look like they are all on all of the time, but in fact they are simply flashing on and off very quickly. At this speed, your eye cannot tell the difference and perceives them as always on. If you were to use a high-speed camera though, you would see that they *are* counting numbers one-by-one, just extremely quickly.

This behaviour occurs because the system is free-running i.e. the ALU creates a result and immediately adds +1 to it, creating a new result that also has +1, and so on, forever and as fast as the ALU can add the numbers together.

Controlling the addition

In order to properly control the addition procedure we need to make the behaviour conditional on inputting a control signal. The control signal should allow a single result to be created and captured.

Since we wish to capture some data, the natural component to use is the state-holding register.

One way to interpret this behaviour is that it should be part of the addition loop we have can freeze time.

OK, so let's try this:

1. Take the circuit you had before.
2. Break the loop between ALU output and input and insert a Register in the way: ALU Result->Register Data In; Register Data Out -> ALU Input B.
3. Add another Input to control the Register.
4. Make the Register repeatedly transparent (`Control = x101`) and opaque (`Control = x00x`).
5. What happens? Why? If you are unsure, discuss with a demonstrator before moving on to the next part.

Completing the design

To complete the design, there is one additional element you need to add. See if you can work out what this is, where it goes and add it to the design. Can you reason about why you have had to add this component?

Portfolio addition

Copy the final design of your system and submit it into your portfolio with an explanation of how to use it and how the control and data paths are interacting to provide you with the correct result.

Well done, you have completed Lab 2!