Digital Electronics Practical 2 - Floating Point & Circuit design (week 3)

Previous Next

Solutions will be uploaded at the end of week 5.

Floating Point & Circuit Design questions

Before you start: Check the solutions to Digital Electronics' practical 1 (in the DE Practical 1 folder). If you don't understand anything - ask!

Objectives: The main objectives of this practical are to reinforce and then build on your understanding of binary arithmetic and logic gates. You can practice the techniques introduced in the lectures, as well as practice the use of SimCir, a digital circuit simulator you can use to simulate circuit creation. You can find SimCir here (https://kazuhikoarase.github.io/simcirjs/). As always, it is important that you confirm your understanding by showing your work to a demonstrator.

Exercises

Exercise 1. (Demonstrate your understanding of the Floating Point representation)

Use the following 16 bit floating point representation: SeeeeeMMMMMMMMMM where:

- S is the sign bit and is either 0 for positive or 1 for negative
- eeeee is the biased exponent, storing the value e+15
- MMMMMMMMMM is the normalised mantissa. Remember a normalised mantissa has an implied higher order bit (first to the left of the implied radix point) with a value of 1 that is not stored with the mantissa but is used in arithmetic operations).

E.g. - 1.101*2^3 is stored as S=1, eeeee = 10010, MMMMMMMMMM=1010 0000 00.

Determine the base 10 numbers represented in the following bit patterns:

1. 0000100100000000

2. 1100101110000000

Encode the following base 10 numbers as bit patterns using the 16 bit floating-point representation outlined above. Remember to normalise the mantissa.

3. +2.75

4. -4.375

Find out how many bits are used for each field (sign, exponent and mantissa) is single and double precision floating point representation. What is the largest and smallest representable number in double precision floating point?

Exercise 2. Watch this video about new transistor design (https://www.youtube.com/watch?v=YIkMaQJSyP8&ab_channel=Intel). There are also lots of videos of chip making, e.g. here (https://www.youtube.com/watch?v=35jWSQXku74&ab_channel=RobertoMulargia)(skip to 2:15 to get to the point).

Exercise 3. Watch this video (https://www.youtube.com/watch?v=aHx-XUA6f9g&ab_channel=HACKADAY) which shows some real transistors up close. The whole video is interesting, so do watch if you have time, but the most relevant bit is from 5.55 for about 4 minutes.

Exercise 4. Visit SimCir online (https://kazuhikoarase.github.io/simcirjs/) and play around a bit to get used to the system. Play with the gates and get used to the program and how each gate behaves.

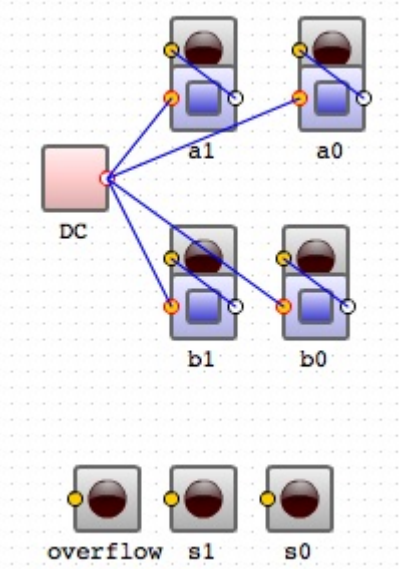
Exercise 5. Create two circuits in SimCir which have:

- A DC 5V power source;
- Two toggle switches representing the inputs A,B;
- An LED representing the output;
- An arrangement of **NOR gates only** such that when the inputs are set, the output LED is lit if and only if:
    - A AND B are ON (1st circuit),
    - A OR B is ON (2nd circuit).

Exercise 6. Create two circuits in SimCir which have:

- A DC 5V power source;
- Two toggle switches representing the inputs A,B;
- An LED representing the output;
- An arrangement of **NAND gates only** such that when the inputs are set, the output LED is lit if and only if:
    - A AND B are ON (1st circuit),
    - A OR B is ON (2nd circuit).

Exercise 7. Create a circuit to add two 2-bit binary numbers. I.e. have four inputs representing a1a0 and b1b0, and three outputs representing s1s0 and c, such that s1s0 gives the addition a1a0+b1b0 and c indicates overflow. (This template shows you the basic setup to start from, but you will have to recreate it yourself.)



Exercise 8. If you find you have time left, how about adding a toggle switch which turns the calculation from part 5 into a subtraction. Recall that the easiest way of dealing with subtraction is to turn the subtrahend to its opposite (negative) and doing the corresponding addition. To turn a number into its negative, recall the twos-complement technique. To see the effects more clearly it would be worth upgrading the whole thing to 4-bit numbers.

Practical extension

Easy:

- Write your own code to take floating point number as a 32-bit binary string as input and output the decimal equivalent.

A bit harder:

- Write your own code to take a decimal number as input and convert it to the closest 32-bit floating point binary number.