# SECTION 1)

## HEADER)

19.10.2025, Imperial College London main campus, DECA

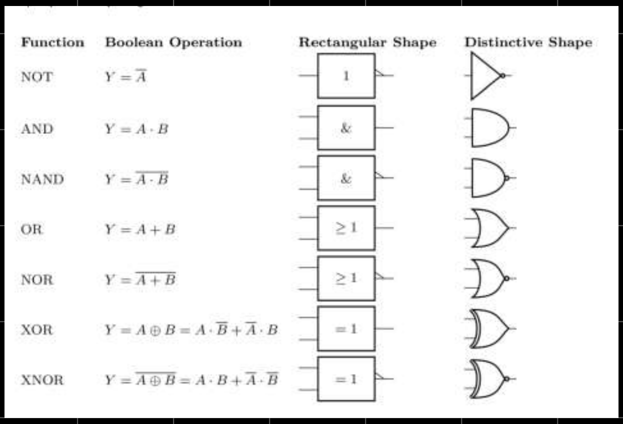
## OBJECTIVES)

Become familiar with Issie software, learn about; logical gates and their different representations (rectangular shape, distinctive shape), multiplexers (2 and 4 inputs with 2 and 4 bits for each input), hierarchical design, busses, building a simple ALU (arithmetic and logic unit)

## BACKGROUND)

Issie is a digital circuit sandbox which allows for building and debugging digital circuits. Logical gates take in Boolean inputs usually denoted with capital letters which take on 1 out of two possible values "0" or "1", low or high etc. They are constructed using CMOS design (complimentary mosfet). They are the basic building blocks of any digital circuits and allow one to implement logic. Basic gates are AND, OR, NOT. Out of those one can build every other one. Buses are essentially grouped parallel wires which carry different data, which later is put together into a single digital value using merged wires or logical gates. Multiplexers serve a very similar role as logical loops in c++( if(s) then a else b ). They have many inputs or arbitrary width and a control input usually denoted as S, which allows to choose which input actually gets passed through the multiplexer. Hierarchical design allows one to use a distinct sheet to build a specific component and then export it out and use it in different parts of the project as a abstract device which only shows inputs and outputs.

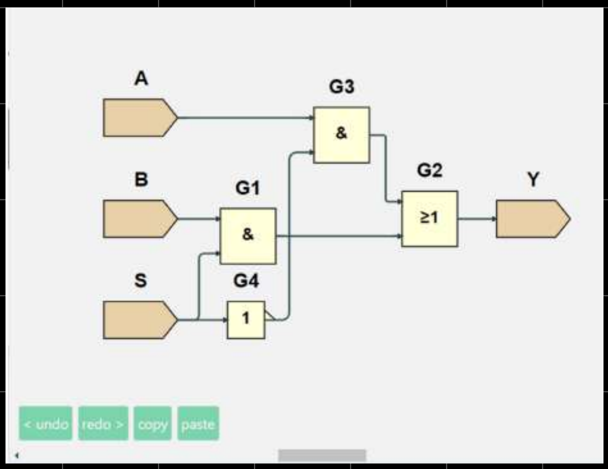
## MATERIALS AND METHODS)



| Function | Boolean Operation | Rectangular Shape | Distinctive Shape |
|---|---|---|---|
| NOT | $Y = \overline{A}$ | 1 | |
| AND | $Y = A \cdot B$ | & | |
| NAND | $Y = \overline{A \cdot B}$ | & | |
| OR | $Y = A + B$ | $\geq 1$ | |
| NOR | $Y = \overline{A + B}$ | $\geq 1$ | |
| XOR | $Y = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$ | =1 | |
| XNOR | $Y = \overline{A \oplus B} = A \cdot B + \overline{A} \cdot \overline{B}$ | =1 | |

Logical gates symbols  (rectangular and distinctive)

Firstly I built a 2x1 input (2 inputs of width 1) multiplexer using AND , OR and inverter gates. The Boolean equation describing it is



$$Y = A \cdot \overline{S} + B \cdot S$$

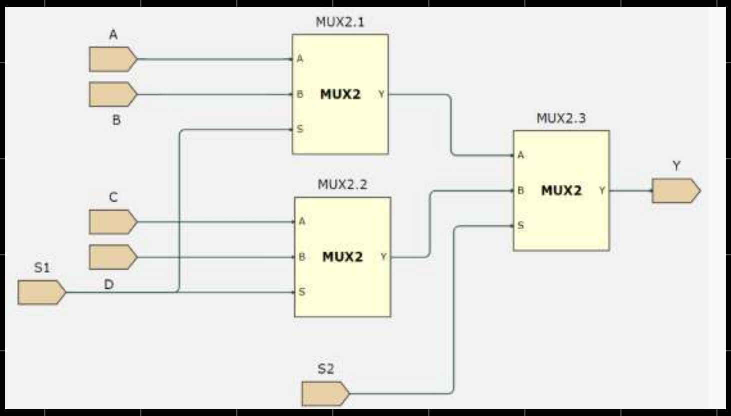
| S | Y |
|---|---|
| 0 | A |
| 1 | B |

2x1( 2 inputs of width 1 ) multiplexer lowest level of abstraction

Which simplifies to Y=B when S=1 and Y=A when S=0. Because of this logic giving s different values allows one to control the output

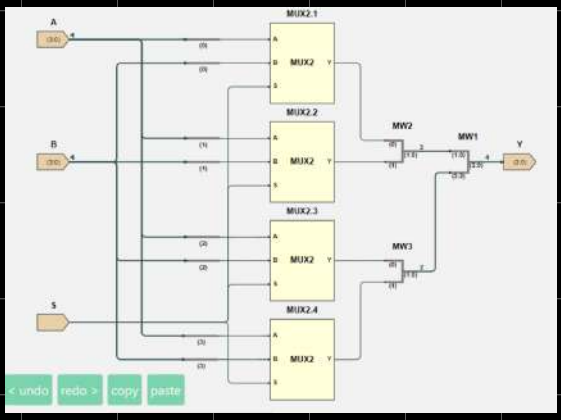
The using hierarchical design I made an abstract 2x1 (2 inputs of width 1) multiplexer. I used it to built a 4x1 (4 inputs of width 1) multiplexer which can be described by the following equation;

$$Y = A\cdot\bar{S}_1\cdot\bar{S}_0 + B\cdot S_1\cdot\bar{S}_0 + C\cdot\bar{S}_1\cdot S_0 + D\cdot S_1 S_0$$

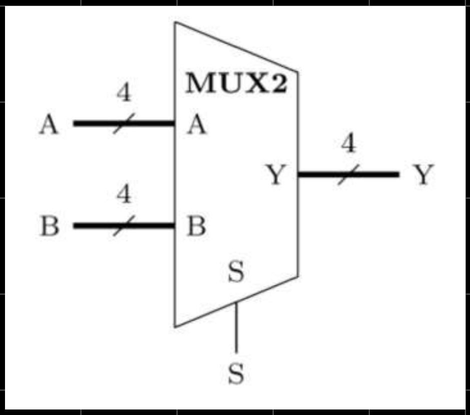


| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |

So again varying values of the 1x2 input (1 input of width 2) allows one to control which input (A,B,C,D) comes out as the output of the multiplexer.



2x4 input (2 inputs of width 4) multiplexer medium level of abstraction



2x4 (2 inputs of width 4) multiplexer highest level of abstraction

## ANALYSIS)

Built elements work in the way theory predicts, which confirms they were constructed properly.

## DISCUSSION)

In checking the validity of the results the simulation tab turned out very useful. It allows not only for debugging construction errors but also for generating truth tables with number or algebraic inputs which often drastically lowers the number of entries and speeds up troubleshooting.

## CONCLUSION)

All goals of the lab have been achieved.

## CHALLENGE)

Building a simple ALU (arithmetic and logic unit) which performs following bitwise operations on 2x4 inputs (2 inputs of width 4)

->NOT A

-> A OR B

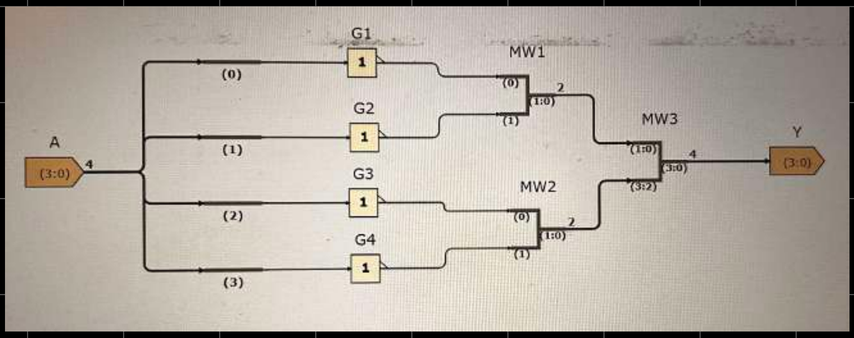
-> A AND B

-> A XOR B

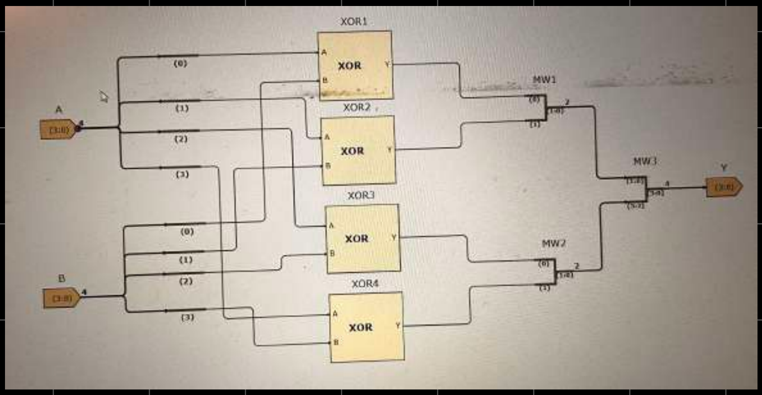
Bitwise operations rely on splitting a m buss of n bits input into m parts and every single one of those is then split into n bits and the each logical operation is performed on parallel bits with the same index and is then a output with the corresponding index

ex( abcd AND efgh = (a AND e, b AND f, c AND g, d AND h) where commas separate two bits of the output)

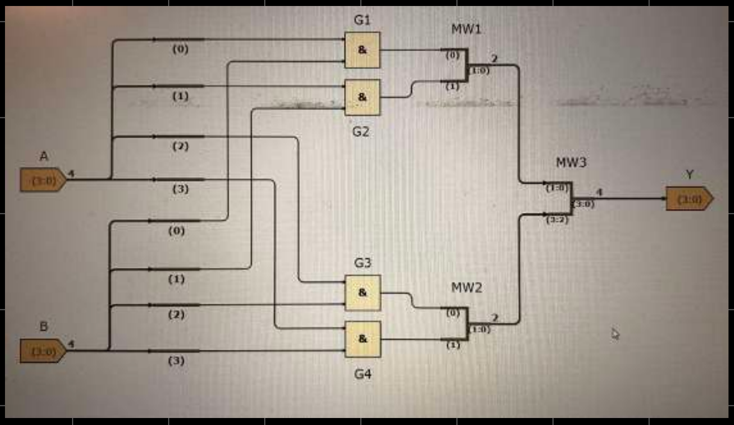
To build the ALU i used a 4x4 input ( 4 inputs of width 4 ) multiplexer earlier made. The only thing left to do was to create 1x4 input (1 input of width 4) not gate , 2x4 (2 inputs of width 4) AND OR XOR gates



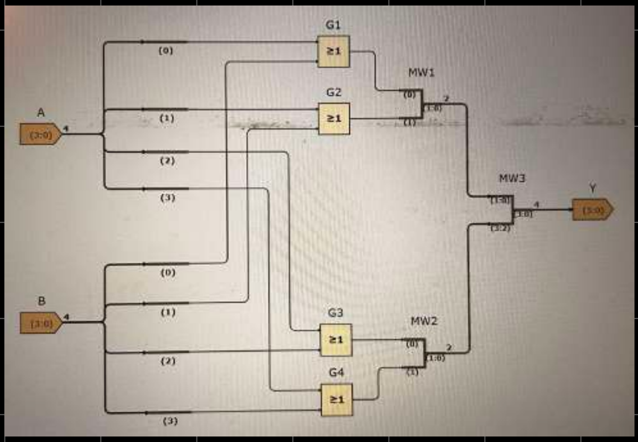
1x4( 1 input of width 4) inverter.



2x4( 2 inputs of width 4) XOR gate

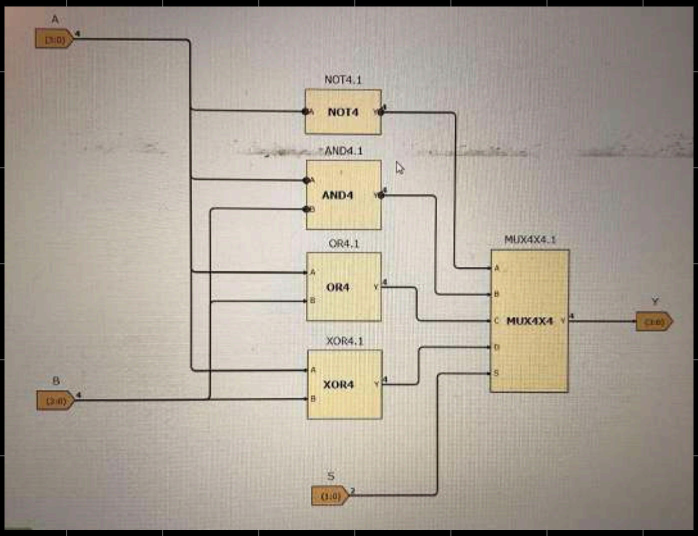


2x4( 2 inputs of width 4) AND gate.



2x4 (2 inputs of width 4) OR gate

Than after connecting then in an appropriate manner I ended up with a basic ALU



## SECTION 2)

### HEADER)

19.10.2025, Imperial College London main campus

### OBJECTIVES)

Build; a half adder, a full adder, a ripple-carry adder, a subtractor

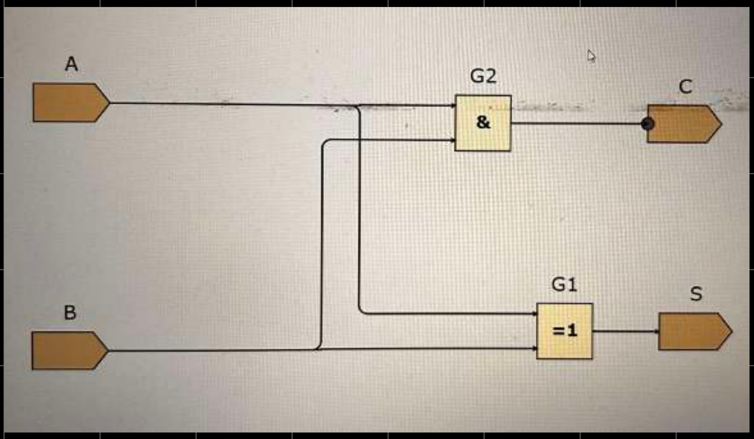
### BACKGROUND)

Half adders are digital circuits which in this case add two one bit numbers and produce a sum and a carry out. Full adders add three one bit numbers together and produce a sum and a carry out as well. Those can be further used to extend the number and width of bits being added in one operation. Furthermore they allow to implement arithmetic operations which are the basis of any advanced system.

### MATERIALS AND METHODS)

A half adder has three inputs A, B and CIN where A and B are added numbers and CIN is a LSB (least significant bit). It has two outputs; S and COUT where S is the LSB of the operation and COUT is the MSB which overflows due to both numbers being to big to fit in a given width

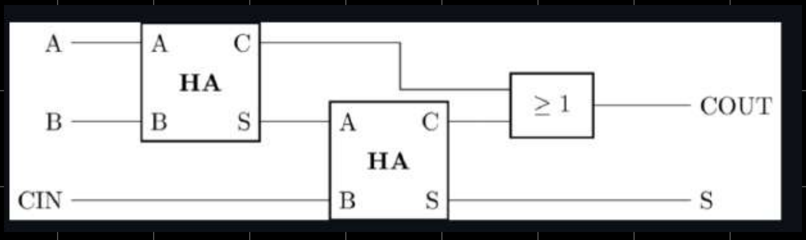
A half adder can be described by this Boolean equation

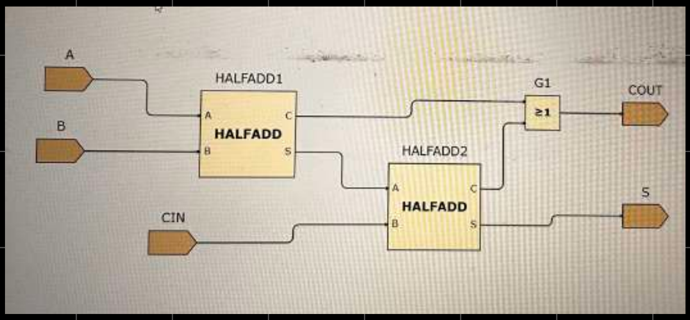


$$S = A \oplus B = A \cdot \bar{B} + \bar{A} \cdot B \quad \wedge \quad C = AB$$

Half adder 2x1

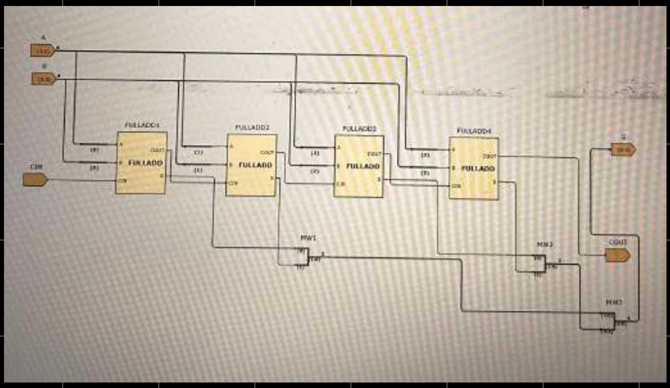
A full adder works by combing in two half adders. Again it has inputs A, B and CIN but this time it allows to add those three numbers together ( addition happens two times, in half adders it only happens once). Firstly it ads A and B using the first half adder. Then it passes the sum as a first input to the second adder, the second input of the second adder is CIN. The sum of the addition of the second adder is the final output sum. Carry outs from both adders are connected via the OR ( because both can never be equal to 1 at the same time due to the construction) gate to the output carry out.

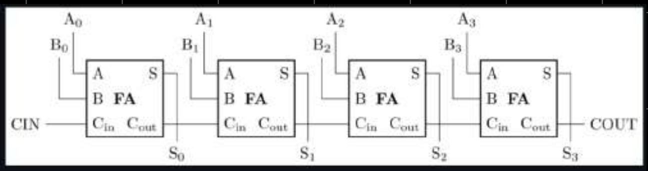




Full adder 3x1

Then I implemented a 4 bit adder. To do so I implemented the ripple-carry adder. It works as follows; 4 previously made full adders are used, each computes the partial sum of the digitis of the same index. Additionally there is a carry in port included to allow for further expansions of the adder to 8, 16 bits and so on. It is connected to the full adder which operates on the LSB then the carry out from that full adder is connected to the second LSB and so on to ensure proper transfer of carry outs (think of adding decimal numbers, carry out is always transported to higher indexes of a number). Then all 4 inputs are grouped together to form 1 number of width 4 which comes with a carry out to ensure all possible additions can be valid.



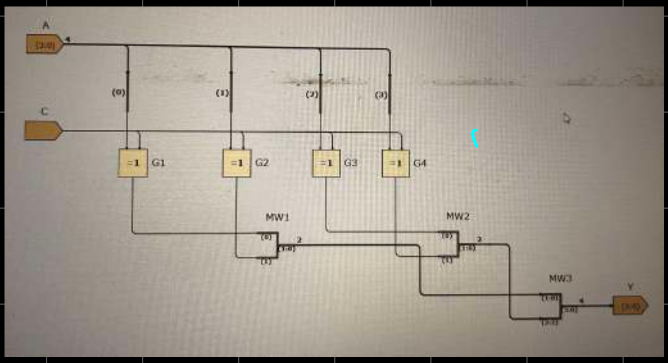


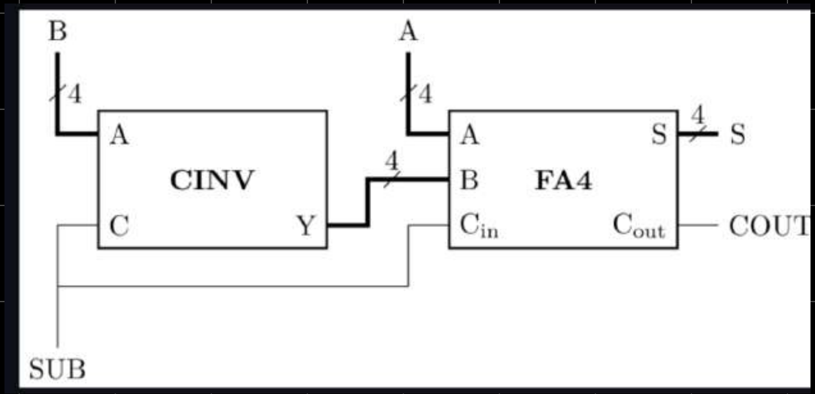
Ripple-carry adder

Then I implemented the ability for the adder to subtract as well. For this purpose I took advantage of the twos complement number.

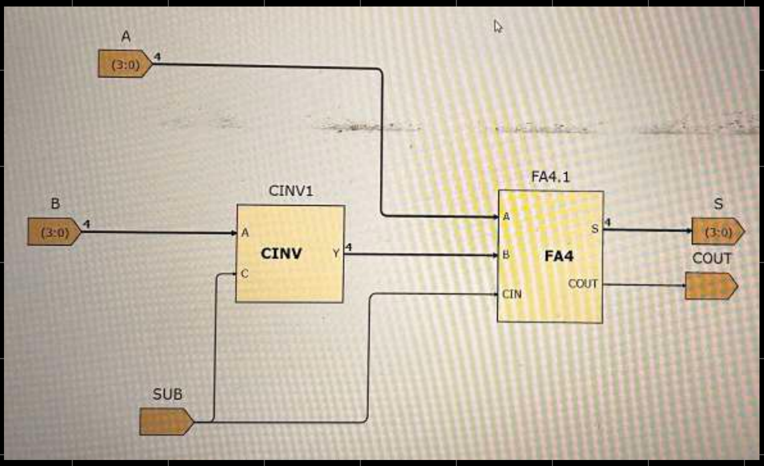
This device takes in a 4 bit number and then performs bitwise inversion. According to the twos complement model you the only need to add 1to the number at hand to receive the negative version of the initial number.

I built a cinv device which transforms any 4 digit binary number to its negative. Firstly I perform bitwise NOT using 4 XOR gates (they allow for more flexibility than inverters, if inputs are A and B when B=0 XOR acts as a buffer but when B=1 XOR acts as an inverter, this allows to control wether we are adding or subtracting numbers) on it and then I add one to the result. Than it is simply enough to add the received number to perform subtraction. Than to complete subtraction  I only need to add one to the inverted number which can be done with the carry in port of the FA4, which is convenient because when SUB =1 we want to subtract. For convenience I included a 2x4 multiplexer which allows to switch between addition and subtraction.





CINV



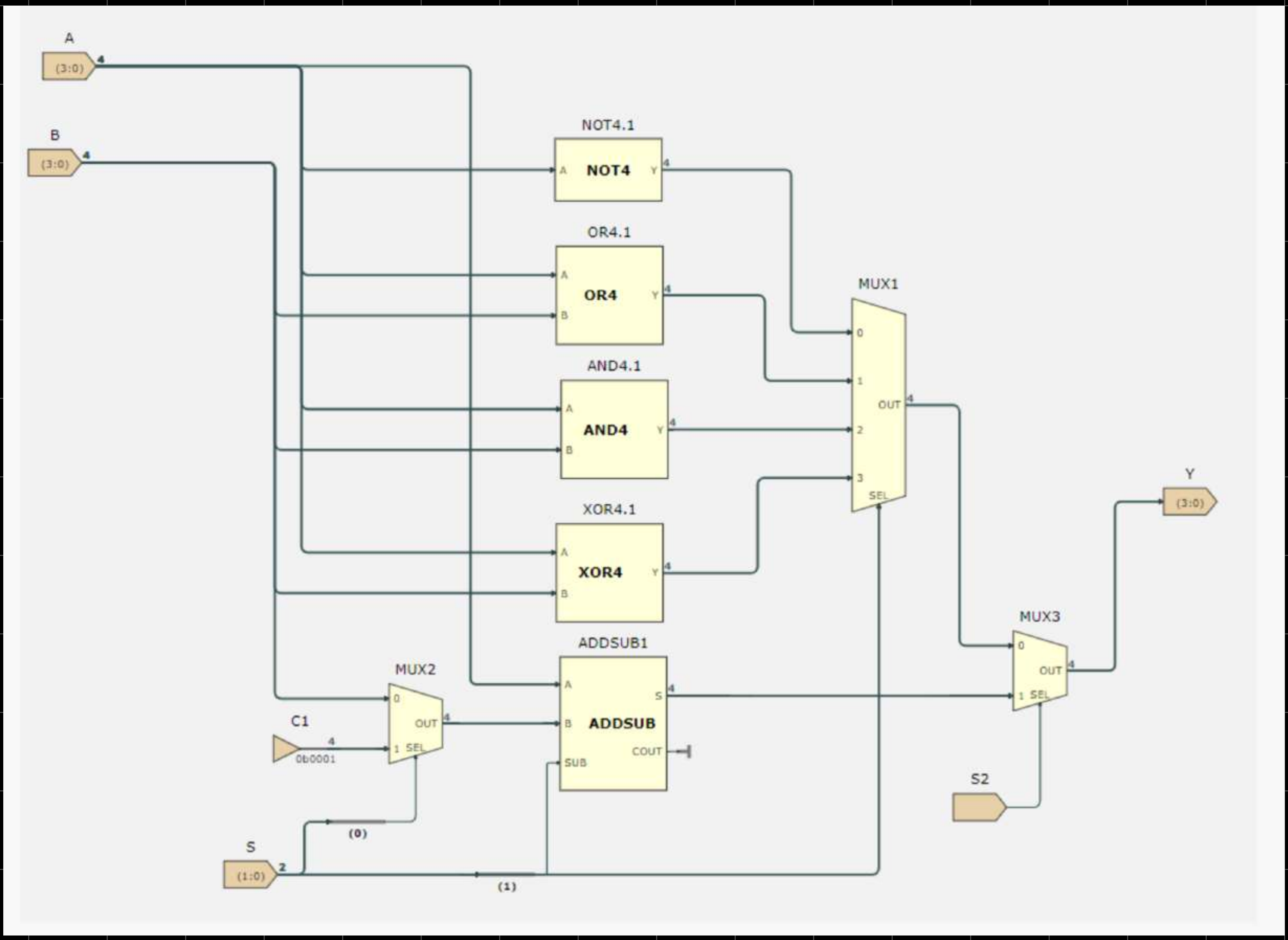
Addsub block

## ANALYSIS)

After extensive testing using the truth table simulations I can verify that all components act as desired and follow the Boolean logic describing them. It its worth stating that when the number of inputs to a function is large ins more convenient to view the algebraic truth table which includes names of literals rather than raw 0s and 1s.

## CHALLENGE)

In the challenge section of this lab part we are asked to add additional functions to the previously made ALU. Now apart from bitwise operations we want it to do arithmetic operations as well, such as incrementation decrementation subtraction and addition. In order to do this I will leverage the earlier built addsub block. For now we leave the previous version of the bitwise operational circuit untouched. The literal which we want to increment/decrement is A so we directly wire it to the addsub block. The we implement a 2x4 (2 inputs of width 4) mux which passes on either B or 0001 in binary when the LSB of S is 0 or 1 respectively. The output of that mux is than passed forward as the second input to the addsub block and based on the MSB of S it is either added2 or subtracted with A. Than to keep the bitwise part of the circuit operating we implement one more mux with a separate input that passes through either bitwise operations or arithmetic ones.



## SECTION 3)

### HEADER)

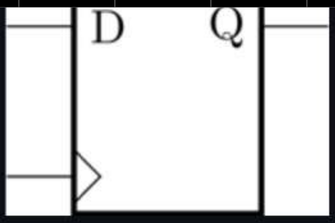
Imperial College London, main campus 25.10.2025

### OBJECTIVES)

Introduce the notion of DFF (D-type flip flop) to build a single 4 bit DFF. Then upon successful wiring of the aforementioned component it will be implemented in a circuit with a 4 bit full adder which uses feedback from the DFF to increment the output which is passed further in the circuit upon a clock tick. Then a synchronous load function is added to enhance the functions of the circuit.

### BACKGROUND)

D-type flip flop is a basic element of sequential circuits (circuits which store data, so at a given point in time the output of a function can’t be determined solely by the current inputs, because memory plays a role in those kind of devices) it works in the following manner. The input D is only passed to the other side of the DFF if a rising clock edge is detected in the clock port. When the value is outputted it remains constant until the next rising edge which can change it or leave it the same. So DDF "remembers" its output.
Exactly because of this property DDFs allow to create feedback loops where the output depends not only on input variables but also on the past.

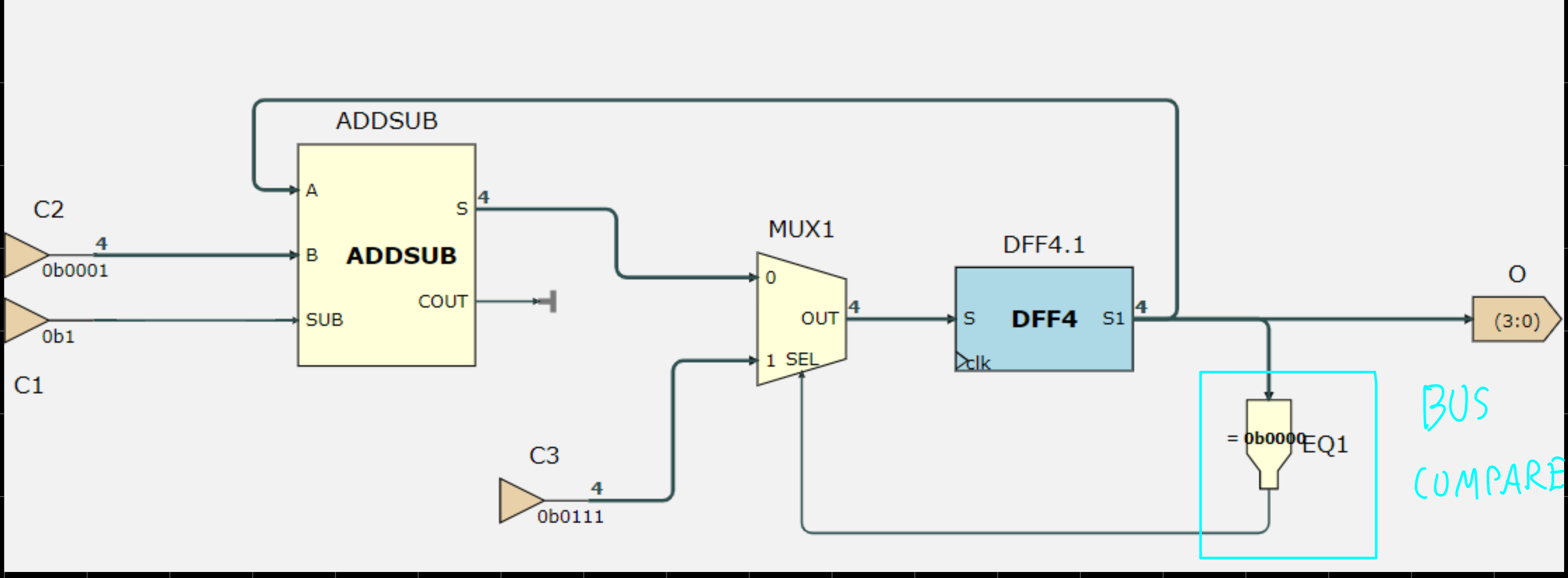


| D | Clock | Q |
|---|---|---|
| X | 0 | NC |
| X | 1 | NC |
| 1 | ≣ | 1 |
| 0 | ≣ | 0 |

*X: Don't care, NC: No Change, ≣: Rising Edge*

DDF

A bus compare is a neat device in Issie which allows one to provide the device with a certain binary number as its "trigger". When the input to this device will be exactly equal to the trigger the device will produce a logical high as its output. Otherwise it produces a logical low.



Bus compare

MATERIALS AND METHODS)

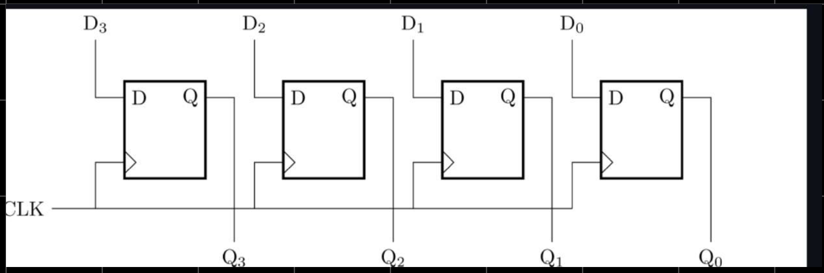
-4 bit full adder

-2x4 multiplexer

-4bit DFF

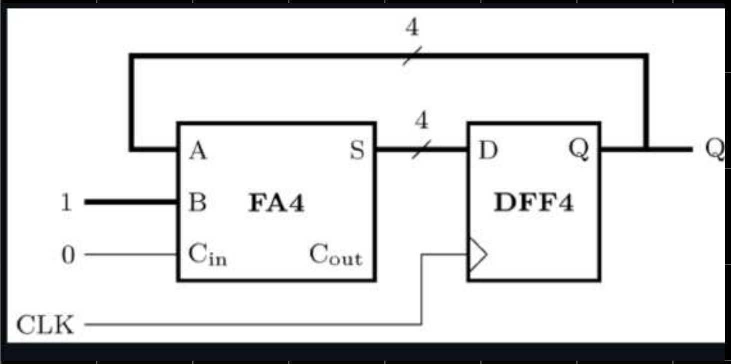
-bus compare

For the purpose of this session we treat the DFF as a abstract device. Since the given DDF only takes in data of width 1 we need to extend it to 4 bits using four distinct DFFs.



It is worth mentioning that in Issie all devices which operate based on a clock are universally connected to it so it's  not necessary to wire them up by hand to the clock in the sheets.

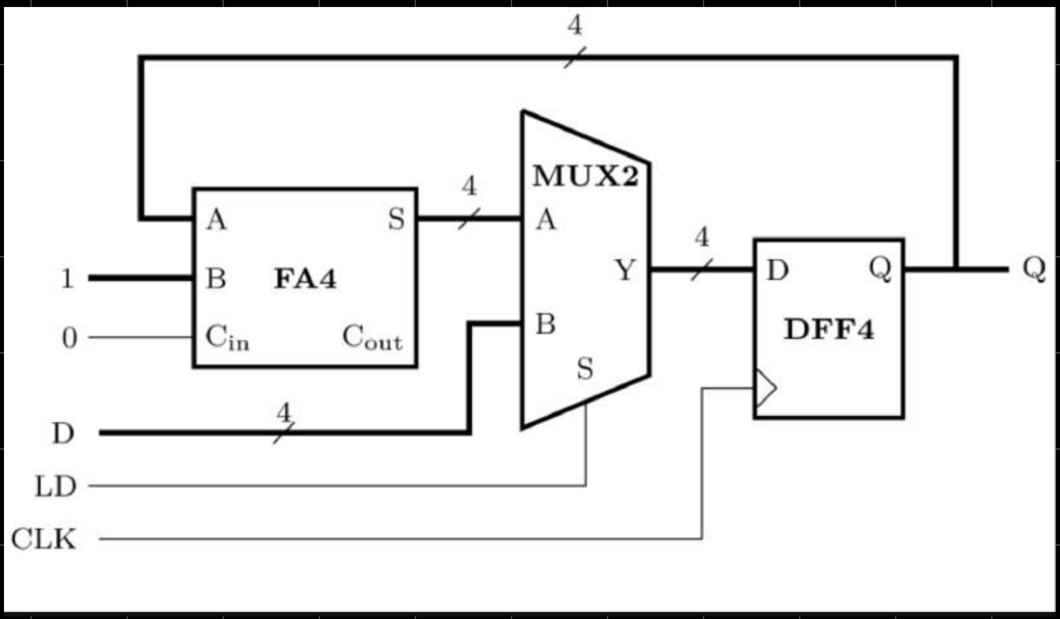
Then this abstract device is connected to the previously made 4 bit full adder in the following manner.



Counter

In the first cycle the input B which is equal to 1 goes through the FA4 and  at that moment A=0 then S=1. This input is passed on to the DFF4 however in order for it to be passed further to Q a rising edge needs to be detected by CLK first. Without the rising edge this circuit is going to stay like this forever. When the clock tick is detected Q=1 and since it's fed back to the FA4 through the A branch on the second loop S=2. With every rising edge this patterns continue which leads to Q being incremented. It is worth noticing that at all times D=1+Q. Since this circuit operates on 4 bits it can easily overflow. Upon that it restarts itself.

Nextly to make this circuit more interesting we want to implement a synchronous load function ( a function witch is synchronized with the clock and loads new data into the circuit). In this case the synchronous load function will act as a refresh button,it will make the counter start counting  from 0.



Counter with a synchronous load function

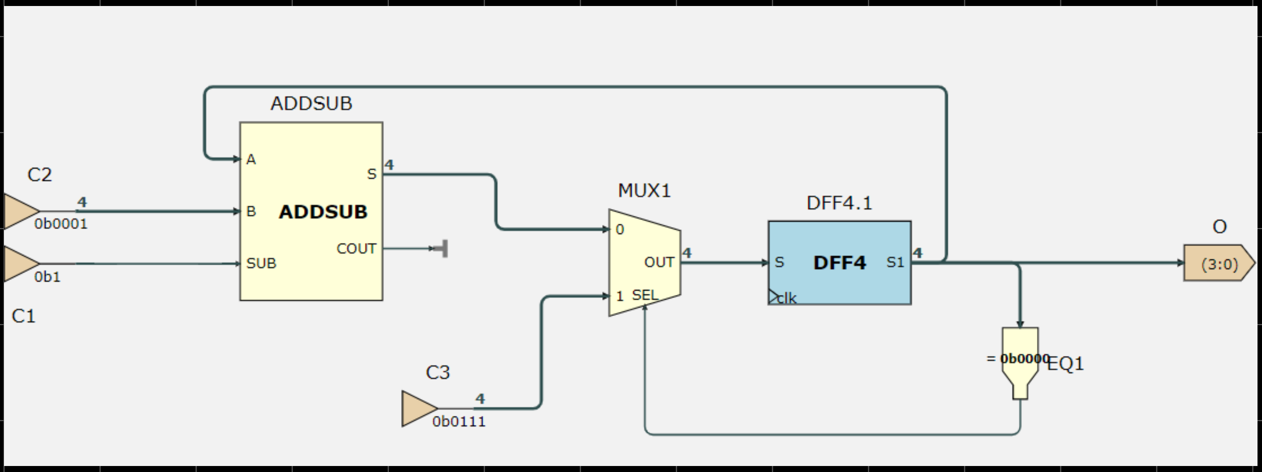
The following circuit work in the following manner. If LD=0 the 2x4 multiplexer will pass on further only the input A. In this case the circuits works exactly like the previous one. It increments Q using the feedback loop formed by the DDF4. However when LD=1 the multiplexer will ignore A and will pass input D through port  B as input to the DFF4. When the next rising edge is detected D will go through DDF4 and Q=B. Because the feedback loop is there B will go as the input A to the FA4 so S=B+1. As long as LD is high the output Q will always be equal to D. Then when LD goes low the multiplexer will pass forward the current value of S which is B+1 which then becomes the input to the DDF and upon the next rising edge is passed though it and Q=B+1. In the case where B=0 it serves as a reset button for the counter.

## Results)

After creating the aforementioned circuits in Issie I tested them using the simulations tool. All of them behaved as expected which shows they have been properly constructed.

## Challenge)

For the challenge I was required to build countdown timer with a auto reset. The idea is that the timer should start from a certain binary number and than with each clock tick it subtracts one from it until it reaches 0 upon which it restarts and the process begins all over indefinitely.



C2 states what value should be subtracted from S1 every iteration

C1 states that the addsub should be in subtraction mode rather than addition.

C3 is the value from which the clock is counting down

EQ1 is a bus compare which resets the circuit to C3 when Q=0

Initially 0= zero so bus compare triggers and causes the multiplexer to pass forward C3 to the DDF4 then when the next rising edge comes this value goes through the DDF4 and O=C3 since 0is not zero bus compare produces a logical low which causes the multiplexer to pass through S coming over from the addsub block. This continues to happen and with each rising edge of the clock 1 is subtracted from the initial value. This happens until 0 is zero again and so the bus compare triggers and restarts the cycle.