BINARY ADDITION

Kira Clements, University of Bristol

WHAT IS 1+1?

Going back to basics of addition, what are the possible results for adding two single-digit bits?

A	B	Decimal Sum
0	0	0
0	1	1
1	0	1
1	1	2

A	B	Binary Sum
0	0	0
0	1	1
1	0	1
1	1	**10**

Each input must either be 0 or 1 while the three possible outputs are 0, 1, and 2 (0b00, 0b01, 0b10).

This gives us the equivalent of 2 single-bit outputs: the Sum (S) and the Carry (C).

HALF ADDER

If we define these two output bits as C (Carry), which represents the decimal value 2^1, and S (Sum), which represents the value 2^0, we get the following truth table:

A	B	C (2^1)	S (2^0)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Same outputs as the truth table for AND

Same outputs as the truth table for XOR

Revisiting logical operators, which will output the same values as S and C?

$C = A \wedge B$, $S = A \oplus B$

Using these logical expression for the Sum and Carry, we can now add 2 single-bit signals together, but how do we extend this to add multiple bits together?

ADDING MORE DIGITS

Decimal and binary addition both use a carry value to determine the value of the next outputs.

We start with adding the right-most digit (LSB or bit number 0 for binary addition) and use the carry from this addition to calculate the outputs immediate to its left (bit number 1 for binary addition).

Decimal addition

A		9	
B	1	3	+
Sum	2	2	
Carry	1		

Binary addition

A		1	0	0	1	
B		1	1	0	1	+
Sum	1	0	1	1	0	
Carry	1	0	0	1		

This is considered an **overflow** as we are currently using 4-bit numbers. Though this is currently ignored, we will revisit!

FULL ADDER

When adding more than single digits, we need to be able to include a **third input in the addition**, *C_in*, which extends our previous half adder truth table:

C_in	A	B	C_out	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The half adder and full adder get their names from the fact that a full adder can (roughly) be created from 2 half adders where the first would calculate A+B and the second would calculate S+C_in.

FULL ADDER FORMULA

C_in	A	B	Sum
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

AB\C_in	0	1
00	0	1
01	1	0
11	0	1
10	1	0

From the Karnaugh map for the Sum output (or straight from its truth table using DNF), we can find the following formula:

$$(C\_in \land \neg A \land \neg B) \lor (\neg C\_in \land \neg A \land B) \lor (C\_in \land A \land B) \lor (\neg C\_in \land A \land \neg B)$$

Though this is the simplest form using just [¬, ∧, ∨], this can also be written as:

$$A \oplus B \oplus C\_in$$

XOR for multiple argument is true when an odd number of its inputs are true

FULL ADDER FORMULA

C_in	A	B	C_out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

AB\C_in	0	1
00	0	0
01	0	1
11	1	1
10	0	1

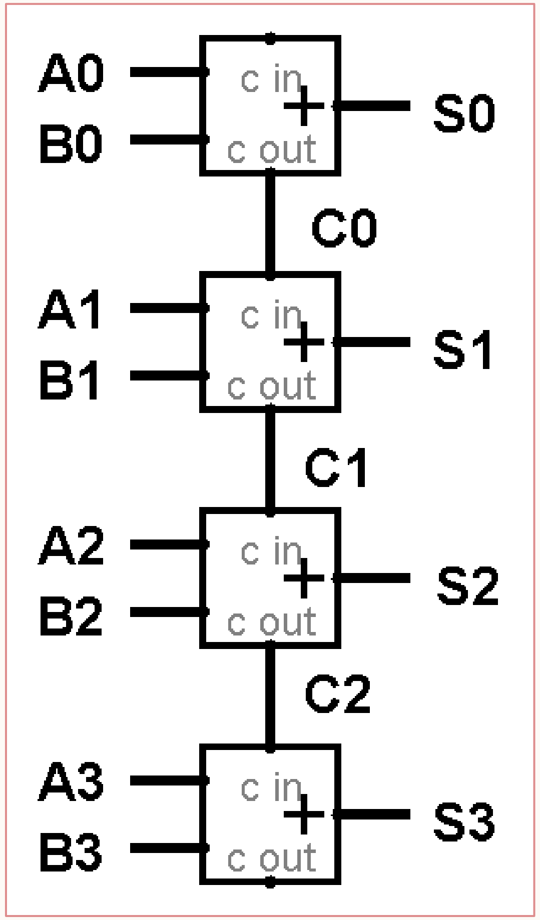
From the Karnaugh map for the Carry out output we can find the following formula:

(C_in ∧ B) ∨ (A ∧ B) ∨ (C_in ∧ A)

Using distributivity, this formula is logically equivalent to (A ∧ B) ∨ (C_in ∧ (A ∨ B)), which is also logically equivalent to **(A ∧ B) ∨ (C_in ∧ (A ⊕ B))**… Why?

Preferred formula in this situation, as A ⊕ B is a signal that can be copied from the Sum circuit

ANOTHER BUILDING BLOCK



Now we've gone from adding 2 bits (A, B) to adding 3 bits (A, B, C_in), we can look at connecting these building blocks to add numbers with more than one bit each!

A	A3	A2	A1	A0	
B	B3	B2	B1	B0	+
Sum	S3	S2	S1	S0	
Carry	C2	C1	C0		

Notice that bit numbering goes right to left, in order that the addition takes place

This is known as a **ripple carry adder**, where the C_out signal of each full adder is the C_in signal of the next full adder. It's named this as the carry signal generated from the LSB can affect the result of any/all of the more significant bits.