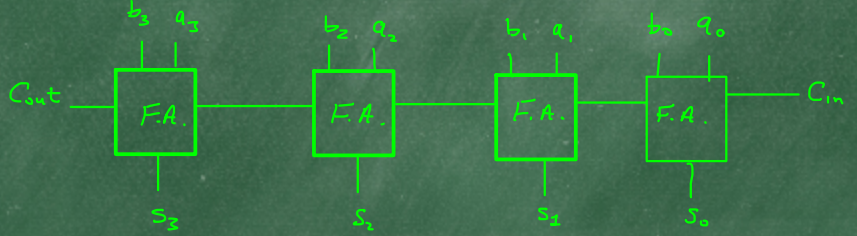
Terminology: Black Box

A Black Box is a device whose outputs depend only on it Inputs.

To use it, you don't need (or indeed want) to know how it works, or what is inside.

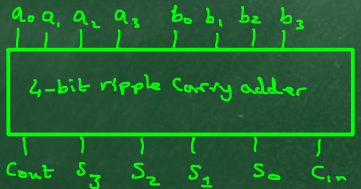
A Black Box abstracts away details and considers only functionality.

We saw this in action when building a multi-bit adder:



In this circuit for adding two 4-bit binary numbers, each F.A. is expressed as a black box

We can go further, and express the 4-bit adder also as a black box:



The story so far

Human: Symbolic Information
- text, Ideas, ..

Sensory Information
- Sound, Colour, Pressure, ...

Machine: Switching Circuits, 2 States: 5v-0v, On – off, 0–1

Base 10 'natural' but 'large' bases also manageable

Base 2 most 'natural'

Modelling Information with Numbers and Interpretation.
- Here this number means this, there it means that

Larger Bases
⇒ few digits needed for given number but more symbols needed.

⟷

Smaller Bases
⇒ more digits needed for given number but fewer symbols needed (i.e. fewer states needed leading to greater simplicity)

← abstract away favouring bases that are a Power of 2 for Simplicity and translation.

The number of numbers we can represent is limited by the physical constraints of our machine.

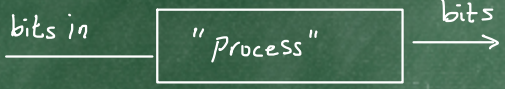
How many numbers can we represent?
— How many bits can we manipulate?

How many numbers do we need for different kinds of data?
— we not only want to use numbers for calculation but also to represent things in collections.

What kinds of interpretations do we apply to our numbers?
— Integers, real numbers, characters, instructions, ...

Representing Information

"Processing" Information



- Arithmetic Manipulation
- "Analyse"
- Animate an Algorithm
- ---

Logical Manipulation
- AND, OR, NOT

Insights of Boole, Shannon, Von Neuman, ---

Implemented by

Logic gates building blocks for

Components to Implement Algebraic Manipulation

Dymanically Create Pathways to Implement "Instructions"

Examples so far:

- Full-ADDER
  - Implementation chosen to Illustrate 'Selection' and motivate mux
- Multi-bit Addition
- multiplexers, demultiplexers
  - use In Creating Pathways
  - use In Implementing Combinatorial functions (using F.A. as an example)

The story Continues..

We have seen two Implementations of the F.A., each used to illustrate a Specific idea. It should be clear therefore that there Is not only one way of Implementing the F.A. – or Indeed any Combinatorial function.

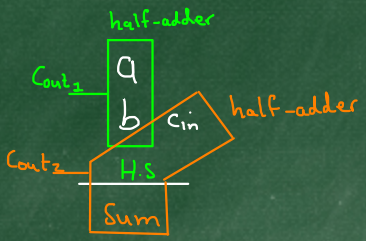
Let's look at another, to foster some Lateral thinking:

Recall that the Half-adder circuit adds two bits to give a half-sum and a Carry out.

Also recall that the Full-adder Circuit adds three bits to give a Sum and a Carry out.

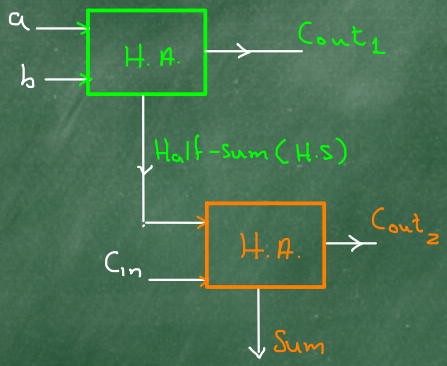
Question: Can we use Half-adders to Implement a Full-adder?

To add a, b and a Carry In (c_{in}), Consider the following:



The Sum would appear to be easy enough to generate.
But what should we do with the two Carry outs?

Let's redraw using Black box notation to make this problem clearer:



Maybe we can get some Insight from a Truth Table

a	b	Hs	$Cout_1$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Let's now add $C_{in}=0$ to Hs and $C_{in}=1$ to Hs

C_{in}	Hs	S	$Cout_2$
0	0	0	0
0	1	1	0
0	1	1	0
0	0	0	0

C_{in}	Hs	S	$Cout_2$
1	0	1	0
1	1	0	1
1	1	0	1
1	0	1	0

Now, let's tabulate the Full-adder Sum and Cout and Compare the results with the Sum we get from Combining the two half-adders. The value of $Cout_1$ which only comes from add a and b. And $Cout_2$ which Comes from adding the Hs to C_{in} :

C_{in}	a	b	Sum	Cout	Sum	$Cout_1$	$Cout_2$
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	1	0	0	1
1	1	0	0	1	0	0	1
1	1	1	1	1	1	1	0

We see that the Sum columns are the same (as expected) we also see that we can get the F.A. Carry out by 'OR'ing the $Cout_1$ and $Cout_2$ columns.

Therefore, to implement a F.A. with two H.A.s :