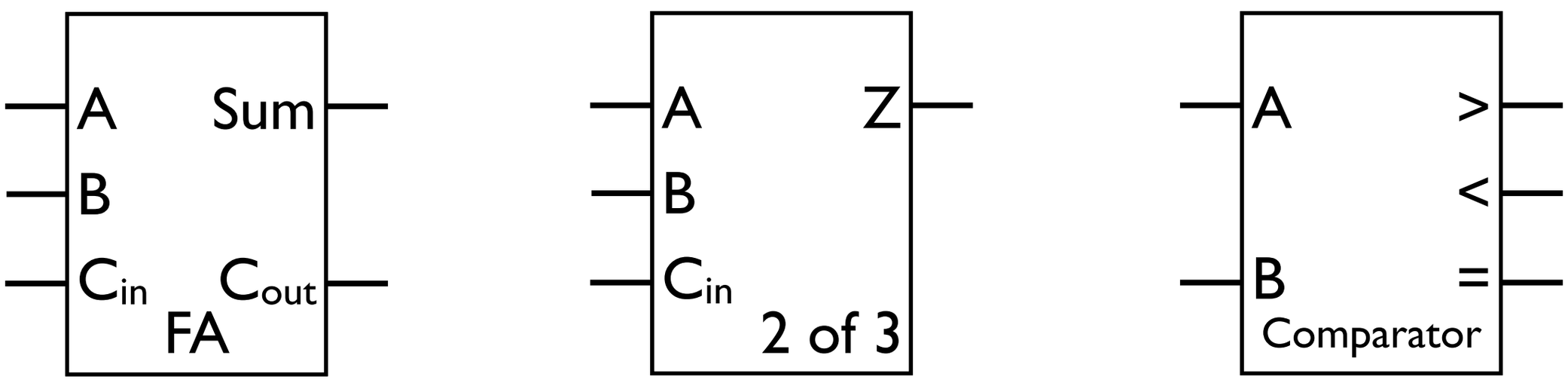
Logic Design Process

ENCE260: Computer Architecture Topic 6

Recap

- We can combine logic variables together using AND, OR and NOT gates to create new devices.
- But how do we know the logic circuits we design are *correct* and *optimal*?



Ciaran Moore
ciaran.moore@canterbury.ac.nz

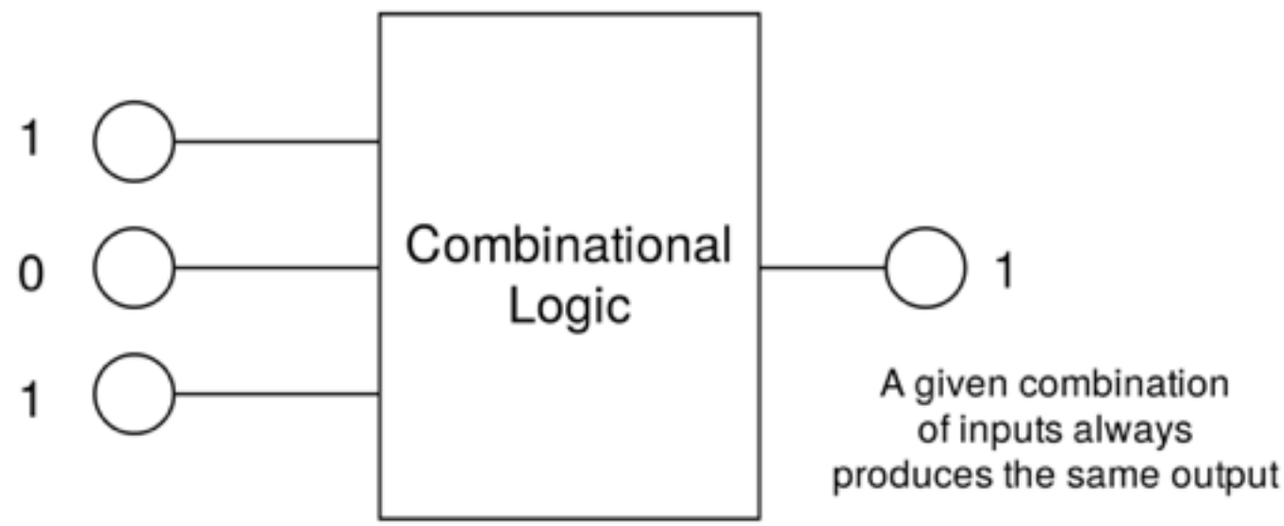
Challenge

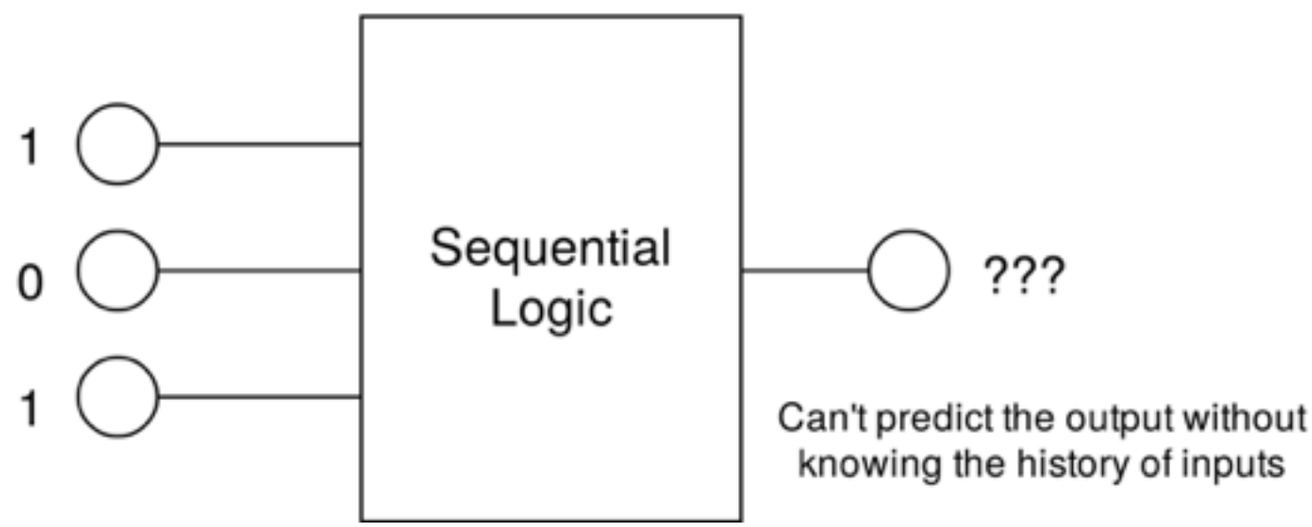
- Design, optimise and verify (combinational) logic circuits to perform arbitrary functions.

unsplash.com

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Combinational Logic





Ciaran Moore
ciaran.moore@canterbury.ac.nz

Combinational Logic Design Process

1. Define the function using a **truth table**
2. Convert the truth table to a **Boolean expression**
3. **Simplify** the expression
4. Verify the expression using a **Karnaugh map**
5. Map Boolean operators to **logic gates**

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Combinational Logic Design Process

1. Define the function using a **truth table**
2. Convert the truth table to a **Boolean expression**
3. **Simplify** the expression
4. Verify the expression using a **Karnaugh map**
5. Map Boolean operators to **logic gates**

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Truth Table

Inputs		Output
a	b	$AND(a,b)$
0	0	0
0	1	0
1	0	0
1	1	1

Every possible combination of input values

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Truth Table

```
def AND(a, b):
    truth_table = {(0, 0): 0,
                   (0, 1): 0,
                   (1, 0): 0,
                   (1, 1): 1}
    return truth_table[(a, b)]
```


Ciaran Moore
ciaran.moore@canterbury.ac.nz

Truth Table

- We can think of the truth table as a *dictionary* or *lookup table* for finding an output value based on the input values.
- In fact, we can implement the `AND(a,b)` Boolean function in Python using a `dict` data structure.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Fundamental Functions

NOT $f(a) = \overline{a}$

a	$\bar{a}$
0	1
1	0

AND $f(a, b) = a \cdot b$

a	b	a · b
0	0	0
0	1	0
1	0	0
1	1	1

OR $f(a, b) = a + b$

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Combinational Logic Design Process

1. Define the function using a **truth table**
2. Convert the truth table to a **Boolean expression**
3. **Simplify** the expression
4. Verify the expression using a **Karnaugh map**
5. Map Boolean operators to **logic gates**

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Truth Table to Expression

a	b	c	$f(a,b,c)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$$f(a,b,c) = a\bar{b}c$$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

A More Complex Example

a	b	c	z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

$$z = \bar{a}\,\bar{b}\,c + a\,\bar{b}\,\bar{c} + a\,\bar{b}\,c + a\,b\,c$$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Truth Table to Expression

- We look for rows in the truth table where the output is 1.
- We then write down the specific combination of inputs for each row where the output is 1.
- For functions with multiple rows giving a 1 output, we OR the input combinations together to create a *Sum-of-Products* expression.
- Finally, we simplify the SoP expression to allow an efficient logic gate implementation.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Aside: Minterms

- A *minterm* is an AND of all inputs (or their inverses).

term	minterm?
abc	
$\bar{a}bc$	
$\bar{a}b$	
b	
$a(b+c)$	

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Aside: Minterms

- A *minterm* is an AND of all inputs (or their inverses).

term	minterm?
abc	✅
$\bar{a}bc$	✅
$\bar{a}b$	❌
b	❌
$a(b+c)$	❌

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Aside: Minterms

- A *minterm* is an AND of all inputs (or their inverses).
- Combining minterms gives us a *sum-of-products* (or-of-ands) form of the function.
- Also called *sum-of-minterms* or *minterm canonical form*.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Combinational Logic Design Process

1. Define the function using a **truth table**
2. Convert the truth table to a **Boolean expression**
3. **Simplify** the expression
4. Verify the expression using a **Karnaugh map**
5. Map Boolean operators to **logic gates**

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Truth Table Simplification

a	b	c	$a\bar{b}$	ac	$\bar{b}c$	$a\bar{b}+ac+\bar{b}c$	z
0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1
0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	1	0	0	1	1
1	0	1	1	1	1	1	1
1	1	0	0	0	0	0	0
1	1	1	0	1	0	1	1

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Truth Table Simplification

$$\overline{a}\,\overline{b}\,c + a\,\overline{b}\,\overline{c} + a\,\overline{b}\,c + a\,b\,c$$

$$=$$

$$a\,\overline{b} + a\,c + \overline{b}\,c$$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Truth Table Simplification

- Our first definition for z had groups of three terms AND'd together.
- Looking at products (i.e. ANDs) of two variables reveals a way to simplify the expression for z.
- Can we do this simplification in a systematic way?

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Algebraic Simplification

- We can pair minterms together and apply the Distributive, Or-Complement, and Identity rules:

$$a\,b\,\bar{c} + a\,b\,c \;=\; a\,b(\bar{c} + c) \;=\; a\,b$$

$$a\,\bar{b}\,c + a\,b\,c \;=\; a(\bar{b} + b)c \;=\; a\,c$$

$$\bar{a}\,\bar{b}\,c + a\,\bar{b}\,c \;=\; (\bar{a} + a)\bar{b}\,c \;=\; \bar{b}\,c$$

- Note that we can use the same minterms multiple times, thanks to idempotence (a + a = a).

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Algebraic Simplification

Identity	$a \cdot 1 = a$ and $a + 0 = a$
Annihilation	$a \cdot 0 = 0$ and $a + 1 = 1$
Or-Complement	$a + \overline{a} = 1$
And-Complement	$a \cdot \overline{a} = 0$
Associative	$a + (b + c) = (a + b) + c, \quad a(bc) = (ab)c$
Commutative	$a + b = b + a, \quad ab = ba$
Distributive	$a(b + c) = ab + ac, \quad a + (bc) = (a + b) \cdot (a + c)$
Double Negation	$\overline{\overline{a}} = a$
Absorption	$a + (a \cdot b) = a$ and $a \cdot (a + b) = a$
Idempotence	$a + a = a$ and $a \cdot a = a$
De Morgan's Laws	$\overline{a + b} = \overline{a} \cdot \overline{b}$ $\overline{a \cdot b} = \overline{a} + \overline{b}$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Example

$$
\begin{aligned}
y &= \bar{a}\,\bar{b}\,c + a\,\bar{b}\,\bar{c} + a\,\bar{b}\,c + a\,b\,\bar{c} \\
&= \bar{a}\,\bar{b}\,c + a\,\bar{b}\,c + a\,\bar{c}\,\bar{b} + a\,\bar{c}\,b \quad [Commutative] \\
&= (\bar{a} + a)\bar{b}\,c + a\,\bar{c}(\bar{b} + b) \quad [Distributive] \\
&= 1 \cdot \bar{b}\,c + a\,\bar{c} \cdot 1 \quad [OrComplement] \\
&= \bar{b}\,c + a\,\bar{c} \quad [Identity]
\end{aligned}
$$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

"Don't Care" Inputs

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	X	X	1

Ciaran Moore
ciaran.moore@canterbury.ac.nz

"Don't Care" Inputs

- In some truth tables we can see that the output is *the same* regardless of the value a particular input takes.
- In these situations we can simplify the truth table by marking the inputs as "don't care" (× instead of 0 or 1).
- This is the same as performing an algebraic simplification.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Combinational Logic Design Process

1. Define the function using a **truth table**
2. Convert the truth table to a **Boolean expression**
3. **Simplify** the expression
4. Verify the expression using a **Karnaugh map**
5. Map Boolean operators to **logic gates**

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Karnaugh Maps

f	$\bar{a}\bar{b}$	$\bar{a}b$	ab	$a\bar{b}$
$\bar{c}\bar{d}$				
$\bar{c}d$				
cd				
$c\bar{d}$				

$f(a,b,c,d)$

Adjacent cells differ by a single variable

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Karnaugh Maps

$$f(a,b,c,d) = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}c\bar{d} + abc\bar{d}$$

f	$\bar{a}\bar{b}$	$\bar{a}b$	ab	$a\bar{b}$
$\bar{c}\bar{d}$	1	0	0	0
$\bar{c}d$	0	1	0	0
cd	0	1	0	0
$c\bar{d}$	1	1	1	1

$$f(a,b,c,d) = c\bar{d} + \bar{a}bd + \bar{a}\bar{b}\bar{d}$$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Karnaugh Maps

- K-maps help us to simplify logical expressions of up to four variables.
- (Horizontally and vertically, but not diagonally) neighbouring cells differ by only a single variable.
- Each cell in the K-map corresponds to a single minterm, which may or may not be part of the function.
- Groups of (Horizontal or Vertical, not Diagonal) neighbouring 1's in the K-map are combined to give expressions in the simplified function.
- NB: Group size is always a power of 2, e.g. 2, 4, 8, 16 etc.
- Groups can overlap and wrap around the boundaries of the K-map.

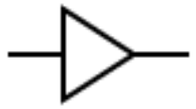
Ciaran Moore
ciaran.moore@canterbury.ac.nz

Combinational Logic Design Process

1. Define the function using a **truth table**
2. Convert the truth table to a **Boolean expression**
3. **Simplify** the expression
4. Verify the expression using a **Karnaugh map**
5. Map Boolean operators to **logic gates**

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Logic Gates

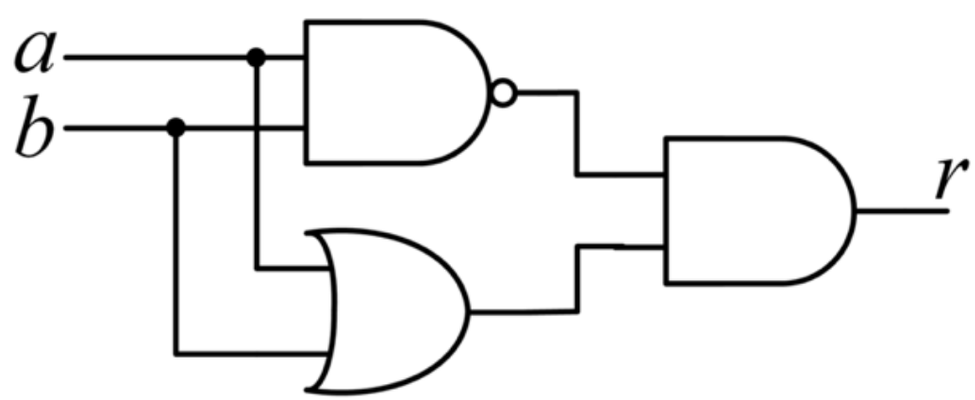
Straight back		AND
Curved back		OR
Triangle		Buffer
Double curved back		XOR

Ciaran Moore
ciaran.moore@canterbury.ac.nz

XOR Gate

Inputs		Output
a	b	$XOR(a,b)$
0	0	0
0	1	1
1	0	1
1	1	0

$$
\begin{aligned}
r(a,b) &= \overline{ab} \cdot (a+b) \\
&= (\overline{a}+\overline{b}) \cdot (a+b) \quad [De\ Morgan] \\
&= \overline{a}a + \overline{b}a + \overline{a}b + \overline{b}b \quad [Distributive] \\
&= a\overline{b} + \overline{a}b \quad [AndComplement]
\end{aligned}
$$



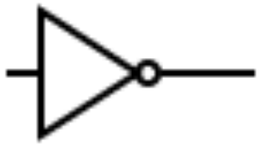
=

Ciaran Moore
ciaran.moore@canterbury.ac.nz

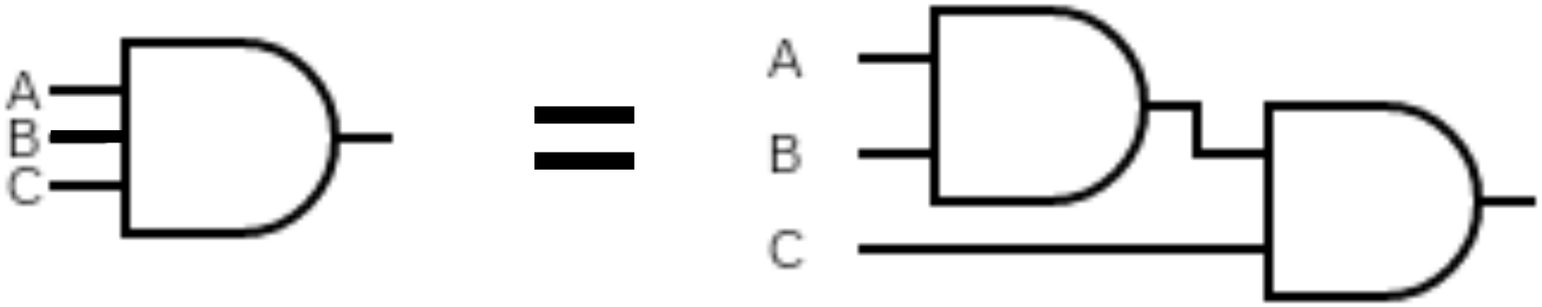
Logic Gates

- Bubbles negate:  NAND

 NOR

 NOT

- Gates can have more than two inputs:

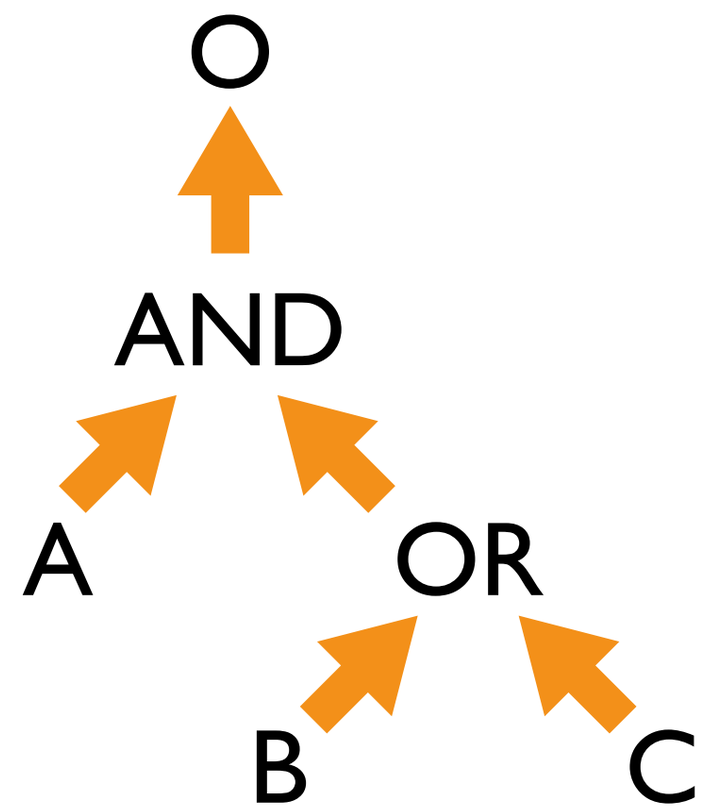


Ciaran Moore
ciaran.moore@canterbury.ac.nz

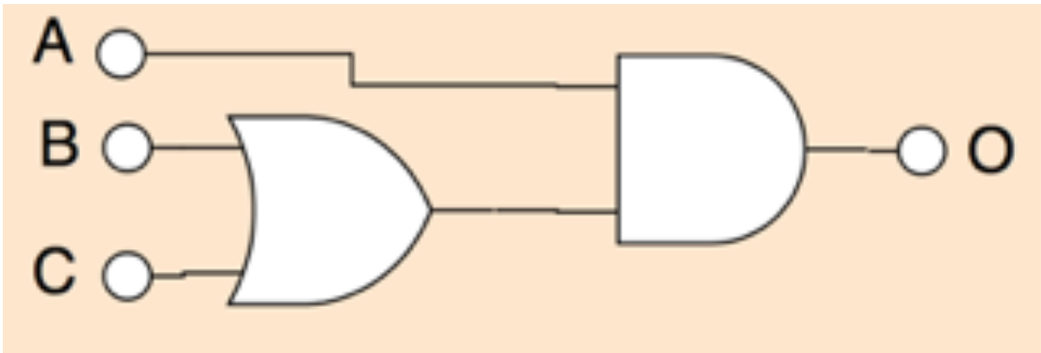
Converting Expressions to Logic Circuits

- Boolean expressions can be thought of as a "tree" of operations, each combining values from lower down the "tree":

O = A(B+C)



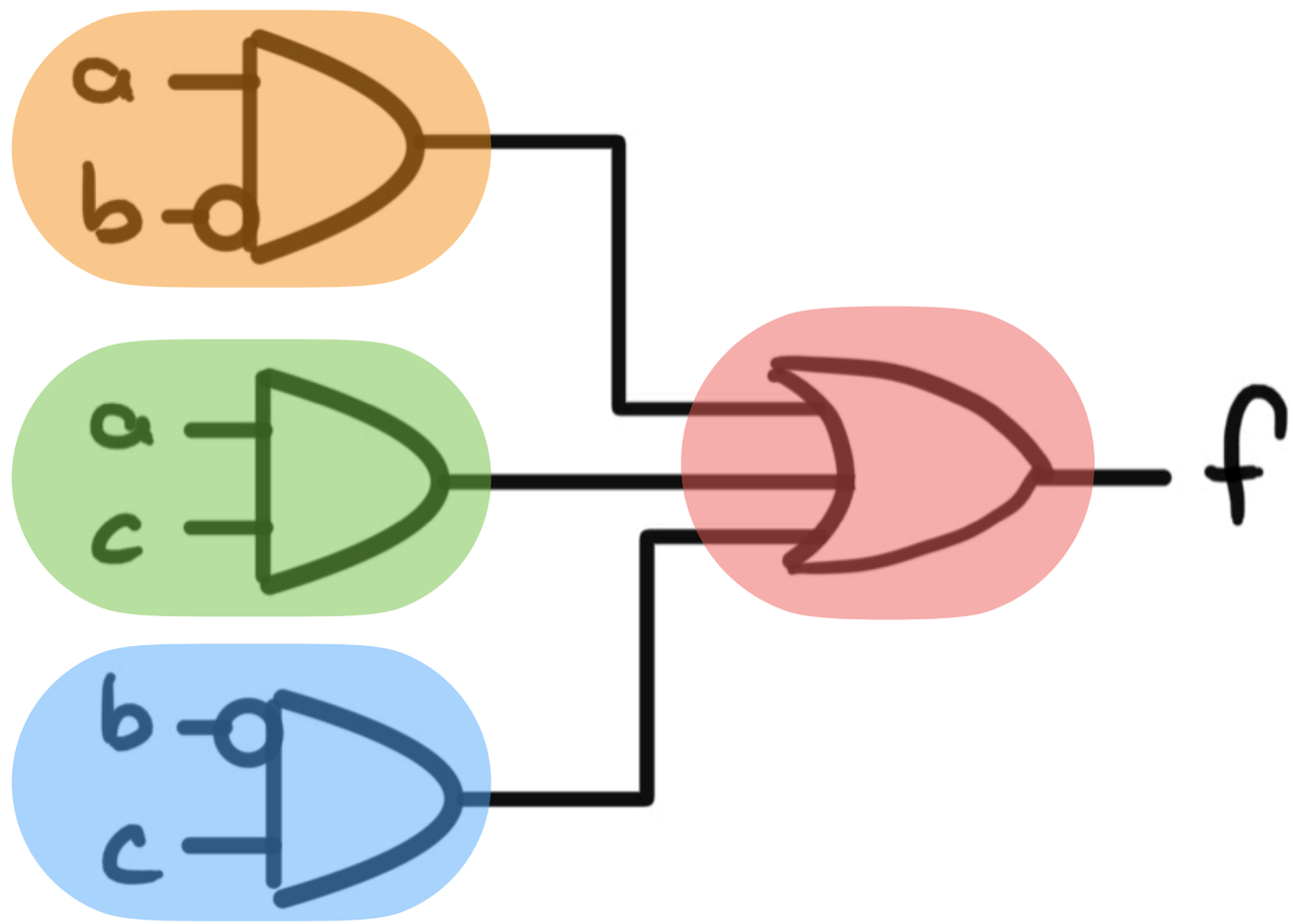
- The shape of this logic tree gives us the shape of our logic circuit:



Ciaran Moore
ciaran.moore@canterbury.ac.nz

Logic Gate Mapping

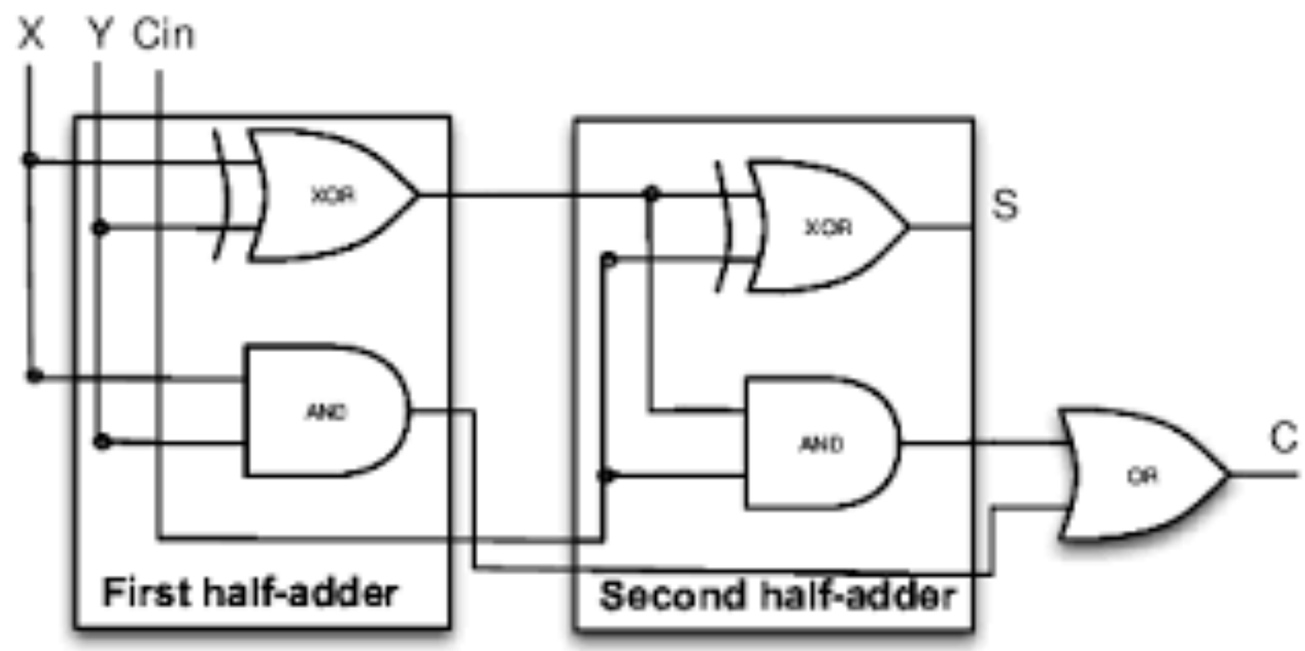
- $f = a\overline{b} + ac + \overline{b}c$



Ciaran Moore
ciaran.moore@canterbury.ac.nz

Design Process

- We can speed up the design process by reusing common circuit blocks to create *hierarchical* designs.
- For example, we can use two half-adders and an OR gate to make a full adder.



Ciaran Moore
ciaran.moore@canterbury.ac.nz

Summary

- Follow the *design process* (truth table → boolean expression → digital logic circuit) to realise a wide range of *combinational* logic circuits.
- Use *Karnaugh maps* to simplify your designs and check their accuracy.
- Reuse logic blocks to build *hierarchical* combinational logic circuits.

Ciaran Moore
ciaran.moore@canterbury.ac.nz