# Lecture 4

From problem description to circuit

# In this lecture we will:

Create circuits from Truth Tables.

Use Karnaugh Maps to minimise circuits.

Work through a design exercise.

# Analysing and transforming circuits

So far we have studied how to analyse and to transform circuits, for example:

Given a circuit

- find its truth table

Given a Boolean equation

- simplify it

etc.

# Circuit Design:

Circuit design brings all our analysis work together.

From a given description of the required behaviour of a circuit (its truth table) we design an actual circuit that implements it.

The solution is NOT unique.

We will look for the “best” solution.

# Two methods which always work

1) Synthesise the circuit from the truth table by using minterms.

ie OR-ing the 1s in the truth table.

2) Synthesise the circuit from the truth table by using maxterms.

ie AND-ing the 0s in the truth table.

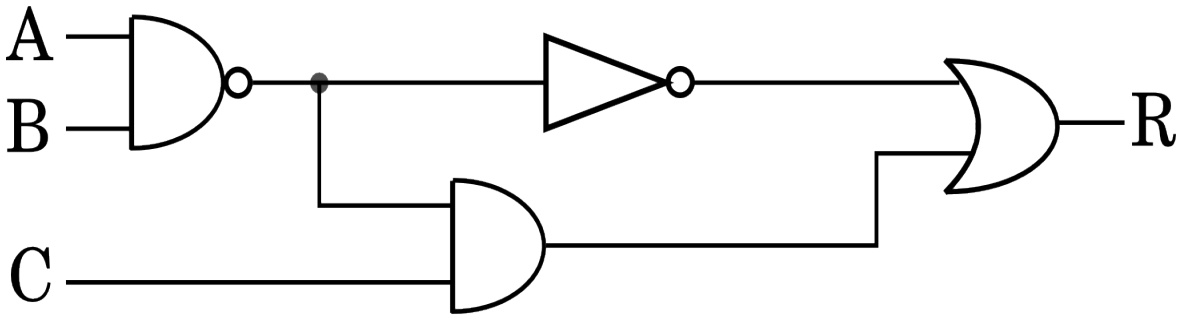
# Synthesise a circuit by Minterms

We start with the truth table which we can get from the specification, or a Boolean equation:

| A | B | C | R |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

R=(A.B)'' + (A.B)'.C

Or a circuit



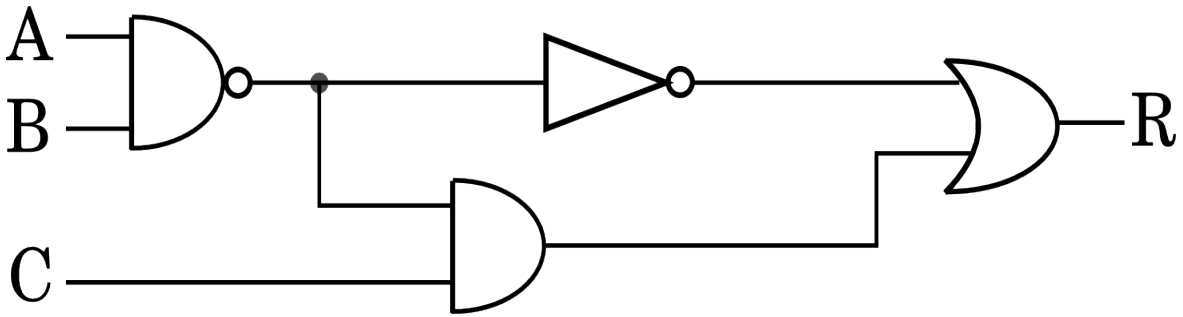
# Synthesise a circuit by Minterms

We start with the truth table which we can get from the specification, or a Boolean equation:

| A | B | C | R |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

R=(A.B)'' + (A.B)'.C

Or a circuit




Find all the conditions making the output 1

R = A'.B'.C + A'.B.C + A.B'.C + A.B.C' + A.B.C

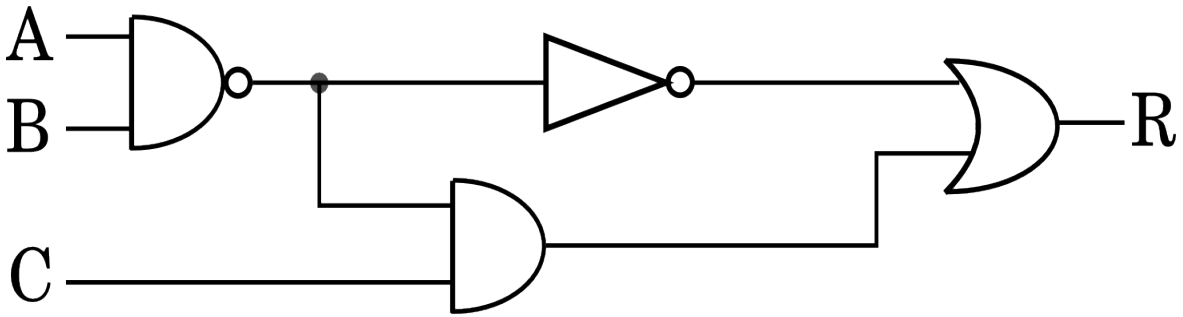
# Synthesise a circuit by Maxterms

We start with the truth table which we can get from the specification, or a Boolean equation:

| A | B | C | R |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

R=(A.B)'' + (A.B)'.C

Or a circuit

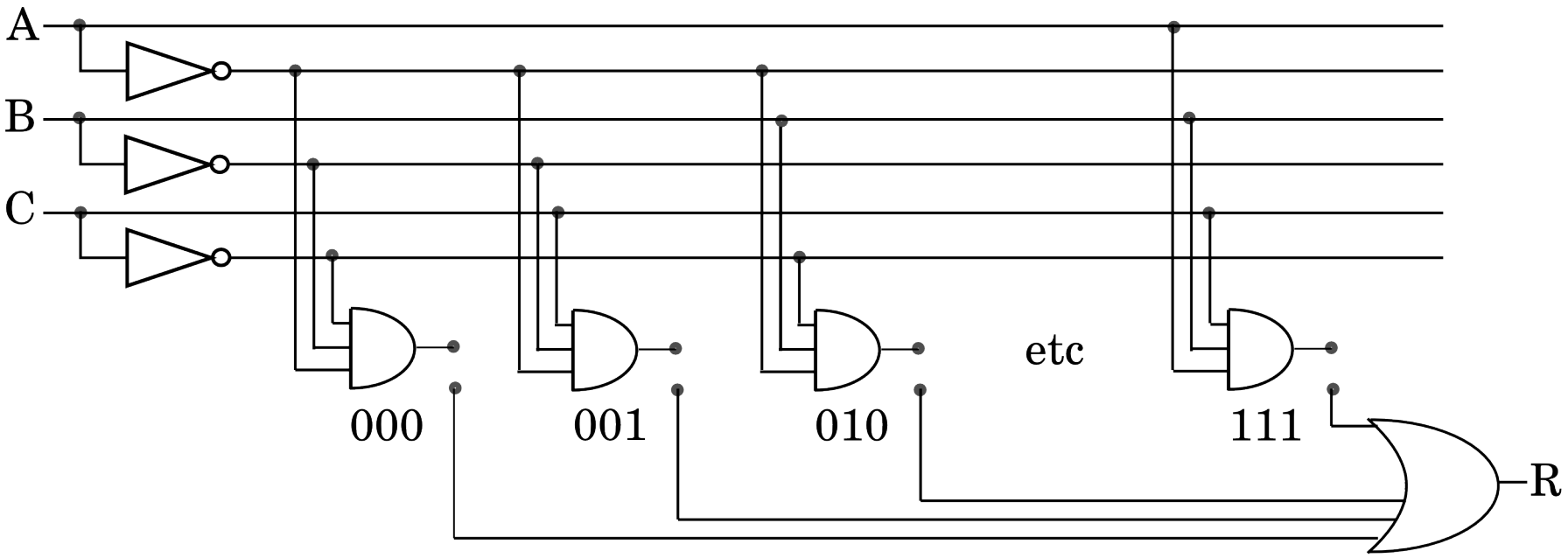



Find all the conditions making the output 0

R = (A+B+C).(A+B'+C).(A'+B+C)

# The universal hardware solution

The Programmable Logic Array - a direct implementation from the canonical minterm form



# Programmable Logic Array chips

Programming – Antifuse “fusable links”.

All possible minterms are considered and added in the final OR gate. (Only the connected ones are implemented).

PAL cannot express the Dual - Boolean product of Maxterms. A different underlying structure is required.

AND: there are no benefits in combinatorial minimisation.

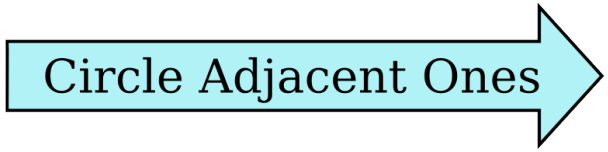
# Karnaugh map Minimisation

| A | B | C | R |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

The programmable logic array is a fast design method, but it does not produce the smallest or fastest result.

As an alternative we can minimise the circuit with a Karnaugh map.

| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |




| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

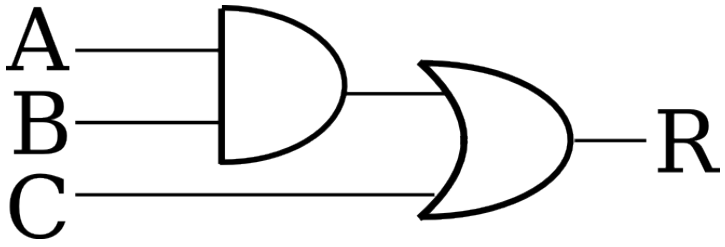
# Minimised circuit

The minimised circuit is much smaller in area and faster than both the original circuit or the programmable logic array.

| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

Circle Adjacent Ones

| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |




$$R = C + A.B$$

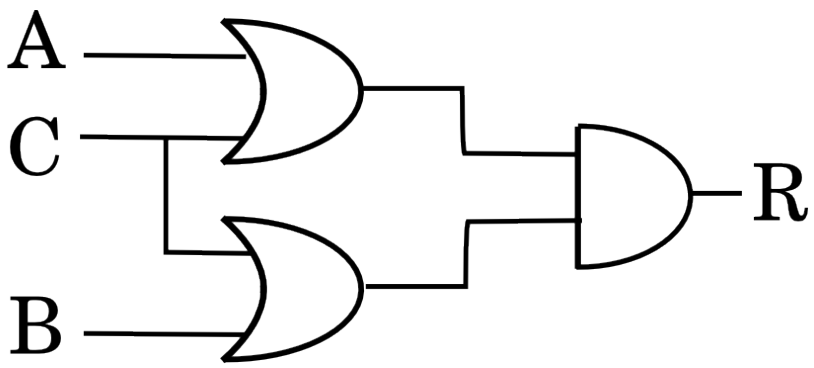
# Minimising with Maxterms

We can also try minimising using the maxterms. In this case we circle the zeros.

However, the final result is not as good.

| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

$$R = (B+C).(A+C)$$



# Design Exercise - Definition

Build a combinatorial digital circuit with three inputs and one output. Inputs A and B are for data, C is for control.

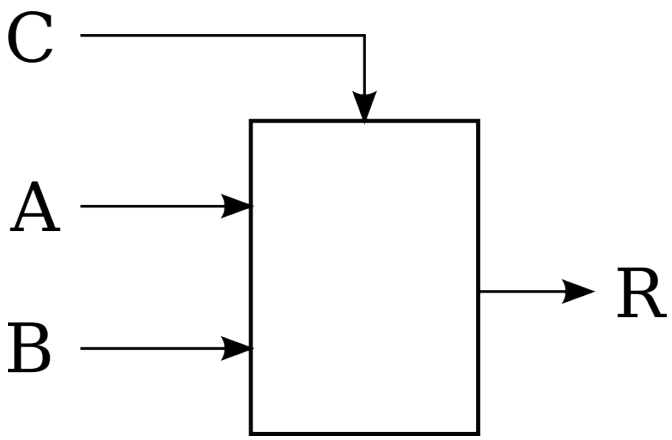
if C = 0: R = A . B

if C = 1: R = A + B

Design the circuit to be as small as possible.

# Formalising the specification

The first stage is to formalise the specification. In this case it isn't too difficult.




| Control | R |
|---|---|
| 0 | A.B |
| 1 | A+B |

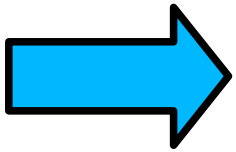
$$R_{(C=0)} = A \cdot B$$

$$R_{(C=1)} = A + B$$

# Generate the truth table

The truth table is a complete unambiguous specification of the circuit. It is easy to generate in this case.

| Control | R |
|---|---|
| 0 | A.B |
| 1 | A+B |



| C | A | B | R |
|---|---|---|---|
| 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 |

# Use the Karnaugh Map

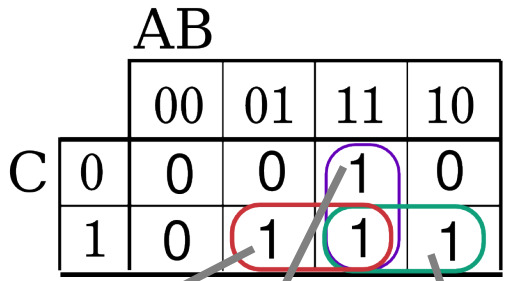
| C | A | B | R |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

The truth table is used to fill up the Karnaugh Map.

We circle the ones and find the minimum equation.

| C \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

Circle Adjacent Ones




$$R = B.C + A.B + A.C$$

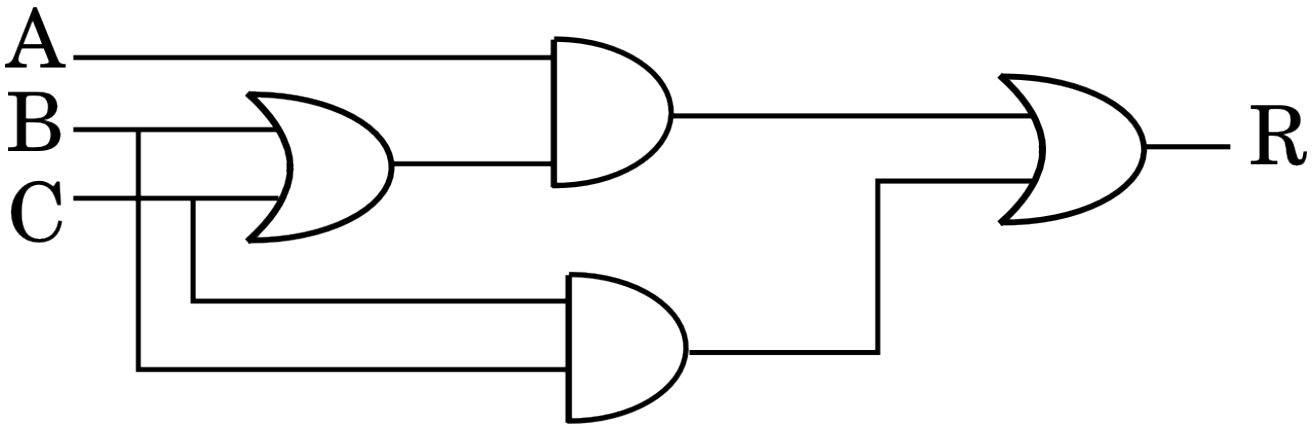
# Further factorisation

In this case we can find a further factorisation:

$R = B.C + A.B + A.C$

$R = B.C + A.(B+C)$

We can now draw the final circuit.



# Problem Break

Use maxterms to find the minimum circuit, starting with the same Karnaugh Map.

| | | AB | | | |
|---|---|---|---|---|---|
| | | 00 | 01 | 11 | 10 |
| C | 0 | 0 | 0 | 1 | 0 |
| | 1 | 0 | 1 | 1 | 1 |

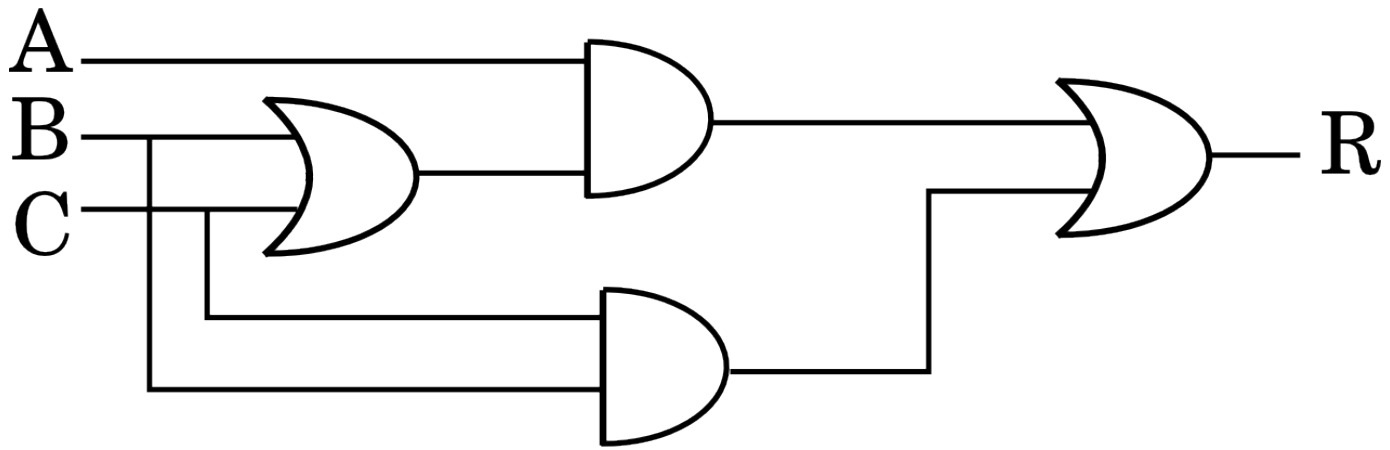
# Making the circuit small

Different gates have different sizes when manufactured as part of a silicon chip. The approximate relative sizes are:

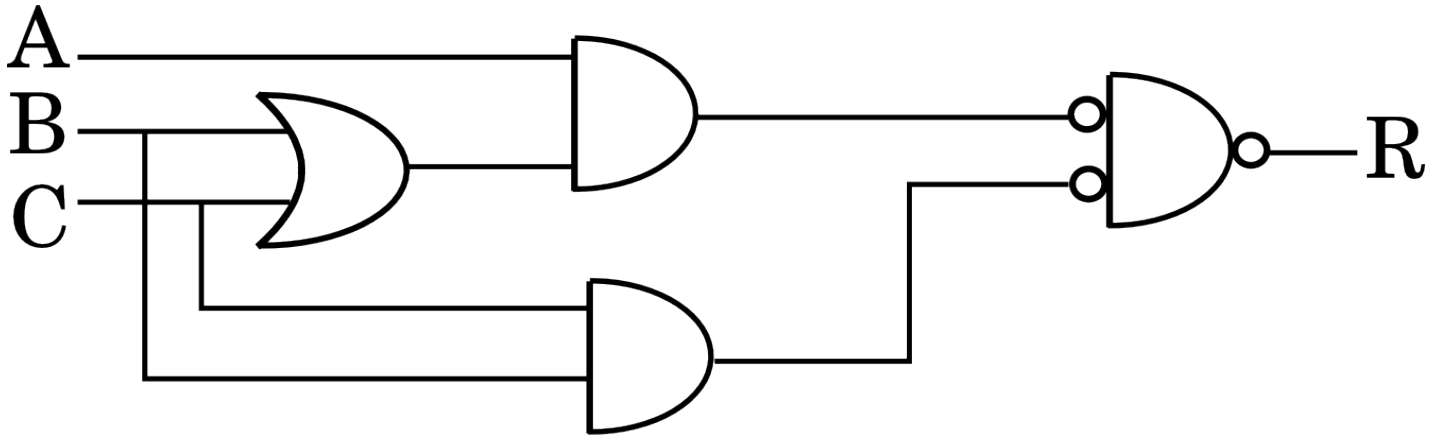
| Gate | Nominal Size |
|---|---|
| INVERTER | 3 |
| NAND, NOR | 4 |
| AND, OR | 6 |
| XOR, XNOR | 8 |

Our circuit design uses AND and OR gates, so we should be able to reduce it's size by changing them to NAND and NOR.

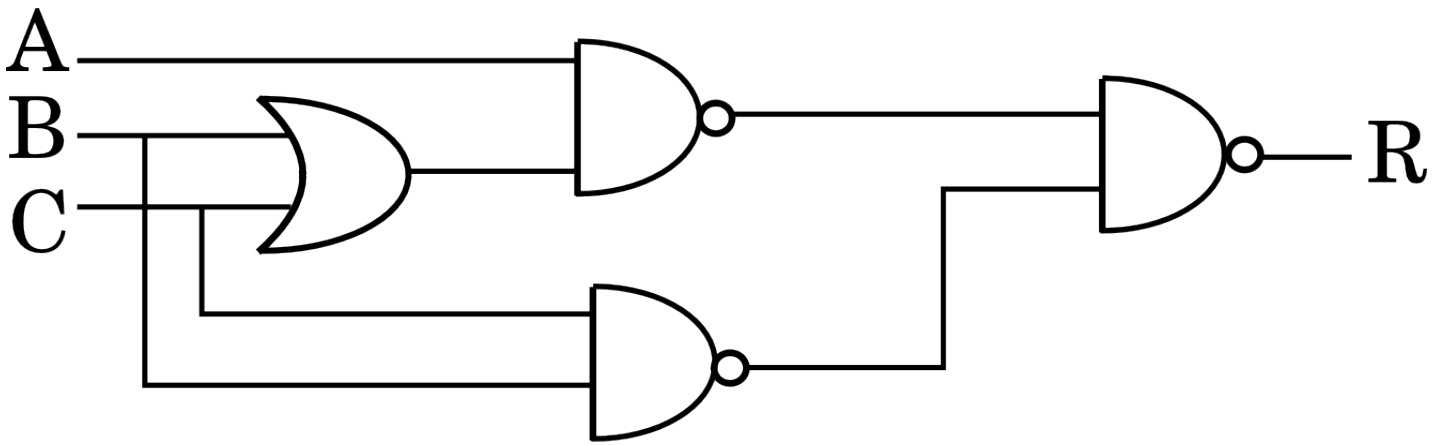
# Swap OR for NAND




Applying de Morgan's theorem to the last OR gate we re-draw the circuit as:



# Regroup the inverters




It looks like we have made an improvement. The size count has gone down from 24 to 18.

# Testing

Unfortunately debugging hardware is difficult. The final circuit may not implement the original specification for a few - difficult to identify - inputs

For example the Intel Pentium P5 had a floating division bug:

Clearly: 4195835*3145727/3145727 = 4195835,

but on a P5 you would get

4195835*3145727/3145727 = 4195579 !

The chip was withdrawn and re-manufactured, incurring a high rectification cost.

# Systematic Testing

Feed a large range of inputs to the circuit and observe its behaviour.

This is done using simulation before manufacture, and after manufacture with real tests.

The problem is that there are too many possible input combination to test a large circuit exhaustively. Thus is is not feasible to identify all bugs.

# Formal techniques

This approach aims to verify the chip at design stage (before silicon)

Abstraction, simulation equivalence, and bounded model checking are used extensively to identify bugs

However: it's very hard to reason about all possible behaviours because of the very large state spaces.

The Intel Core i7 EXE module validated by formal techniques.

# Now it's your turn - Coursework 1

The first coursework is a design exercise like the one we have just done.

Everybody has their own circuit to design.

Testing is done with a simple in house simulator called DIGISIM.

The handout contains a full worked solution.