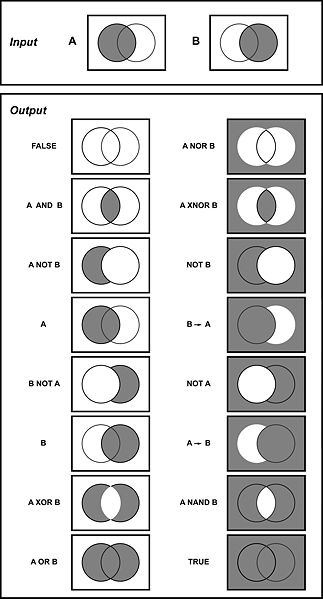
Logic Workbook Name…………………………………….Class…….

[](http://upload.wikimedia.org/wikipedia/commons/1/11/LogicGates.j)

Contents

[Logic Gates and Boolean Notation 3](#_Toc492887179)

[Task: 1. Logic gate Truth Tables 3](#_Toc492887180)

[Task: 2. Some straightforward word problems 5](#_Toc492887181)

[Task: 3. Boolean algebraic formula practice 6](#_Toc492887182)

[Task: 4. Simple Gate combination truth tables 7](#_Toc492887183)

[Working out Boolean equations from circuits 8](#_Toc492887184)

[Example 1 8](#_Toc492887185)

[Example 2 9](#_Toc492887186)

[Task: 5. Boolean Expressions from Circuits 12](#_Toc492887187)

[Task: 6. Boolean Circuit to Algebra Practice 14](#_Toc492887188)

[Task: 7. Circuit and truth table from an equation 16](#_Toc492887189)

[Boolean Identities and Simplification 17](#_Toc492887190)

[Task: 8. Logism Identity proof 17](#_Toc492887191)

[By Truth Table 17](#_Toc492887192)

[Truth Table Simplification example 18](#_Toc492887193)

[Task: 9. Finish the Truth Table 18](#_Toc492887194)

[Task: 10. Truth table simplification practice 19](#_Toc492887195)

[Task: 11. Simplify the following expressions as much as possible. 25](#_Toc492887196)

[Truth Table Proofs 26](#_Toc492887197)

[Task: 12. Algebraic Boolean simplification practice 1 26](#_Toc492887198)

[Task: 13. Draw the Logic Circuit diagram for the Boolean expressions given. 28](#_Toc492887199)

[Task: 15. More Boolean simplification Practice 2 29](#_Toc492887200)

[Appendix A 31](#_Toc492887201)

|  |
| --- |
| Logic Gates and Boolean Notation Logic Gates Construct truth tables for the following gates: NOT, AND, OR, XOR, NAND, NOR. |

### Logic gate Truth Tables

Complete the truth tables to show the output generated for the given inputs.

**AND gate**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| AND | |  |  |  | | --- | --- | --- | | A | B | A.B | | 0 | 0 |  | | 0 | 1 |  | | 1 | 0 |  | | 1 | 1 |  | |

The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. A.B (BEWARE!!!! X, \* or just a space are also used )

**OR gate**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OR | |  |  |  | | --- | --- | --- | | A | B | A+B | | 0 | 0 |  | | 0 | 1 |  | | 1 | 0 |  | | 1 | 1 |  | |

The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high.  A plus (+) is used to show the OR.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| NOT1 | |  |  | | --- | --- | | A | Ā | | 0 |  | | 1 |  | |

It is also known as an *inverter*.  If the input variable is A, the inverted output is known as NOT A.

(A with a bar over the top), A’, ~A, ¬A, are all common notations

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NAND | |  |  |  | | --- | --- | --- | | **INPUT** | | **OUTPUT** | | A | B | A NAND B | | 0 | 0 |  | | 0 | 1 |  | | 1 | 0 |  | | 1 | 1 |  | |

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if **any** of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

**NOR gate**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NOR | |  |  |  | | --- | --- | --- | | **INPUT** A   B | | **OUTPUT** A NOR B | | 0 | 0 |  | | 0 | 1 |  | | 1 | 0 |  | | 1 | 1 |  | |

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate.  The outputs of all NOR gates are low if **any** of the inputs are high.

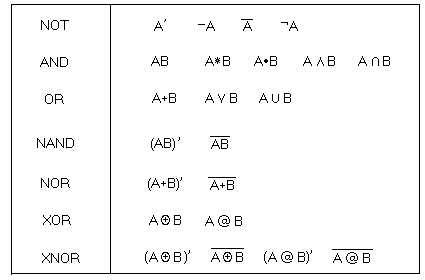
The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

**XOR gate**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EOR | |  |  |  | | --- | --- | --- | | **INPUT A   B** | | **OUTPUT A XOR B** | | **0** | **0** |  | | **0** | **1** |  | | **1** | **0** |  | | **1** | **1** |  | |

The '**Exclusive-OR**' gate is a circuit which will give a high output if **either, but not both**, of its two inputs are high.  An encircled plus sign (enplus) is used to show the EOR operation.

Summary of Alternative Notations



### Some straightforward word problems

**What type of “gate” would be used in the following situations?**

A domestic alarm will go off if it detects smoke or heat

A car engine will start if it has a key turned and a security code input

A waiter offers you tea or coffee, but not both, after a meal. You must have one as it was paid for in the cost of the whole meal

A waiter in another restaurant offers you tea or coffee or neither, but not both, after a meal.

An emergency red LED lighting system comes on in a room when a sensor detects that the usual power supply has failed

The principal will give a special prize to anyone who has not missed any lessons or had a grade 2 in an interim report

### Boolean algebraic formula practice

Write the following using the algebraic notation. Operands in lower Case. Operators Upper case. Make sure the order of operation is clear

1. a Or Zero
2. NOT a AND Zero
3. a OR NOT a
4. a OR a
5. a OR a AND b
6. a OR NOT a AND b
7. a AND (NOT a OR b)
8. a AND b OR NOT a AND b
9. (NOT a OR NOT b ) AND (NOT a OR b)
10. y OR y AND NOT y
11. x AND y OR x AND NOT y

### Simple Gate combination truth tables

Study the following circuit and complete the table to show the output achieved for the given inputs

A, B and C are inputs. Q is the output

|  |  |  |
| --- | --- | --- |
| A | B | Q |
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 |  |
| 1 | 1 |  |



A

B

|  |  |  |
| --- | --- | --- |
| A | B | Q |
| 0 | 0 |  |
| 0 | 1 |  |
| 1 | 0 |  |
| 1 | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | Q |
| 0 | 0 | 0 | A |
| 0 | 0 | 1 | B |
| 0 | 1 | 0 |  |
| 0 | 1 | 1 | C |
| 1 | 0 | 0 |  |
| 1 | 0 | 1 |  |
| 1 | 1 | 0 |  |
| 1 | 1 | 1 |  |

As the combinations become more complex we will needs a strategy:

# Working out Boolean equations from circuits

Sometimes we are not given a truth table, but a logic circuit diagram from which we have to derive the Boolean expression. This sounds complicated but as long as you are careful in what you do and work sensibly from the inputs across to the outputs you should be o.k.

Let us work through an example together.

Example 1: Derive the Boolean expression for the following logic circuit.

To start the process try to divide the circuit into stages as shown above.

**A**

**B**

**C**

**Q**

Stage 1 Stage 2 Stage 3

This breaks down a large circuit into smaller more manageable chunks. It would be virtually impossible to write down the expression for Q immediately just by looking at the logic diagram, so don’t even try, you are more than likely to make a mistake.

Having identified the different stages in the circuit diagram, we now proceed to write down the Boolean expressions for all of the output sections of stage 1, as shown on the following page.

**A**

**B**

**C**

**Q**

Stage 1 Stage 2 Stage 3

**A**

**B**

**C**

Now we move on to look at the outputs of stage 2, as shown below:

And finally we can complete stage 3, to arrive at the expression for the system as

**A**

**B**

**C**

**Q**

Stage 1 Stage 2 Stage 3

**A**

**B**

**C**

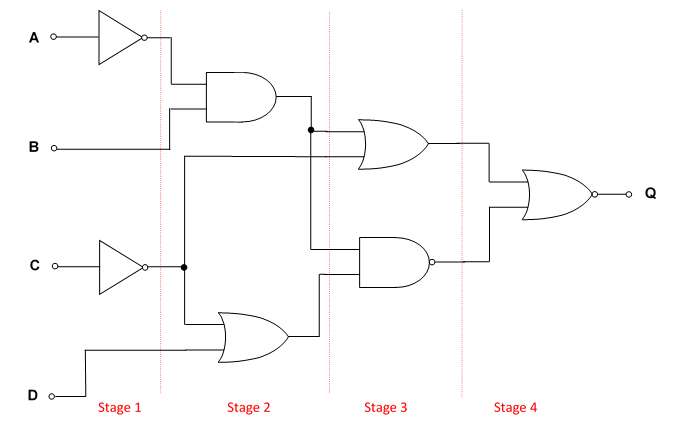
**A.B**

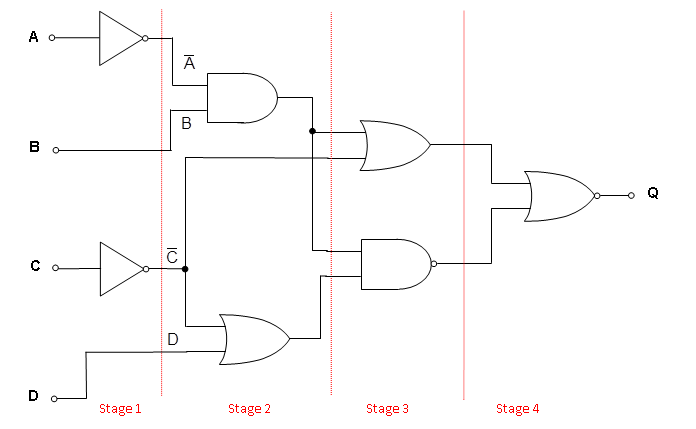
**C**

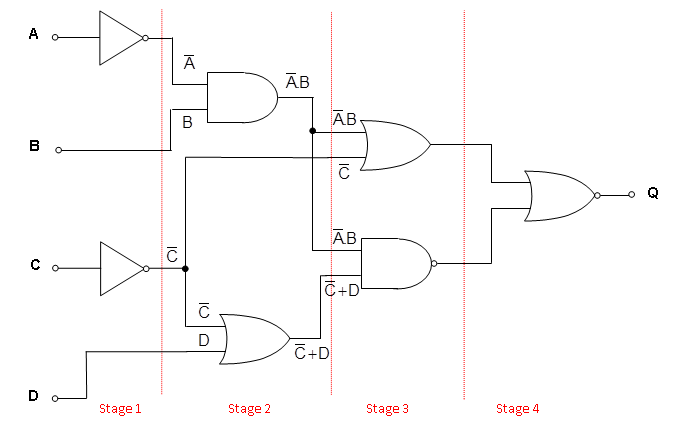


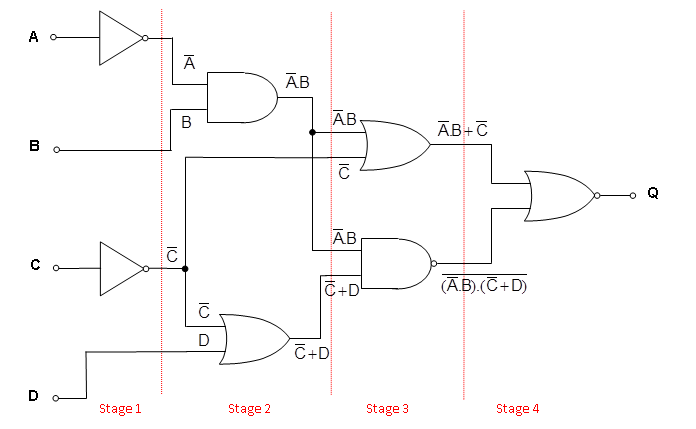
## Example 2

So to begin with we will work out all of the outputs in stage 1.



Then we look at stage 2.

Now for stage 3, we have to be very careful as for the first time we are going to come across NAND gates, and it is really important to keep terms together, so we use brackets to keep things together.



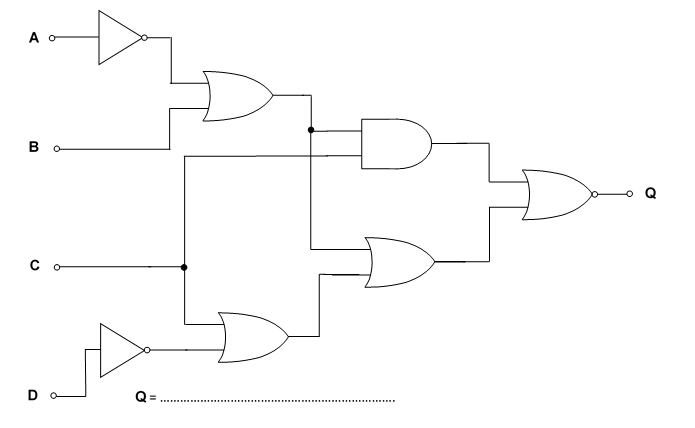
In stage 4 we combine these two large expressions with a NOR function. Again notice the use of brackets to keep terms together.

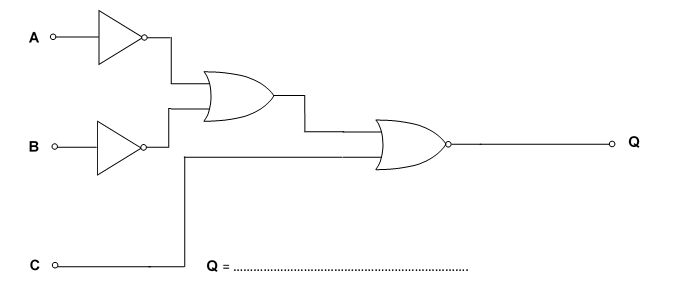


There is no easy way of learning how to complete these logic expressions from circuit diagrams other than through practice. So the following examples should help you practice the skills needed.

### Boolean Expressions from Circuits

1



2. 

3.

**A**

**B**

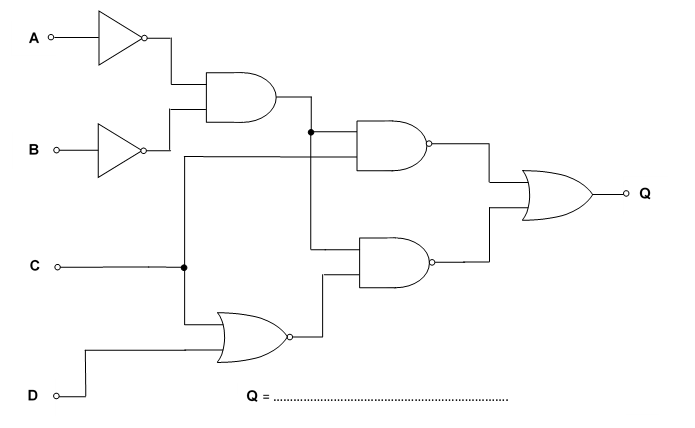
**C**

**Q**

**D**

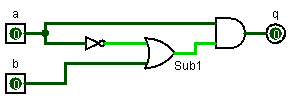
**Q** = …………………………………………………………….

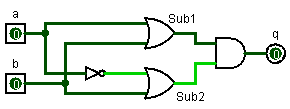
4.



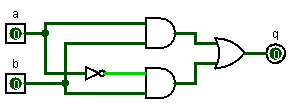
### Boolean Circuit to Algebra Practice

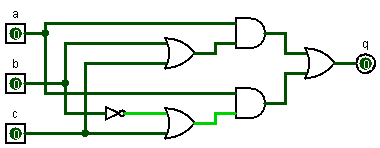
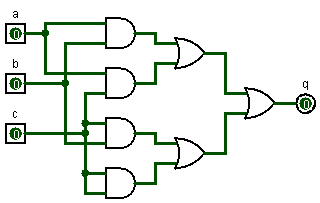
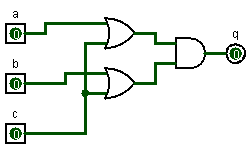
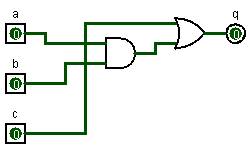
Write the Boolean Algebraic expression and a Truth Table for the following circuits. Where there are “SUB” points on the circuit you should write the Boolean expression that describes the output at that point. (only use logism to check your answers)

1. 

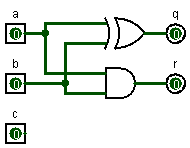
1. 

(adding sub points to help you is sensible.. from now on evaluate your own sub points… probably after every gate

1. 

1. 
2. 
3. 
4. 

A special circuits



### Circuit and truth table from an equation

For each of the following boolean statements use Logisim to help you write down the correct Boolean notation,   
Draw the circuit and construct the truth table:

A and (A or NOT B)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B |  | A+ | A.(A+) |
| 0 | 0 |  |  |  |
| 0 | 1 |  |  |  |
| 1 | 0 |  |  |  |
| 1 | 1 |  |  |  |

A AND B OR A AND C

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | A.B | A.C | (A.B)+(A.C) |
| 0 | 0 | 0 |  |  |  |
| 0 | 0 | 1 |  |  |  |
| 0 | 1 | 0 |  |  |  |
| 0 | 1 | 1 |  |  |  |
| 1 | 0 | 0 |  |  |  |
| 1 | 0 | 1 |  |  |  |
| 1 | 1 | 0 |  |  |  |
| 1 | 1 | 1 |  |  |  |

A AND B OR A AND NOT B

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

(A NAND B) or (A XOR B)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | | B |  |  |  |
| 0 | | 0 |  |  |  |
| 0 | | 1 |  |  |  |
| 1 | | 0 |  |  |  |
| 1 | | 1 |  |  |  |
| Boolean algebra: be familiar with the use of identities and De Morgan’s laws to manipulate and simplify simple Boolean expressions. | | | | | |

# Boolean Identities and Simplification

This is the same as numeric identities.. BASICALLY one side of the equation has an identical output to the other side…. BUT you may need to see them to believe them!!!

A, B, C, D are inputs

NOT() takes precedence over AND() takes precedence over OR(+)

Parenthesis () can be used to “force” precedence

|  |  |  |
| --- | --- | --- |
| Name  (you don't need to  remember the names) | Axiom |  |
| **Idempotent** | A+A=A AA=A |  |
| **Identity** | A+0=A A+1=1 |  |
| **Complement** | A+~A=1 ~(~A)=A |  |
| **Commutative** | A+B=B+A |  |
| **Associative** | (A+B)+C=A+(B+C)  A(BC)=(AB)C |  |
| **Distributive** | A•(B+C)=A•B+A•C  (A+B)(C+D)=AC + AD + BC + BD  A+BC=(A+B)(A+C) |  |
| **Absorption** | A+A•B=A A+~A•B=A+B |  |
| **De Morgan’s** | ~(A+B)=~A•~B ~(A•B)=~A+~B |  |

### Logism Identity proof

For each **side** of the Boolean equations above create a Logisim circuit. Either analyse the circuit to produce a Truth Table or work through the combinations. Copy the truth table from each side of the equation into a word document so you can see that the output is the same .

Just like numerical maths you can use identities to make expression shorter, simpler or maybe longer and clearer to see what is physically

## By Truth Table

To construct a truth table for a Boolean expression follow these steps:

1. Create a column for each input and fill-in all the possible inputs in binary numeric order
2. You should now have a table with 2n rows (where n is the number in inputs).
3. Look at the expression and think what precedence each operation takes…. (Brackets🡪 invert🡪AND🡪OR).
4. Create a column for each inversion of a single operand
5. Create a column for each AND operation
6. Create a column for each OR operation
7. If there are any NAND operations create a column for the AND then a second column for the inversion. (treat NOR the same way)
8. Now combine the different sections of the expression to create outputs for the truth table.
9. Finally look at the Final output and see if it is the same as a smaller simpler expression. (often this is one of the sub expressions you put in the table)

## Truth Table Simplification example

Let’s look at this expression

**A•(B+C•A)+~B**

1. column for each input (with 8 rows)

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

1. I’ll add the rows in at the end to save space
2. 🡪 7) Now work through the expression in order of precedence  
   Inversions

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | ~B |

AND (within the brackets)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | ~B | C•A |

OR (within the brackets)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | ~B | C•A | B+C•A |

AND (outside the brackets)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | C | ~B | C•A | B+C•A | A•(B+C•A) |

OR (outside the brackets)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | ~B | C•A | B+C•A | A•(B+C•A) | A•(B+C•A)+~B |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | ~B | C•A | B+C•A | A•(B+C•A) | A•(B+C•A)+~B |  |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |  |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |  |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |  |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |  |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |  |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |  |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |  |

SO…… The final pattern doesn’t match up to any single gate, it doesn’t match up to any part of the expression.. what do you do next???

Notice that the bottom 4bits are all 1 that is the same as input-A.. That suggests that the final simple expression is A OR “something”. Notice also that ~B has the same first 4 bits as the final pattern. To combine these to different part of the pattern is the most straightforward as it only requires the separate part to be linked by an OR gate… The simplest form of the above expression is: A + ~B

### Finish the Truth Table

Prove this for yourself now. Add to the final column “A + ~B” and complete the truth table.

### Truth table simplification practice

For each of the following functions, draw a truth table, and attempt to spot the simplification.

1. a.b+ a’.b
2. (a’+’b).(a’+b)
3. y.(x+z’)+x.z’

1. 
2. 
3. 

De Morgan's laws



|  |  |
| --- | --- |
|  | \overline{A+B}=\overline A \cdot \overline B  not (A or B) = (not A) and (not B) |
|  | \overline{A \cdot B}=\overline A + \overline B not (A and B) = (not A) or (not B) |

De Morgan's laws are based on the equivalent truth-values of each pair of statements

The rules are as follows:

1. If you break a ‘bar’ change the sign underneath the break.
2. If you complete a ‘bar’ change the sign underneath where the bar is joined. (We will use this in Topic 1.2.4)

1. If we start with , DeMorgan’s theorem suggests that this can be written as . The ‘bar’ has been broken and the sign changed underneath the break in the bar. We can check this by looking at the truth table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **0** | **0** | **1** | **1** |  |  |
| **0** | **1** | **1** | **0** |  |  |
| **1** | **0** | **0** | **1** |  |  |
| **1** | **1** | **0** | **0** |  |  |

This shows that the two logic expressions are the same.

2. If we start with , DeMorgan’s theorem suggests that this can be written as . The ‘bar’ has been broken and the sign changed underneath the break in the bar. We can check this by looking at the truth table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **0** | **0** | **1** | **1** |  |  |
| **0** | **1** | **1** | **0** |  |  |
| **1** | **0** | **0** | **1** |  |  |
| **1** | **1** | **0** | **0** |  |  |

This shows that the two logic expressions are the same.

The rule seems to work, according to the truth tables but we have only used very basic logic expressions here to prove the rule. If we looked at a more realistic problem then this would involve more terms and a more complex expression.

We will put a few of our new skills to the test by looking at a typical problem. You can work out some of the parts as we go through the example, but the correct solution will be provided at each stage so that you can check that you are on the correct path.

Consider the following logic circuit.

**A**

**B**

**C**

**Q**

**D**

**Q** = …………………………………………………………….

Use the process we looked at earlier to derive a Boolean expression for the logic circuit shown above.

Remember to be very careful with the NAND gates.

The Boolean expression you should have arrived at is as follows:



This looks quite complicated but by using DeMorgan’s theorem we can make this a little bit easier. As a general ‘rule of thumb’ start from the top and work downwards. So the first thing that we have to do is to break to top ‘bar’ between the two brackets. This gives us the following:



If we look at the first term we can see that there is a double ‘bar’ over the expression. This means that the term is inverted and then inverted again, which will return the original state of the term. Therefore the **double inversion** as it is called can be removed.



We now apply DeMorgan’s theorem again to the second term to give the following:



Again we have a double inversion, applied only to the variable **C**, which can be removed to leave the final expression as:



If there were additional terms in the expression, this could now be simplified using the normal rules of Boolean algebra as discussed previously.

Now for a couple of examples for you to try!

### Simplify the following expressions as much as possible.

1. 

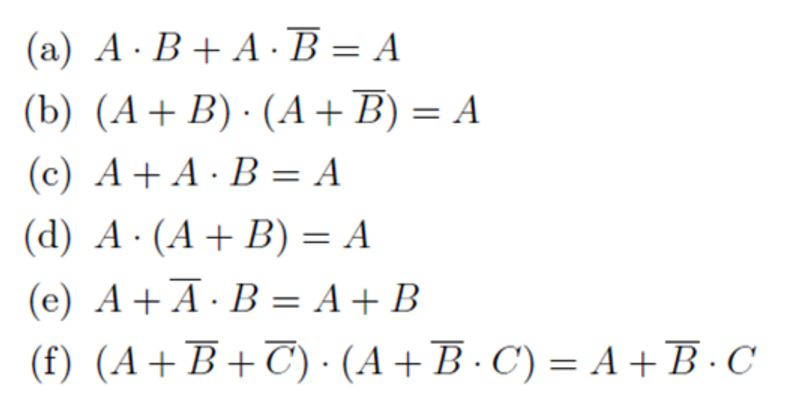
2. 

### 

### Truth Table Proofs

The same process as simplification BUT the final column has been given.

For each of the following create a truth table. You will need To put your tables into a HeAD and upload it to the Assignment on Godalming Online. You should use Excel or Word etc.. to create the tables



### Algebraic Boolean simplification practice 1

Using the Boolean Identities in Appendix A. Simplify the following Boolean expressions using Algebra.

1. 

2. 

3. 

4. 

5. 

6. 

7. 

8. 

### Draw the Logic Circuit diagram for the Boolean expressions given.

1. 

2. 

### More Boolean simplification Practice 2

Using the Boolean Identities in Appendix A. Simplify the following Boolean expressions using Algebra. The first few are just one line identities, they get gradually trickier throughout.

**Basic**



**Slightly Tricky**



**EVIL (treat as Extension)**



# Appendix A



### 