ICS 6B
Boolean Algebra & Logic

Lecture Notes for Summer Quarter, 2008

Michele Rousseau

Set 9 – Ch. 11.1, 11.2, 11.3
(Some slides inspired and adapted from Alessandra Pantano)

Announcements

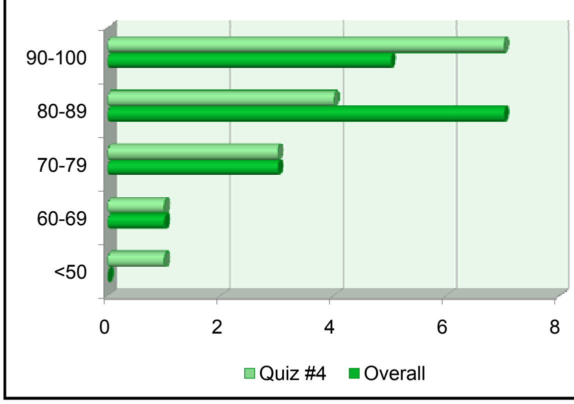
- Regrades for everything returned today are due on Thursday
- Quiz #5 will Lecture set 8 and 9

Quiz & Overall grades

Quiz #4
- Max: 98%
- Min: 48%
- Avg: 84%
- Median: 86%

Overall Scores
- Max: 98%
- Min: 69%
- Avg: 80%
- Median: 86%

Quiz #4 & Overall



Today's Lecture

- Chapter 11 (11.1, 11.2, 11.3)
  - Boolean Functions (11.1)
  - Representing Boolean Functions (11.2)
  - Logic Gates (11.3)

Chapter 11: Section 11.1

Boolean Functions

Order of Operations

- In order to evaluate these we need to understand the order of preference
  - Things in Parenthesis come first
  - Then
    - 1 – Complement
    - 2 – Product
    - 3 – Sum

Boolean Functions Examples

2-degree Boolean functions

$F(x,y) = \neg(x+y) \cdot (xy) + \neg x + \neg(xy)$

$F(x,y)=[(x+y)+y]+\neg xy$

Example

$F(x,y)= [(x+y)+y] + \neg xy$

Let $x=0,\ y=1$

$[(\neg 0+1)+1] + \neg 0 \cdot 1$

Do the complement before the product

Do the complement before the sum

$= [(1+1)+1] + 1 \cdot 1$

$= [1+1] + 1 = 1 + 1 = 1$

Similarly,

$1 \cdot 0 + \neg(0+1)$

$= 1 \cdot 0 + \neg 1 = 0 + 0 \ = 0$

More Examples

- Compute the values of the Boolean function
  $F(x,y) = x + \neg y$

➔ we use a Table with columns $x,\ y,\ \neg y,$ $x + \neg y = F(x,y)$

It is convenient to have this intermediate step

x	y	¬ y	F(x,y)=x+¬ y
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1

Tables like this are very convenient

More Examples

Compute $F(x,y) = x \cdot \neg y + \neg(x+y)$

x	y	¬y	x·¬y	x+y	¬(x+y)	F(x,y)=x·¬y+¬(x+y)
0	0	1	0	0	1	1
0	1	0	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	1	0	0

+

Boolean Function of > 2 degrees

Boolean functions of $n \geq 3$ are defined similarly.

A function of degree n is a map

$F: \{0,1\} \times \{0,1\} \times \ldots \times \{0,1\} \longrightarrow \{0,1\}$

n times

Notice that the domain of the function are bit strings of length n: $x_1 x_2 \ldots x_n$, with $x_i = 1$ or 0.

For each such string, $F(x_1, x_2, \ldots, x_n)$ can be 0 or 1

Example

$F(x,y,z) = \neg x + yz$ is a Boolean function of degree 3

To compute F, you need to find the value of F on each of the 2^n strings (where n=degree)

in this case is n=3 so we need to find the value of

$2^3 = 8$ strings – so the table F will have 8 rows

x	y	z
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

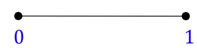
Drawing Boolean Functions (of degree ≤3)

Degree 1

If F is a function of 1 Boolean variable, (i.e. F=F(x)) then F is completely determined by F(0) and F(1).

Notice that F(0) and F(1) can only be 1 or 0, so F is uniquely determined by saying whether F takes the value 1 or 0, on 1 or 0 on either of them.

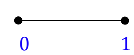
Draw the length 1 strings 0 and 1 as vertices of a segment



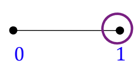
Drawing Boolean Functions of degree 1

Now circle the vertices on which F takes the value 1.

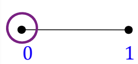
We have 4 possibilities:



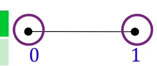
x	F(x)
1	0
0	0



x	F(x)
1	1
0	0



x	F(x)
1	0
0	1



x	F(x)
1	1
0	1

Notice that we have established a one-to-one correspondence between Boolean function of degree 1 and the subsets of the set of strings of length 1.

There are 2 strings of length 1, so there are 2^2 subsets and indeed we find 4 Boolean functions

Boolean Functions of degree 2

A Boolean function of degree 2 is a function of 2 Boolean variables x,y with values in {0,1}.

You can think of x,y as a bit-string of length 2.

Both x and y can take 2 values (0 and 1), so we have $2 \cdot 2 = 2^2 = 4$ strings of length 2.

So, the total number of choices for F is

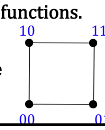
# of choices for value of F on 2nd string

$$\underbrace{2\cdot 2 \cdot \ldots \cdot 2}_{2^2 \text{ times}} = 2^{2^2} = 2^4 = 16$$

2^2 times -- Total # of strings

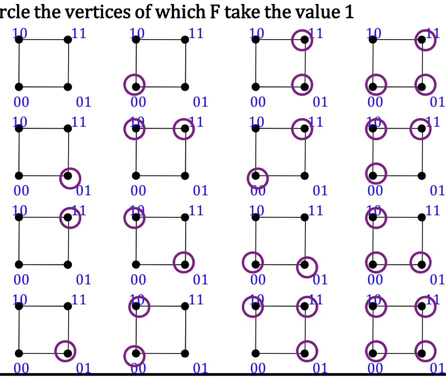
We have a total of 16 possible Boolean functions.

The 4 strings (00, 01, 10, 11) can be represented as the vertices of a square



Drawing Boolean Functions – degree 2

We circle the vertices of which F take the value 1

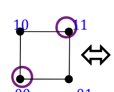


Boolean Functions of degree 2

It is easy to write down the corresponding function. If the vertex corresponds to a string x,y, then

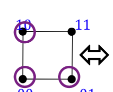
$$F(x,y) = \begin{cases} 1 \text{ – if the vertex is circled} \\ 0 \text{ – if the vertex is NOT circled} \end{cases}$$

Example



⇔

x	y	F(x,y)
0	0	1
0	1	0
1	0	0
1	1	1



⇔

x	y	F(x,y)
0	0	1
0	1	1
1	0	1
1	1	0

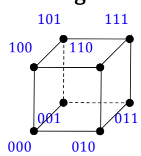
Drawing Boolean Functions of degree 3

F is defined on strings of length 3.

There are 2^3=8 such strings:

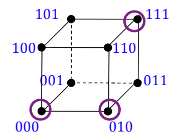
(000, 001, 010, 011, 100, 101, 110, 111)

We represent the strings as vertices on a cube



Example

To assign F, we decide on which strings F takes the value 1 and we circle the corresponding vertices



⇔

x	y	z	F(x,y,z)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

We have one-to-one correspondence between subsets of vertices and Boolean functions.

n=3 (degree 3)

→ we have $2^n = 8$ strings

→ $2^n = 8$ vertices

⇔ $2^{2^n} = 2^8$ subsets of vertices

⇔ $2^{2^n} = 2^8$ Boolean functions

General Fact:

There are $2^{(2^n)}$ Boolean functions of degree n.

[Because there are 2^n strings of length n, and you can assign 2 values to each string. This gives you $\underbrace{2\cdot 2\cdot \ldots \cdot 2}_{2^n \text{ times}}$ choices, (i.e. 2^{2^n} Choices)

Important Remark:

A function can be represented by many different expressions.

E.g. F(x)=x; G(x)=x+0; H(x)= ¬ (¬x)

are all the *same function* (even if they are given by *different expressions*).

Indeed they take the same value on every string

x	F(x)=x
1	1
0	0

=

x	G(x)=x+0
1	1
0	0

=

x	¬x	H(x)= ¬ (¬x)
1	0	1
0	1	0

So when we say that F and G are equal, we mean that they take the same value on **every** string.

We do **NOT** require that they are defined by the same expression

Example

Show that F(x) = x + x & G(x)=x are the same Boolean function.

We compute the tables and compare the values

x	F(x)=x + x
1	1 + 1 = 1
0	0 + 0 = 0

x	G(x)=x
1	1
0	0

=

We call **x+x = x** a ***Boolean identity***

We also say that (x + x) and (x) are equivalent expressions.

Boolean Identities

Identity	*Name*
$\bar{\bar{x}} = x$	Law of the double complement
$x + x = x$ $x \cdot x = x$	Idempotent laws
$x + 0 = x$ $x \cdot 1 = x$	Identity laws
$x + 1 = 1$ $x \cdot 0 = 0$	Domination laws
$x + y = y + x$ $xy = yx$	Commutative laws
$x + (y + z) = (x + y) + z$ $x(yz) = (xy)x$	Associative laws

Boolean Identities (2)

Identity	*Name*
$x + yz = (x + y)(x + z)$ $x(y + z) = xy + xz$	Distributive laws
$\overline{(xy)} = \bar{x} + \bar{y}$ $\overline{(x + y)} = \bar{x}\,\bar{y}$	De Morgan's laws
$x + xy = x$ $x(x + y) = x$	Absorption laws
$x + \bar{x} = 1$	Unit property
$x\bar{x} = 0$	Zero property

Example

Verify the DeMorgan's law ¬(xy) = ¬x + ¬y

We prove that F(x,y)= ¬(xy) & F(x,y) = ¬x+ ¬y are the same function

x	y	xy	F(xy)=¬(xy)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

x	y	¬x	¬y	F(xy)=¬x +¬y
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

=

Note: Another example of the different expressions of the same function

Duality

Suppose you have an expression involving Boolean variables.

(i.e. [x + ¬y +z]+(1+x))

The dual expression is obtained by

- Replacing every 1 with a 0 (and every 0 with a 1)
- Replacing every + by · (and every · by +)

So you interchange 1's & 0's, and sums & products

Examples:

Switch 1 and 0

Expression	Dual Expression
(x + y) +1	(xy) · 0
(x · ¬y) + y	(x + ¬y) · y
x (y + 0)	x + (y · 1)
[x + ¬y + z]+(1+x)	[x · ¬y · z] ·(0· x)

Switch + and ·

Remark:

If F is given by a certain expression, and G is given by the dual expression, then the values of F are exactly the complements of the values of G.

Example: F(x) = (x+y)+(¬y · 1); G(x)=(xy) · (¬y+0)

x	y	x + y	¬y	¬y·1	F(x)
0	0	0	1	1	1
0	1	1	0	0	1
1	0	1	1	1	1
1	1	1	0	0	1

x	y	x y	¬y	¬y+0	G(x)
0	0	0	1	1	0
0	1	0	0	0	0
1	0	0	1	1	0
1	1	1	0	0	0

Complement Values

Homework Section 11.1

- 1(a-d)
- 3(a-d)
- 5(c,d)
- 7(for 5c,d)
- 10
- 13
- 15
- 17

Chapter 11: Section 11.2

Representing Boolean Functions

Representing Boolean Functions

Problem: Given the values of F, find an expression for F.

e.g.

x	y	F(xy)
0	0	0
0	1	0
1	0	0
1	1	1

F takes the value 1 only on the string 1,1

i.e. $F(x,y)=1 \Leftrightarrow x=1$ AND $y=1$

$\Leftrightarrow xy=1$

Choose $F(x,y)=xy$

Recall the relationship between AND & ·

More Examples

Problem: Given the values of F, find an expression for F.

x	y	F(xy)
0	0	0
0	1	0
1	0	1
1	1	0

F takes the value 1 only on the string 1,0

i.e. $F(x,y)=1 \Leftrightarrow x=1$ AND $y=0$

$\Leftrightarrow x\neg y=1$

Choose $F(x,y)=x\neg y$

x	y	F(xy)
0	0	1
0	1	0
1	0	0
1	1	0

$F(x,y)=1 \Leftrightarrow x=0$ AND $y=0$

$\Leftrightarrow \neg x \neg y = 1$

Choose $F(x,y)= \neg x \neg y$

Another Example

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$F(x,y,z)=1 \Leftrightarrow x=0, y=1, z=1$

$\Leftrightarrow \neg x=y=z=1$

Choose $F(x,y)=\neg x\cdot y\cdot z$

In General..

If $F(x_1, x_2, ..., x_n)=1$ on a single string then F can be represented as a "*minterm*"

(i.e. a product $y_1 y_2 \ldots y_n$ where each y_i is either $= x_i$ or $=\neg x_i$.)

More precisely, if the string is 011…10…1

Then, look at :

The i^{th} entry (which is either 0 or 1)

If the i^{th} entry is 1, chose $y_i = x_i$

If the i^{th} entry is 0, chose $y_i = \neg x_i$

For example:

If F=0 except on 10010 then $F(x_1, x_2, x_3, x_4, x_5) =1$

$\Leftrightarrow x_1=1, x_2=0, x_3=0, x_4=1, x_5=0$

$\Leftrightarrow x_1\cdot \neg x_2 \cdot\neg x_3\cdot x_4 \cdot \neg x_5=1$

Proof

$F(x_1, x_2, x_3, x_4, x_5) =1$

$\Leftrightarrow (x_1, x_2, x_3, x_4, x_5)=10010$

$\Leftrightarrow (x_1=1, x_2=0, x_3=0, x_4=1, x_5=0)$

$\Leftrightarrow x_1=\neg x_2 =\neg x_3= x_4 = \neg x_5=1$

$\Leftrightarrow x_1\cdot \neg x_2 \cdot\neg x_3\cdot x_4 \cdot \neg x_5=1$

$\Rightarrow$ You can choose

$F(x_1, x_2, x_3, x_4, x_5) = x_1\cdot \neg x_2 \cdot\neg x_3\cdot x_4 \cdot \neg x_5$

Q.E.D

What about this function?

x	y	F(xy)
0	0	0
0	1	1
1	0	1
1	1	0

We see that $F(x,y) =1$

$\Leftrightarrow x=0, y=1$ OR $x=1, y=0$

$\Leftrightarrow \neg x=y=1$ OR $x= \neg y=1$

$\Leftrightarrow \neg xy=1$ OR $x \neg y=1$

$\Leftrightarrow \neg xy + x \neg y$

So we can choose $F(x,y)= \neg xy + x \neg y$

In General, give the values of F, you can:

1. Write down all the strings on which F=1
2. each string gives you a "minterm" like before
3. take the sum of these minterms

=> the result is an expression for F

Example

x	y	z	F(xy)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

1- Isolate the strings on which F=1
   000; 010; 011; 100
2- Get the minterm corresponding to each string:
   ¬x¬y¬z ; ¬xy¬z ; ¬xyz; x¬y¬z
3- Take the sum to obtain an expression for F:

F(x,y,z)= ¬x¬y¬z + ¬xy¬z + ¬xyz + xy¬z

Sum of Products Expansion

Suppose that F is represented by a sum of minterms, then we say F has a sum of products expansion.

Problem: Find the sum of products expansion of the function F(x,y,z) = (x+y) ¬z

x	y	z	x + y	¬z	F(x,y,z)=(x+y) ¬z
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	0

We follow these steps

1. Find the table values of F
2. Find the sum of products expansion for F

We notice that F(x,y,z)=1

⇔ ¬ x=y= ¬ z OR x= ¬y= ¬z=1 OR x=y= ¬z=1 ⇔

F(x,y,z)= ¬x y ¬z + x¬ y ¬z + x y ¬z

Examples

Problem: Let F be a function s.t. F(x,y,z)=1 ⇔ xy=0
Find the sum of products expansion of F.

By hypothesis, F(x,y,z)=1⇔ x=0 OR y=0

So the strings on which F=1 are

000, 001, 010, 011, 100, 101

To each of them we associate a minterm:

¬x ¬ y ¬ z ; ¬x ¬ yz ; ¬x y ¬ z ; ¬xy z; x ¬y ¬z; x ¬yz

F(x,y,z)=
¬x ¬ y ¬ z + ¬x ¬ yz +¬x y ¬ z +¬xy z+ x ¬y ¬z + x ¬yz

Functional Completeness

Using the sum of products expansion we can write any Boolean function F as a sum of minterms.

So, every Boolean function can be represented by using the Boolean operators +, ·, ¬

Thus, we say that the set of operations
{+, ·, ¬} is *functionally complete*

Can we find a smaller set of operations that are functionally complete?

Yes = Using DeMorgan's Law

- Because it relates all 3 operations (+, ·, &¬)
- ¬(x ·y) = ¬x+¬y
- So, by the double complement law xy = ¬(¬x +¬y)
- We can replace ever occurrence of the product by a combination of + and complement.
- The result is an expression of F that only involves + &¬. Thus {+, ¬} is a functionally complete set of operations.
- Similarly we can use DeMorgan's Law to replace every occurrence of + by a combination of ·&¬.
- Thus {·,¬} is also a functionally complete set of operations

Can we find an even smaller set of operations that is functionally complete?

Yes. Consider the NAND operator (denoted by |) and defined by
1|1 =0 &
1|0 = 0|1 = 0|0=1

x	y	xy	¬(xy)	x|y
0	0	0	1	1
0	1	0	1	1
1	0	0	1	1
1	1	1	0	0

One can see that x|y = ¬(xy)

Then both the product and the complement can be expressed in terms of NAND:

¬(x·x)=x|x ⇔ ¬x=x|x
xy=¬(¬(xy))
=¬(x|y)
= (x|y)|(x|y)

Because {¬,·} is functionally complete so is {NAND} alone.

NOR

Similarly

NOR (denoted ↓) defined by

1 ↓ 1 = 1 ↓ 0 = 0 ↓ 1 = 0 and

0 ↓ 0 = 1

is also functionally complete.

x	y	x+y	¬(x+y)	x↓y
0	0	0	1	1
0	1	1	0	0
1	0	1	0	0
1	1	1	0	0

Notice that ¬(x+y) = x ↓ y

So, ¬(x+x) = x ↓ x ⇔ ¬x = x ↓ x and

x+y = ¬ (¬ (x+y))

= ¬ (x ↓ y)

= (x ↓ y) ↓ (x ↓ y)

Because {¬,+} is functionally complete so is {NOR} alone.

Homework Section 11.2

- 1(a,c),
- 3(a-d),
- 5,
- 12(c,d),
- 13(c,d),
- 14(a),
- 15(a,b)

Chapter 11: Section 11.3

Logic Gates

Logic Gates

Boolean Algebra can be used to model circuits of electronic devices.

- I/O of circuits are Boolean variables.
- A circuit can be realized as a combo of gates.
- Each gate represents an operation on Boolean variables.

Suppose we have a "combinational circuit",

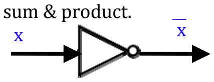
i.e. a circuit with no memory capability

- In other words: the output depends only on the input, not on the current state of the circuit.

Then we have 3 basic types of gates

- Corresponding to the 3 basic operations: complement, sum & product.

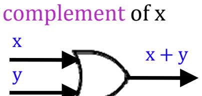
1 – The inverter



Input: single Boolean variable x

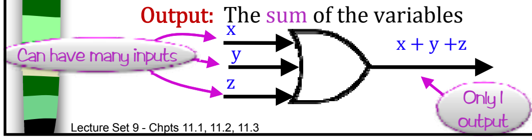
Output: The complement of x

2- The "OR" Gate



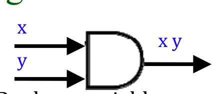
Input: 2 or more Boolean variables

Output: The sum of the variables



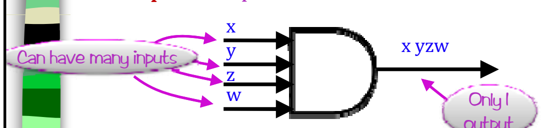
Circuits and Logic Gate

3- The "AND" Gate



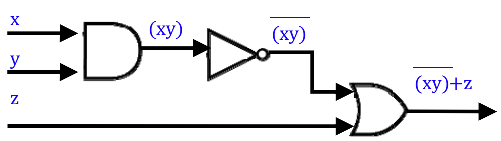
Input: 2 or more Boolean variables

Output: The product of the variables

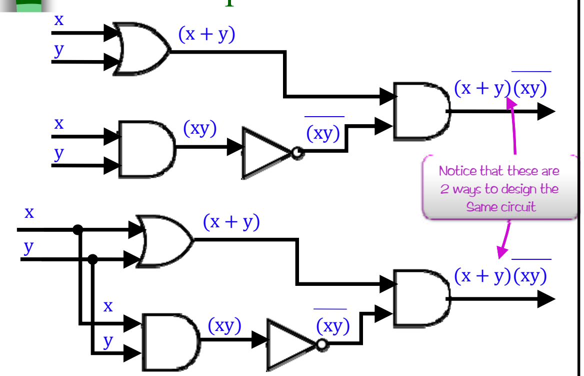


A circuit is a combination of 2 or more of these ***3 gates.***
Each combinational circuit represents a Boolean function
(i.e. a combo of +, ·, ¬ on a set of Boolean variables)

Example

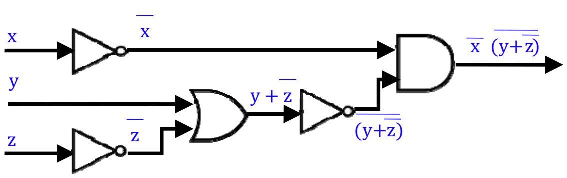


More Examples



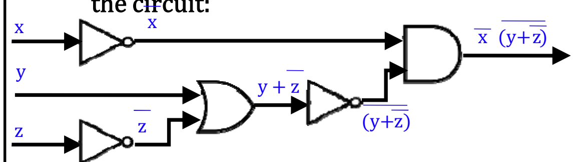
Problems

Find the Boolean function represented by the circuit:



Problem 1: Find a Boolean function from a circuit

Find the Boolean function represented by the circuit:

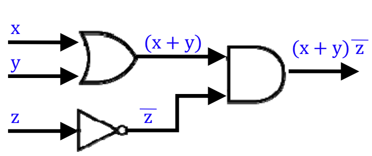


So this circuit represents the boolean function

F(x,y,z)= $\overline{x}\,\overline{(y+z)}$

Problem #2: Find a circuit from a Boolean function

Find the circuit that represents the Boolean function F(x,y,z)=(x+y) $\overline{z}$



Problem 3: Find a circuit that performs a given function

Suppose you have a majority voting system with 3 individuals. Design a circuit that determines whether a proposal passes.

Let x,y,z be the votes of 3 people. The proposal passes if at least 2 people vote yes.

So, F(x,y,z)=1 if x=y=1 or x=z=1 or y=z=1.

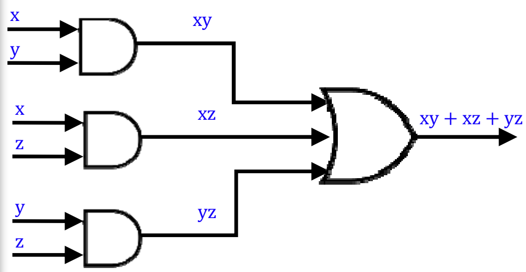
Equivalently, F(x,y,z=1) if xy=1 or xz=1 or yz=1

Then we can represent F by

F(x,y,z)= xy+yz+xz

Problem #3

Draw a circuit that produces $F(x,y,z) = xy + xz + yz$



Problem #4 - Adders

Our next task consists of designing a circuit that adds up two numbers written in base 2

Recall how to add base 2 numbers:

$0 + 0 = 0$

$0 + 1 = 1 + 0 = 1$

$1 + 1 = 1\,0$

Notice you have to carry the 1

We want to find $x + y$

So the inputs will be x and y with a value of 0 or 1

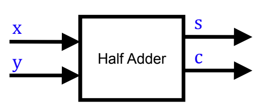
The output will consist of two bits s and c

- s is the sum bit and
- c is the carry

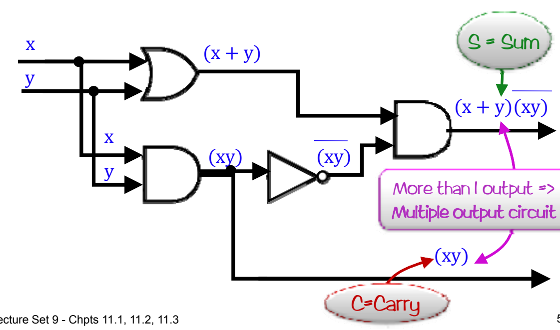
Designing an Adder (2)

First we'll design a half-adder – then we'll move onto the full adder

It is called a Half adder because it doesn't take a carry as input



Problem #4 - Half-Adder(3)



Full Adder

Now we want to add two 3-bit numbers.

Our inputs are $(x_0\, x_1\, x_2)$ and $(y_0\, y_1\, y_2)$

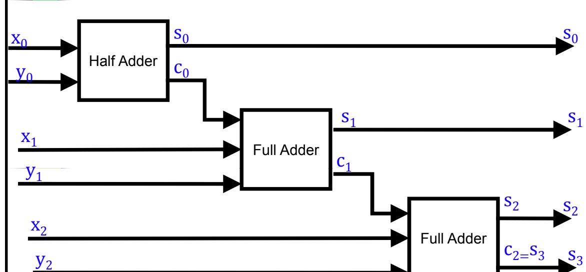
(e.g. could be 111 + 010)

We have to construct a full adder which takes the carry bits into consideration

The full adder will have the 3 input bits:

x_i, y_i, c_i

Problem #4 – Full Adder



Notice that the outputs are: s_0, s_1, s_2, c_2 -- $c_2 = s_3$, so $(s_3\, s_2\, s_1\, s_0)$ is exactly the binary expression of $(x+y)$, thus the circuit performs as desired

Homework Section 11.3

- 1
- 3
- 5
- 7
- 17