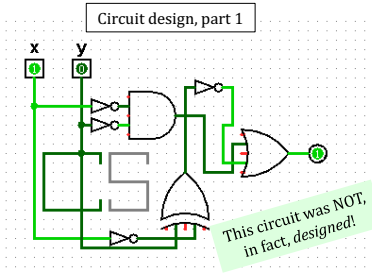
OK Recurring?
OKer than before?

Hw: Circuit design!

Circuit design, part 1



Professor Disappears!

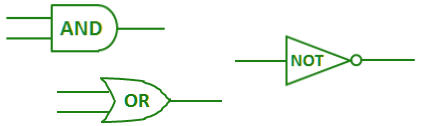
Claremont (AP) —An obscure computer science professor failed to show up for classes last week. Some sources suggest that he was present but invisible, but others alleged that he had decamped to Oakland, California. Still others felt that it was unlikely that *anybody* would voluntarily visit Oakland, and were convinced that he had been abducted by three-eyed aliens who probed him (continued on p. 42)

CS ||| Today

Minterm Expansion Principle

That's *minterm*, **NOT** *midterm*

A circuit for ***any function*** can be built from...

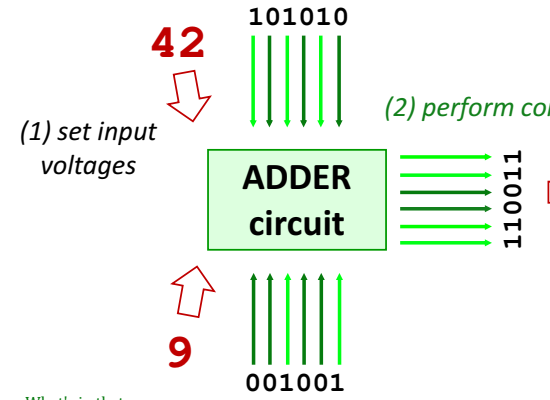


...***just*** these three logic gates!

The big picture...

In a computer, each bit is represented as a voltage (**1** is +3v and **0** is 0v)

Computation is simply the **deliberate combination** of those voltages!



What's in that green box?

Richard Feynman: *"Computation is just a physics experiment that always works!"*

Part 1: Represent your function as ***bits***...

Any function can be represented using only bits...

IN			OUT
x	y	c	circuit output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

This one is named the **sum** function

That's some function, all right!

three bits in...

...one bit out

Our building blocks: ***logic gates***

input		output
x	y	AND (x, y)
0	0	0
0	1	0
1	0	0
1	1	1

AND outputs 1 only if **ALL** inputs are 1

AND

input		output
x	y	OR (x, y)
0	0	0
0	1	1
1	0	1
1	1	1

OR outputs 1 if **ANY** input is 1

OR

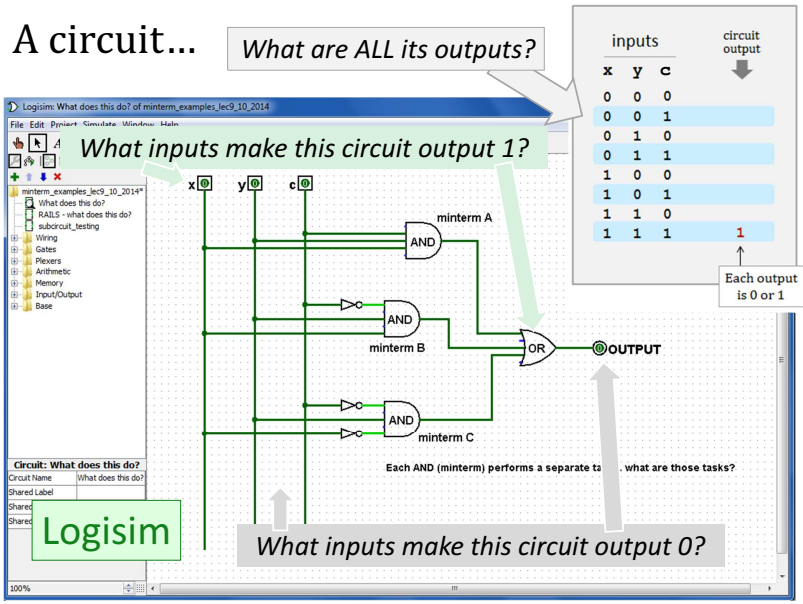
input	output
x	NOT (x)
0	1
1	0

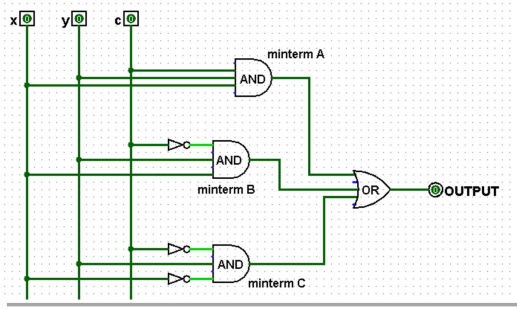
NOT reverses its input

NOT

A circuit…

What are ALL its outputs?

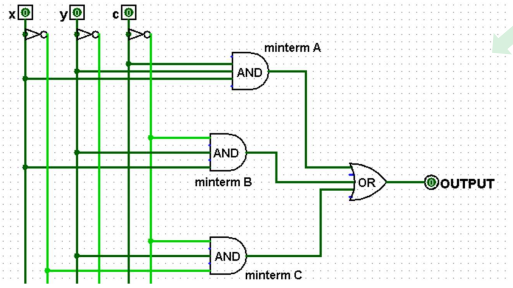




Rails

There is NO difference between these two circuits!

How?



using *rails* for **not x**, **not y**, **not c**

Any advantages of this "rails" approach?

Any disadvantages?

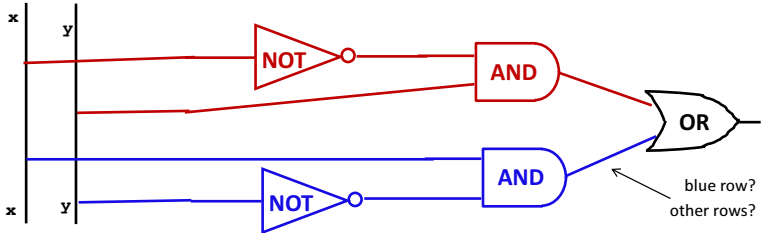
A constructive proof…

(i) Specify a **truth table** defining ***any*** function you want

input		output
x	**y**	**f(x,y)**
0	0	0
0	1	1
1	0	1
1	1	0

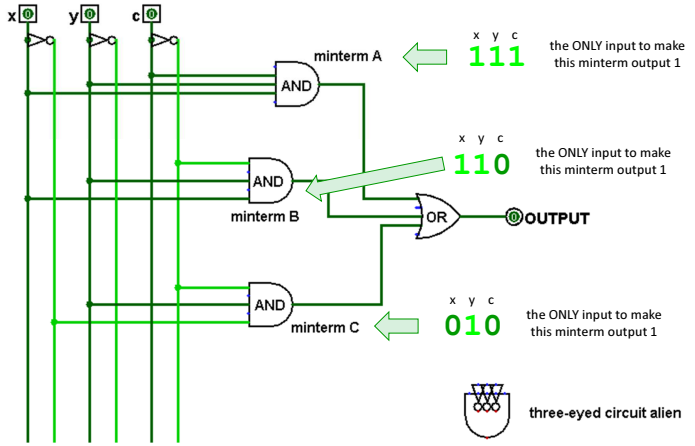
(ii) For each input row whose output needs to be 1, build an **AND** circuit that outputs 1 *only for that specific input*!

(iii) **OR** them all together



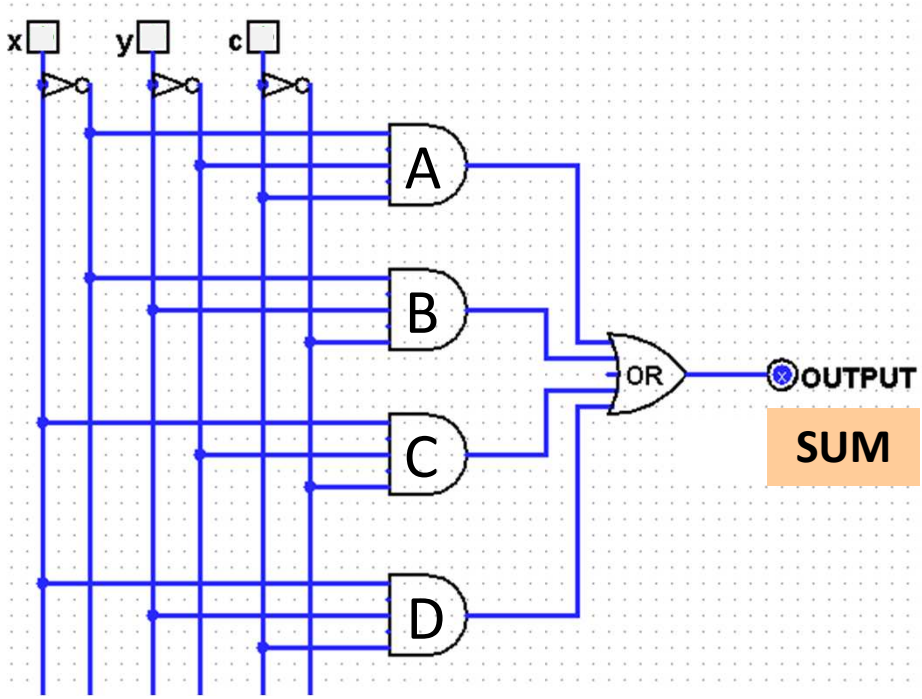
Each minterm selects ***one*** input:

a ***minterm*** is an AND gate that "***selects***" a single input row



Minterm Expansion Principle

Looks a little *wiry* to me!



input			output
x	y	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	

Take 2…

(1) Fill in the function values (the truth table) for this circuit…

Hint: Determine the input that turns each AND gate—each *minterm*—to **`True`**

(2) Draw the **upstream** wires that will implement this function as a circuit.

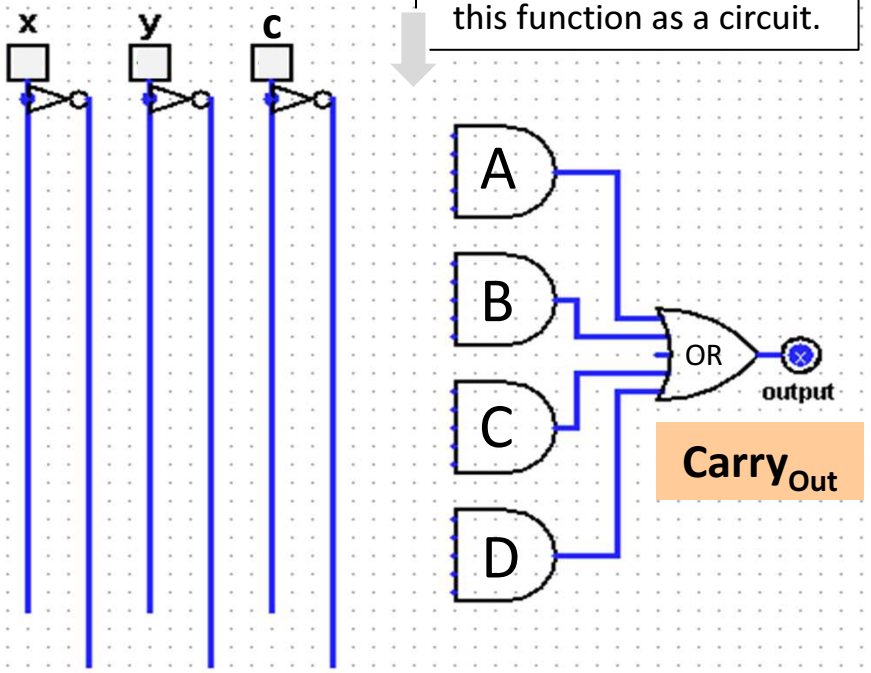
(Extra #1) Any gates you can optimize away here?

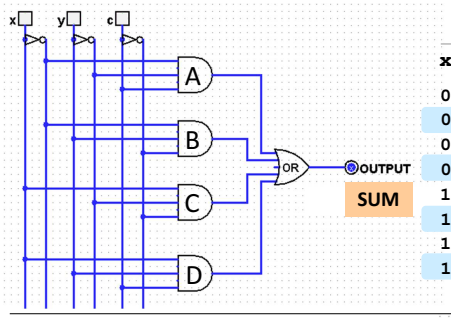
(Extra #2) How could you replace the OR with only ANDs and NOTs – ***so ORs aren't needed*** *!?*

(Extra #3) How do the two circuits on this page implement ***addition of any two binary #s?***

```
  111
 1011  x
+1111  y
-----
11010
```

input			output
x	y	c	
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





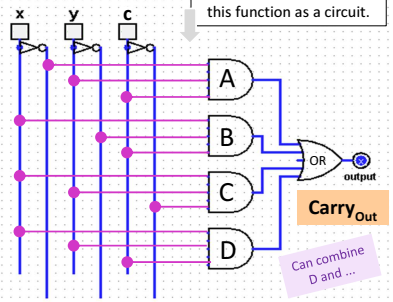
input				output
x	y	c		
0	0	0		0
0	0	1	A	1
0	1	0	B	1
0	1	1		0
1	0	0	C	1
1	0	1		0
1	1	0		0
1	1	1	D	1

Take 2…

(1) Fill in the function values (the truth table) for this circuit…

Hint: Determine the input that turns each AND gate—each *minterm*—to `True`

(2) Draw the **upstream** wires that will implement this function as a circuit.



(Extra #1) Any gates you can optimize away here?

(Extra #2) How could you replace the OR with only ANDs and NOTs – ***so ORs aren't needed*** *!?*

(Extra #3) How do the two circuits on this page implement ***addition of any two binary #s!?***

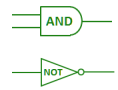
```
  111
 1011  x
+1111  y
-----
11010
```

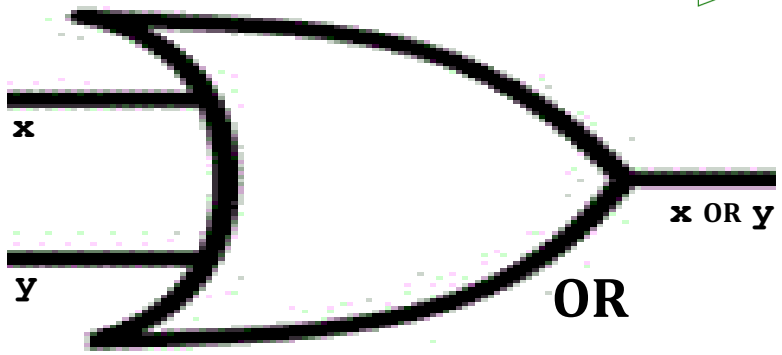
input				output
x	y	c		
0	0	0		0
0	0	1		0
0	1	0		0
0	1	1	A	1
1	0	0		0
1	0	1	B	1
1	1	0	C	1
1	1	1	D	1

input		output
x	y	OR(x,y)
0	0	0
0	1	1
1	0	1
1	1	1

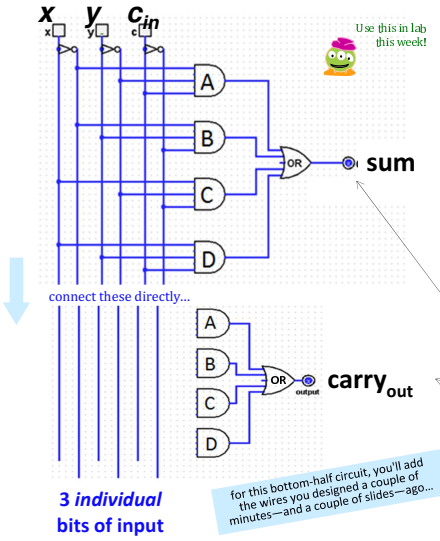
OR else ?!

Can you get rid of ORs by using only NOTs and ANDs?

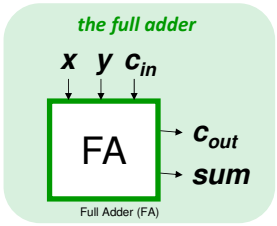




Lab 6: *adders!*

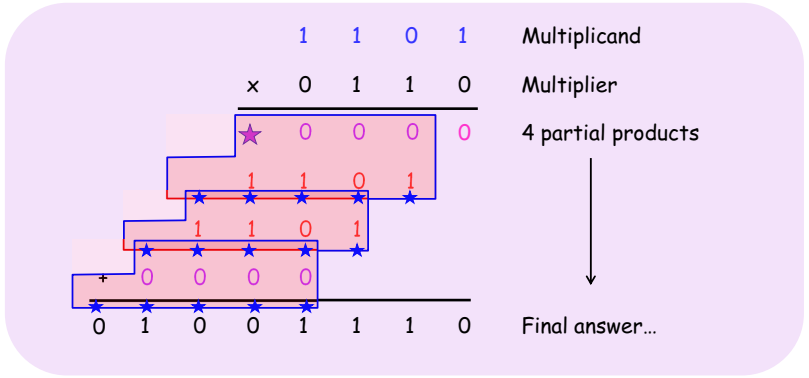


A ***full adder*** sums three input bits to create *a 2-bit* ***binary*** *output*



2 bits of output: a two-bit binary #

hw6pr2: A 4-bit *multiplier*



(A3) Remember that the AND gate is ***single-bit*** multiplication.

(A2) Use a 4x1-bit helper circuit to find the four *partial products…*

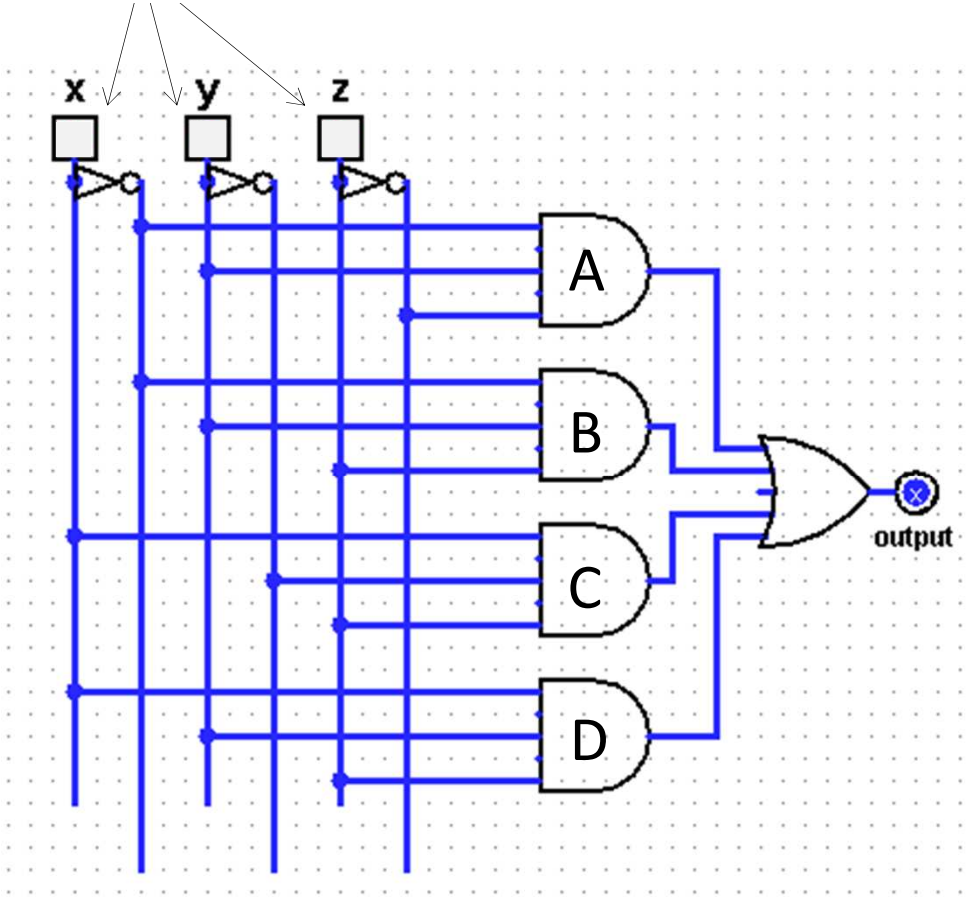
(A1) You need three ripple-carry adders to finish: *see above…*

Quiz

Name(s) _____

Fill in the function values for this circuit (the truth table)

Each input x, y, and z can ***independently*** be 0 or 1, for *eight* total possible inputs:



inputs			circuit output
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	1

Each output is 0 or 1

(1) This circuit uses 8 logic gates – *how many of each?* **AND ___ OR ___ NOT ___**

(2) Follow the inputs of x=1, y=1, z=1 and see why the overall output is 1 ... (already labeled)

(3) For ***each*** possible input, write the circuit output in the truth table above.

(4) Could this circuit use ***fewer*** logic gates? ***If so, how?!*** ***If not, how do you know?!***

This circuit is prime!