CIRCUITS

David Kauchak
CS 51 – Spring 2026

Admin

Assignment 3

Check autograder results after submitting assignment

Examples

The Logisim circuit examples can be found at:

http://www.cs.pomona.edu/classes/cs51/circuits/

Inside a CPU

Gates

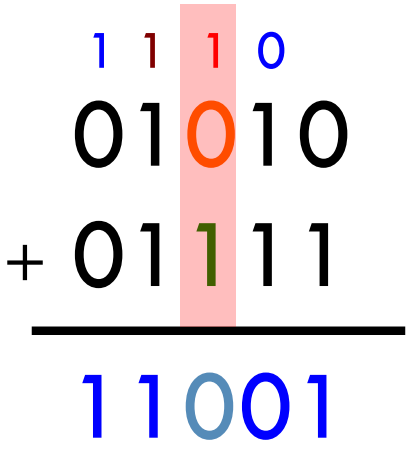
Quick recap

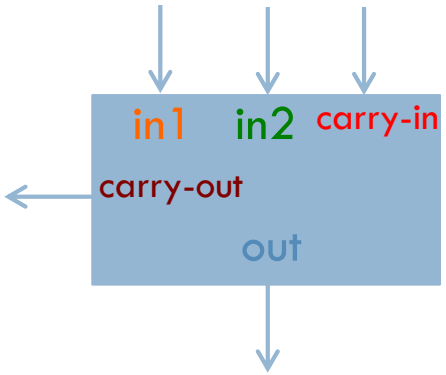
$$\begin{array}{r} 01010 \\ +\ 01111 \\ \hline \end{array}$$

Quick recap

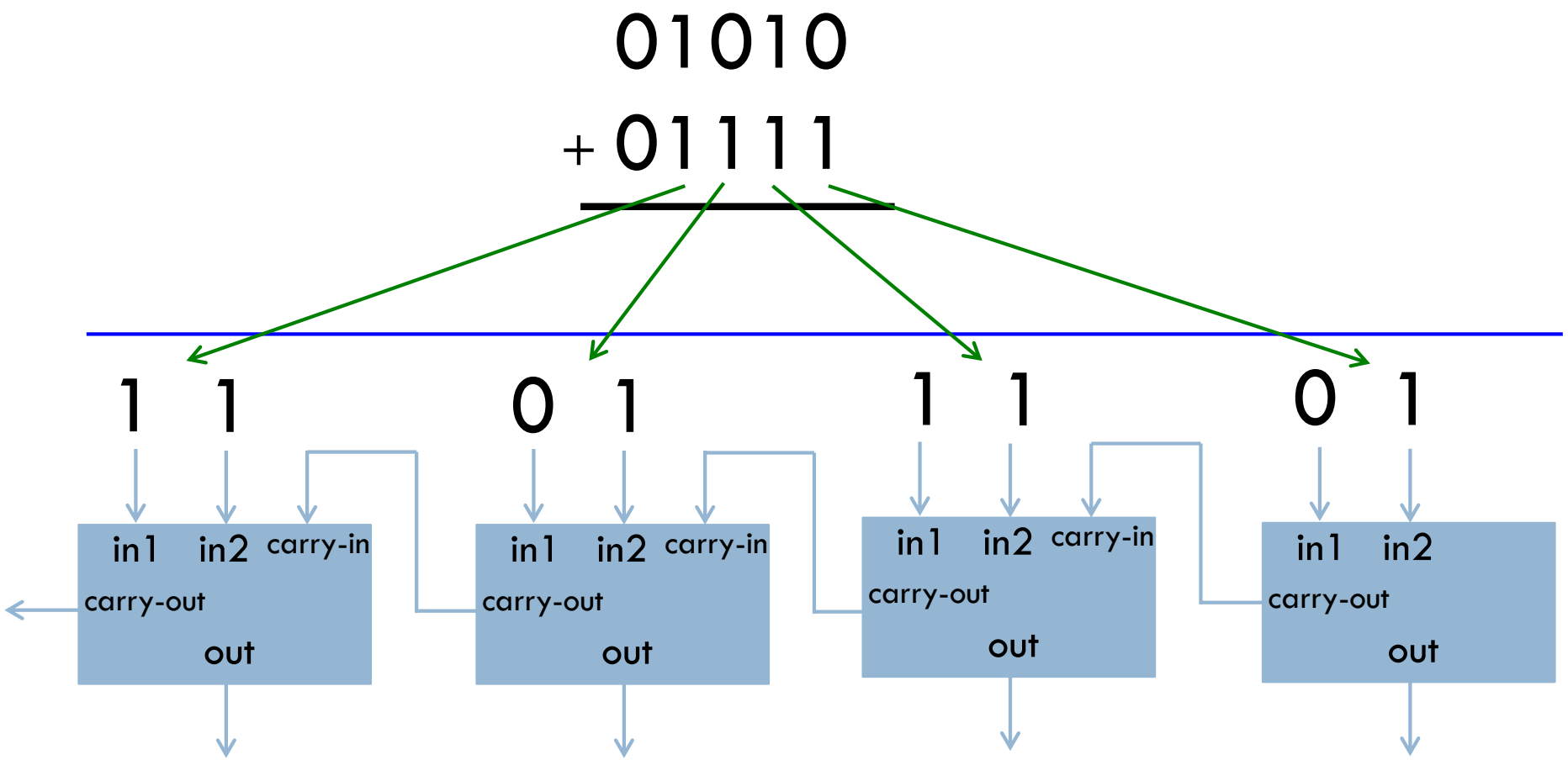
```
  1 1 1 0
  01010
+ 01111
-------
  11001
```

A component



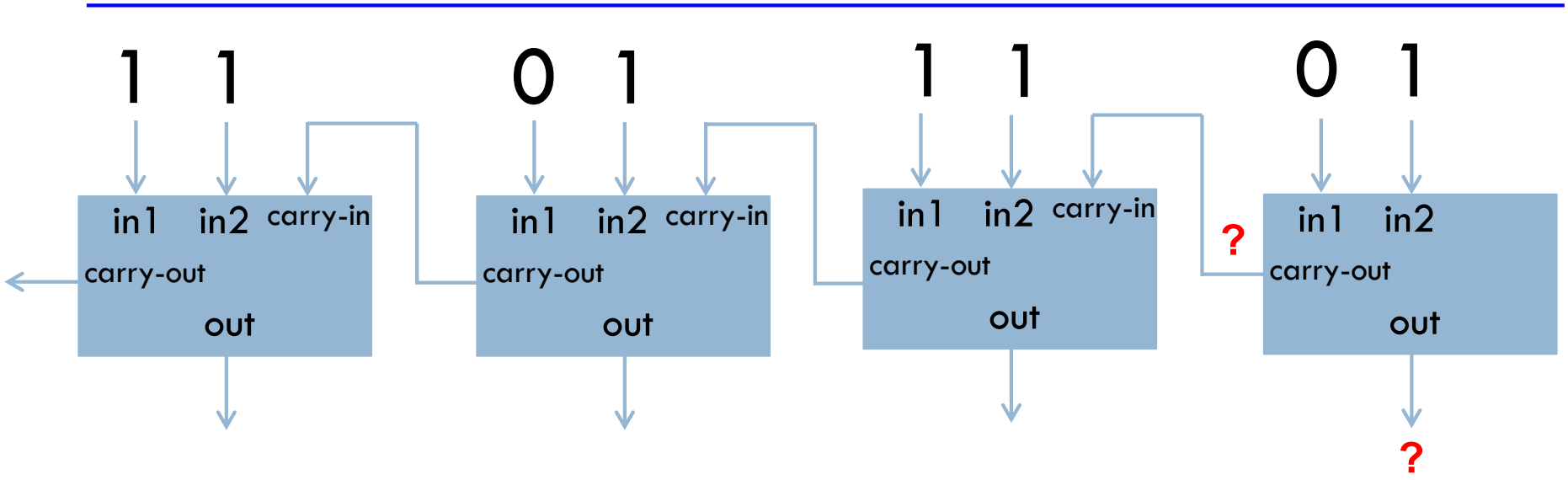


Adding with components



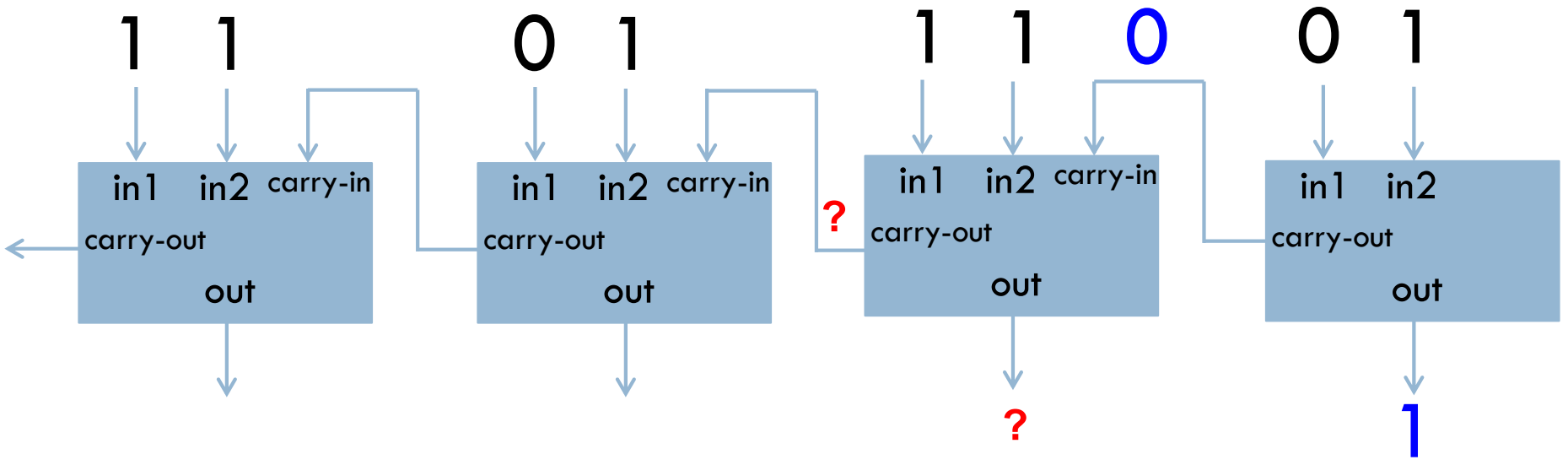
Adding with components

```
  01010
+ 01111
-------
```



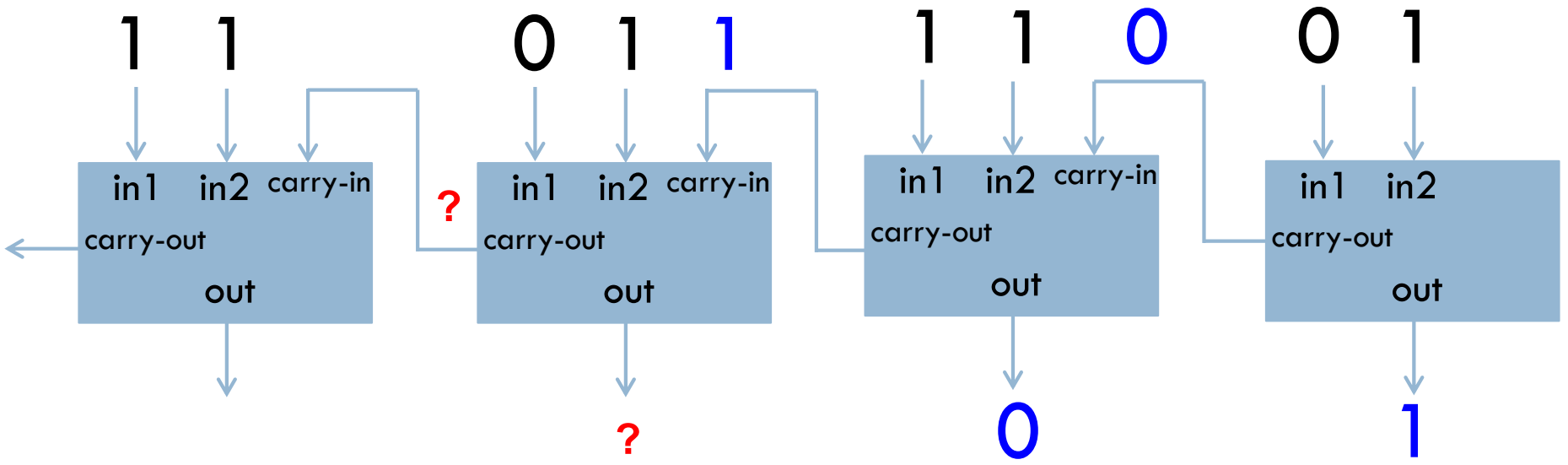
Adding with components

```
    0
  01010
+ 01111
-------
   1
```



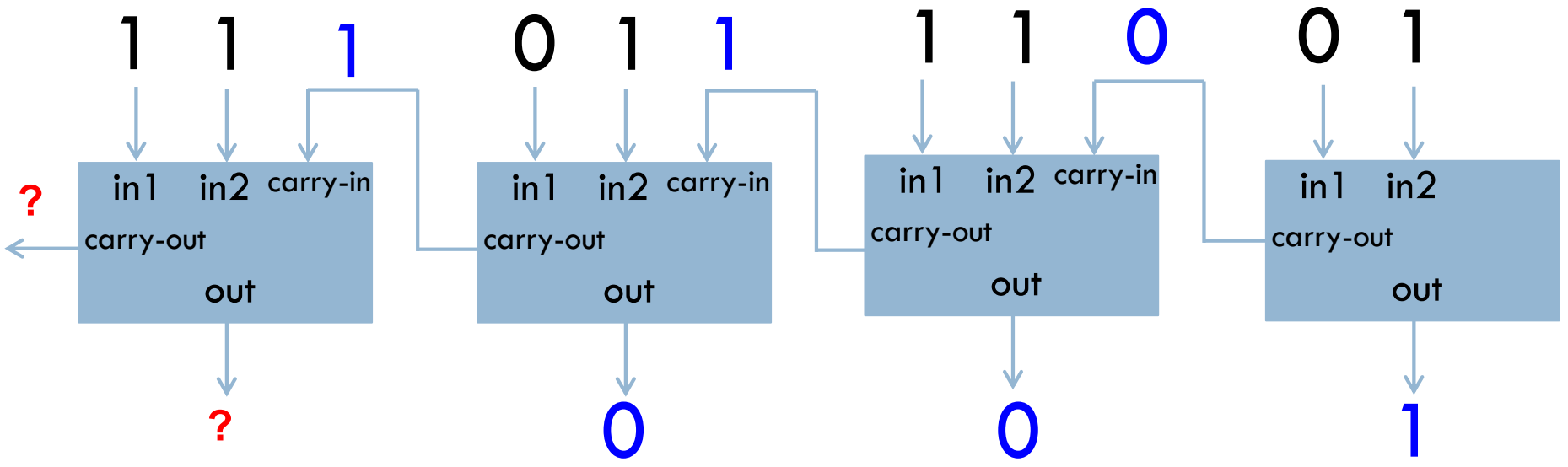
Adding with components

```
   10
  01010
+ 01111
--------
   01
```



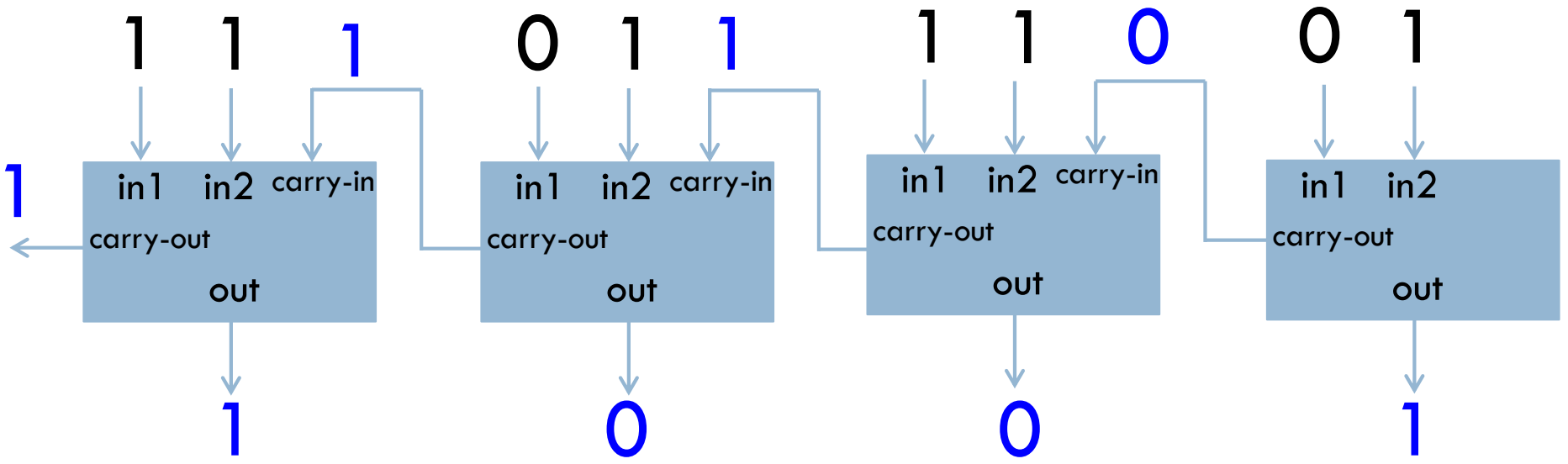
Adding with components

```
  110
 01010
+01111
------
  001
```

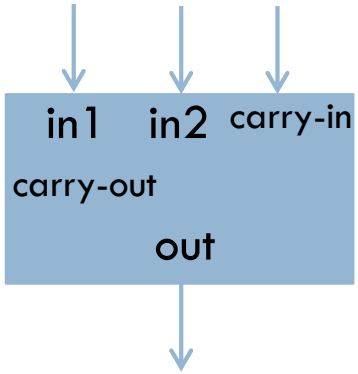


Adding with components

```
  1110
  01010
+ 01111
-------
 11001
```

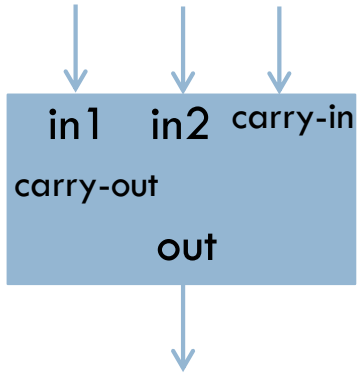


Implementing the component



What goes on inside the component?

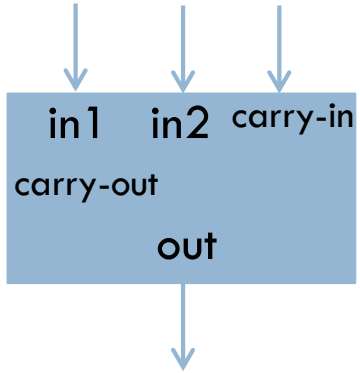
Implementing the component



in1	in2	carry-in	out	carry-out
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

What are the outputs?

Implementing the component



in1	in2	carry-in	out	carry-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Gates and Boolean logic

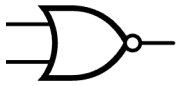
NOT ($\neg$) 

AND ($\wedge$) 

OR (V) 

XOR ($\oplus$) 

NAND ($\overline{\wedge}$) 

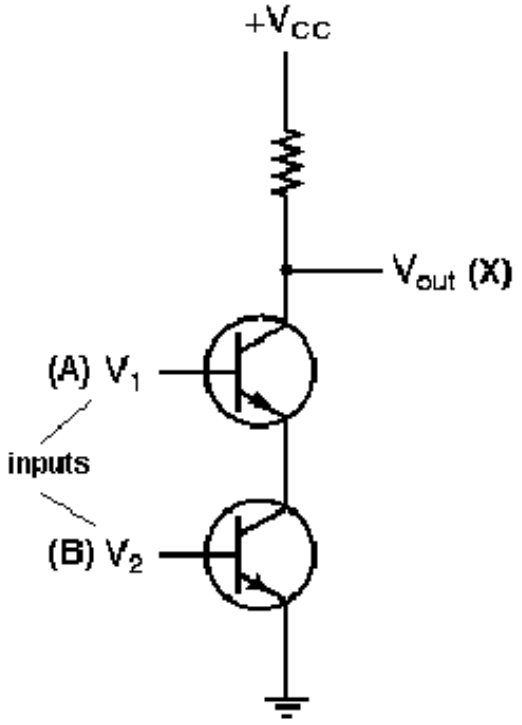
NOR ($\overline{\vee}$) 

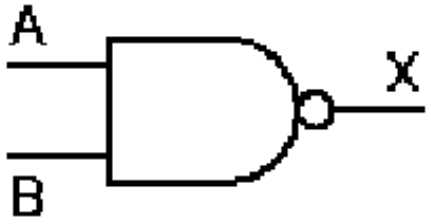
X	Y	$\neg$ X	X $\wedge$ Y	X $\vee$ Y	X $\oplus$ Y	X $\overline{\wedge}$ Y	X $\overline{\vee}$ Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

Gates have inputs and outputs
values are 0 or 1

They are **hardware** components!

Gates as hardware

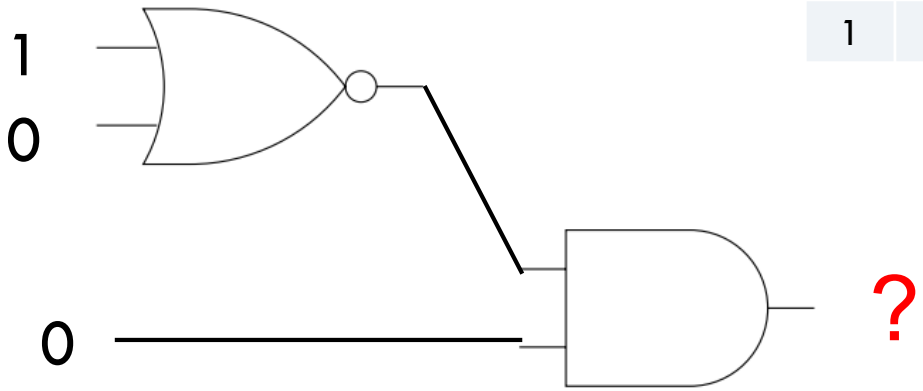


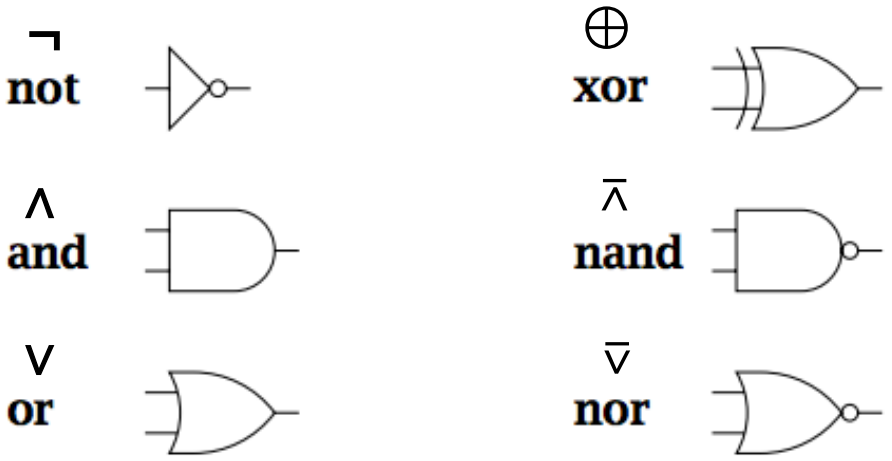


A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Utilizing gates

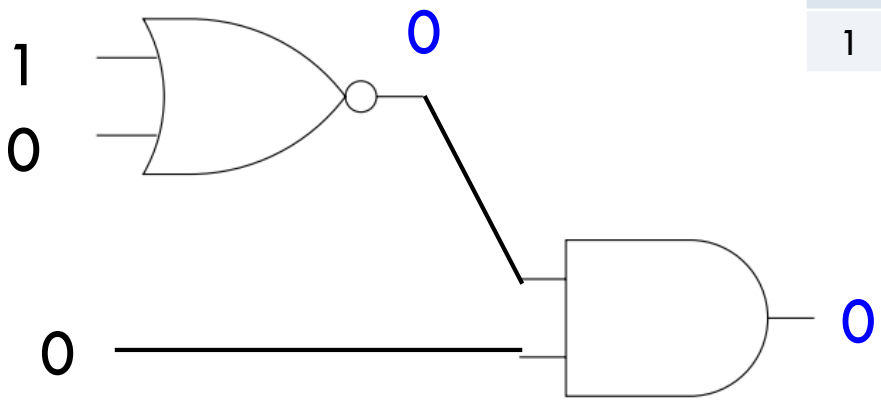
X	Y	¬ X	X ∧ Y	X ∨ Y	X ⊕ Y	X ⊼ Y	X ⊽ Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

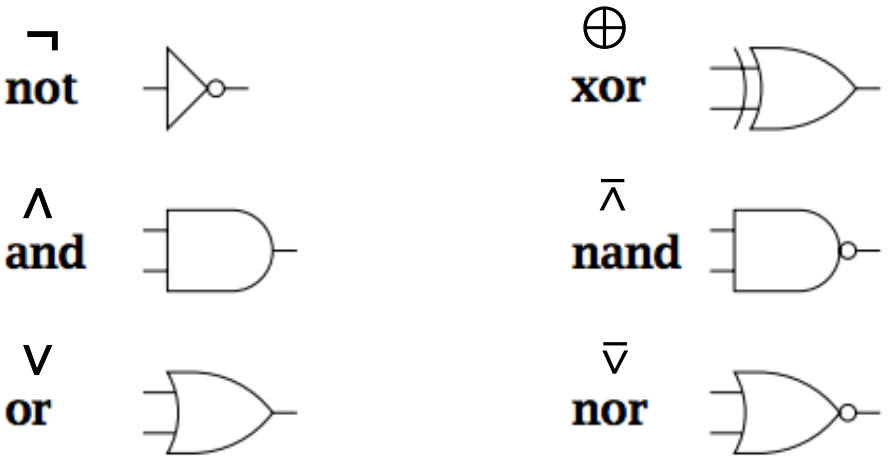




Utilizing gates

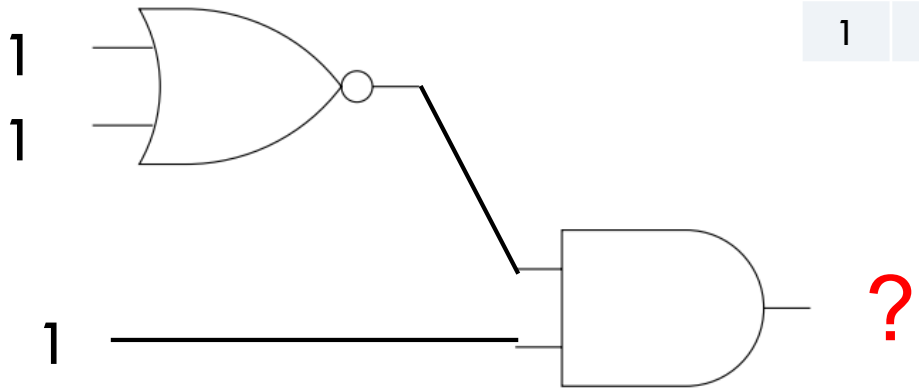
X	Y	¬ X	X ∧ Y	X ∨ Y	X ⊕ Y	X ⊼ Y	X ⊽ Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

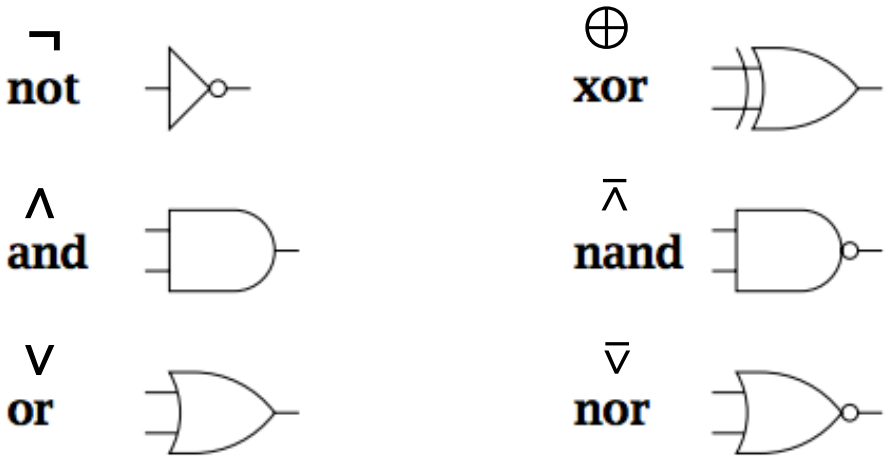




Utilizing gates

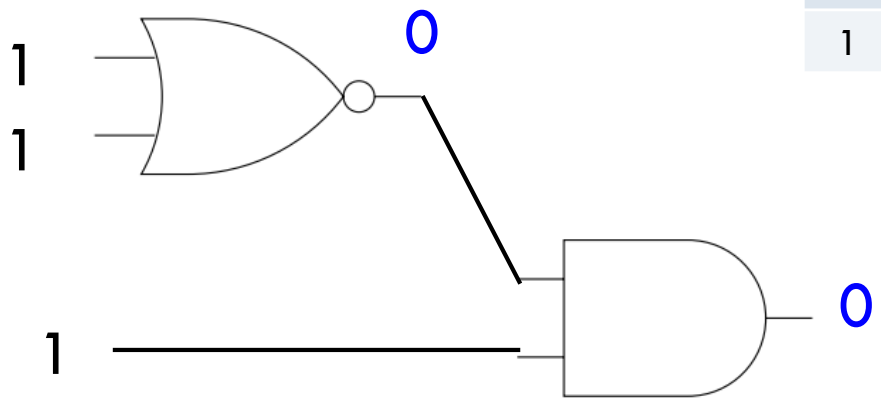
X	Y	¬ X	X ∧ Y	X ∨ Y	X ⊕ Y	X ⊼ Y	X ⊽ Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0



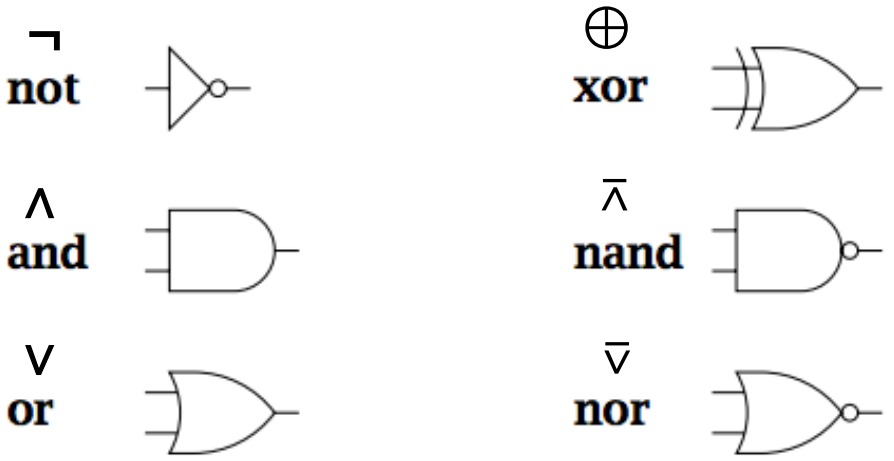


Utilizing gates

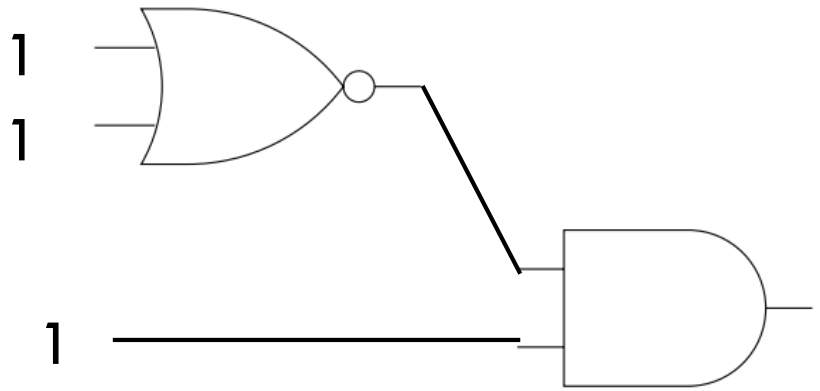
X	Y	¬ X	X ∧ Y	X ∨ Y	X ⊕ Y	X ⊼ Y	X ⊽ Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0



When is this circuit 1?



Utilizing gates



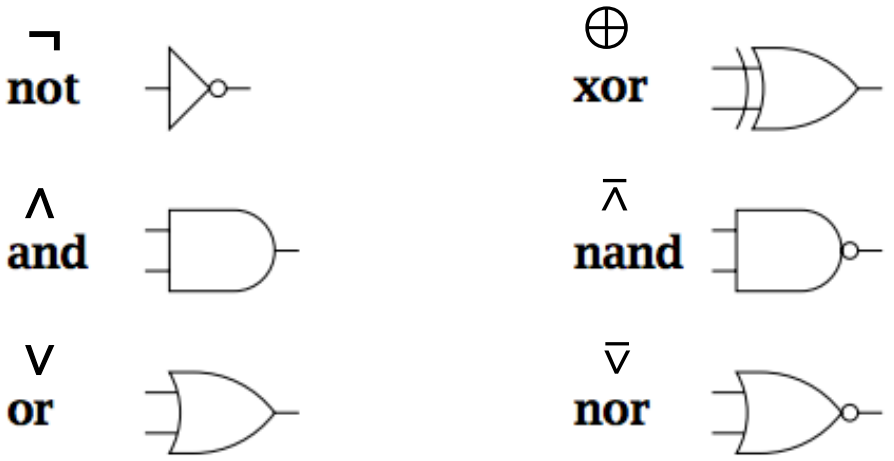
X	Y	Z	OUT
0	0	0	0
0	**0**	**1**	**1**
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Designing more interesting circuits

X	Y	¬ X	X ∧ Y	X ∨ Y	X ⊕ Y	X ⊼ Y	X ⊽ Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

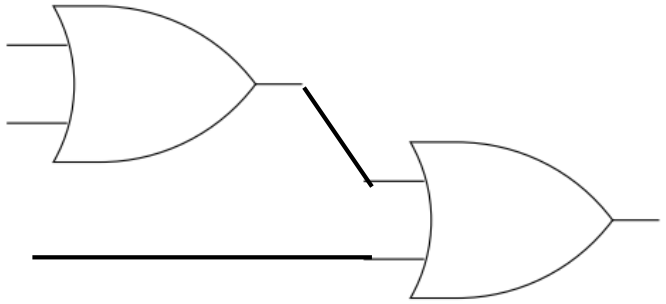
X	Y	Z	OUT
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Design a circuit for this



Designing more interesting circuits

X	Y	Z	OUT
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

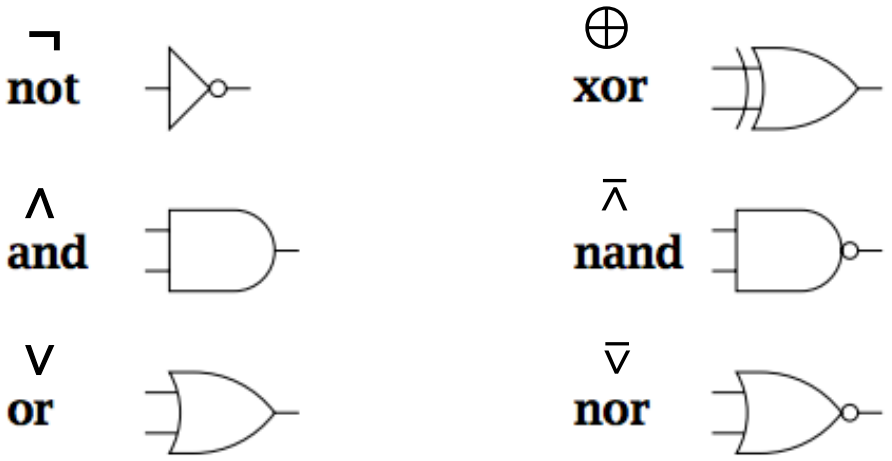


Designing more interesting circuits

X	Y	¬ X	X ∧ Y	X ∨ Y	X ⊕ Y	X ⊼ Y	X ⊽ Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

X	Y	Z	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	0	0

Design a circuit for this



Truth table to Boolean expression

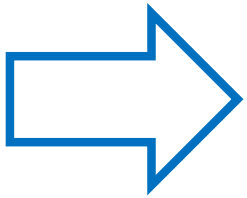
What approaches have we seen for developing a Boolean expression from a truth table?

X	Y	Z	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Truth table to Boolean expression

What approaches have we seen for developing a Boolean expression from a truth table?

X	Y	Z	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



minterm expansion (DNF – OR of ANDs)

maxterm expansion (CNF – AND or ORs)

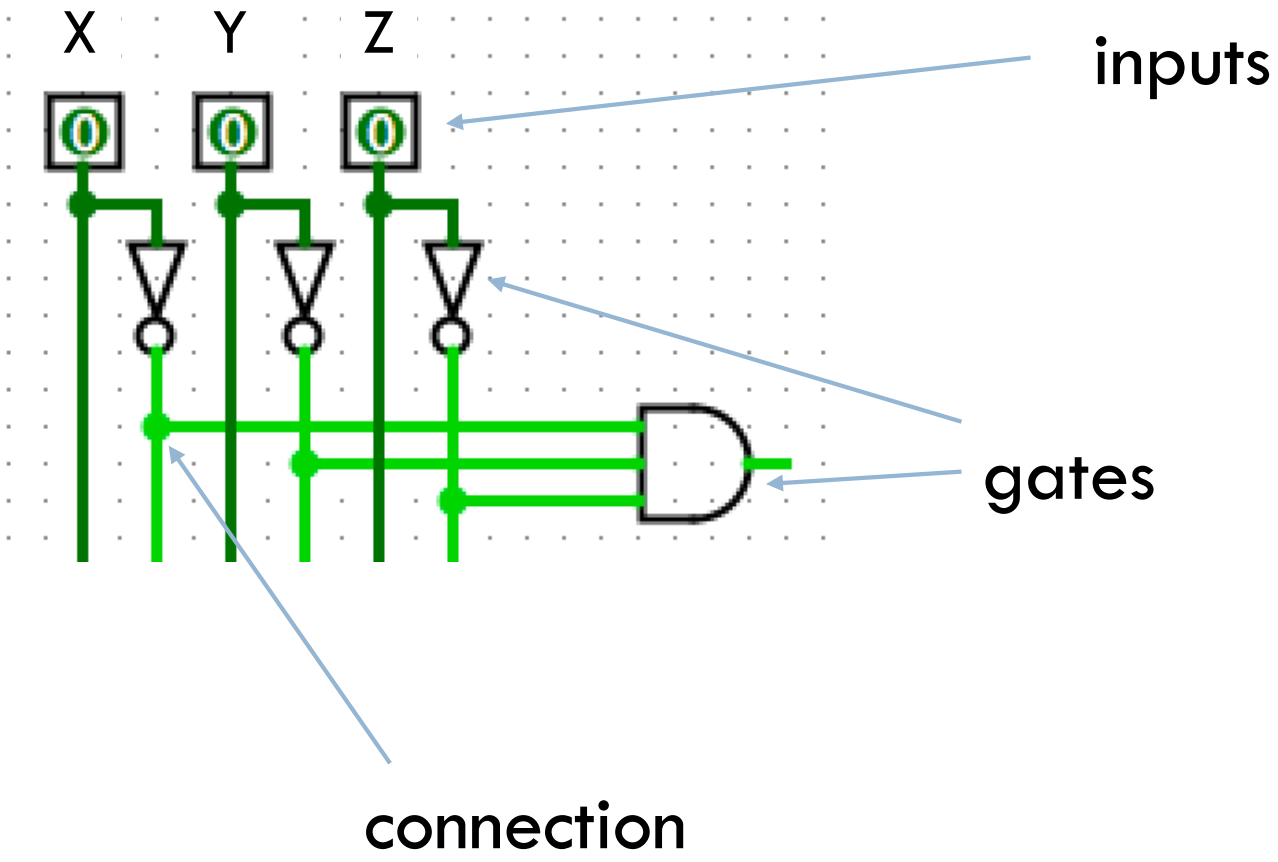
K-Maps (OR of ANDs)

Minterm expansion

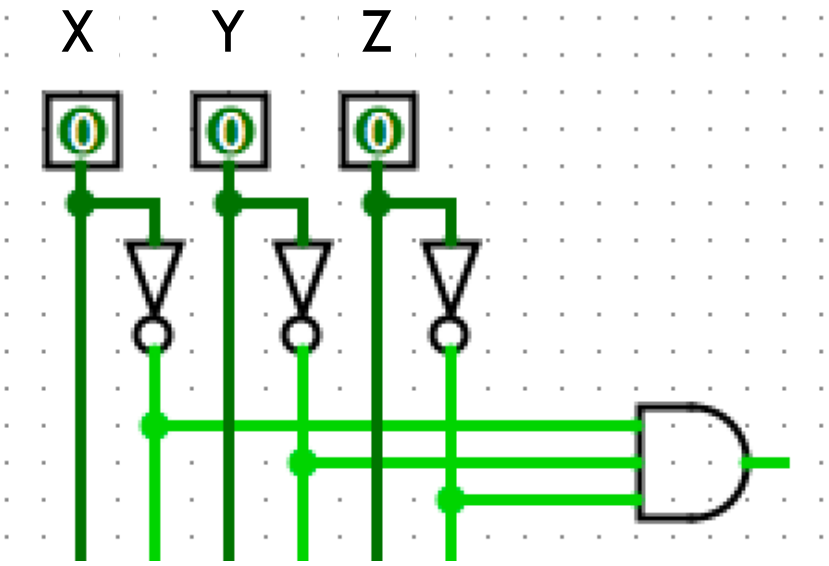
All these should be 1 and everything else 0

X	Y	Z	OUT	Minterm
0	0	0	1	⌐X ∧ ⌐Y ∧ ⌐Z
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	X ∧ ⌐Y ∧ ⌐Z
1	0	1	0	
1	1	0	1	X ∧ Y ∧ ⌐Z
1	1	1	0	

Minterm expansion

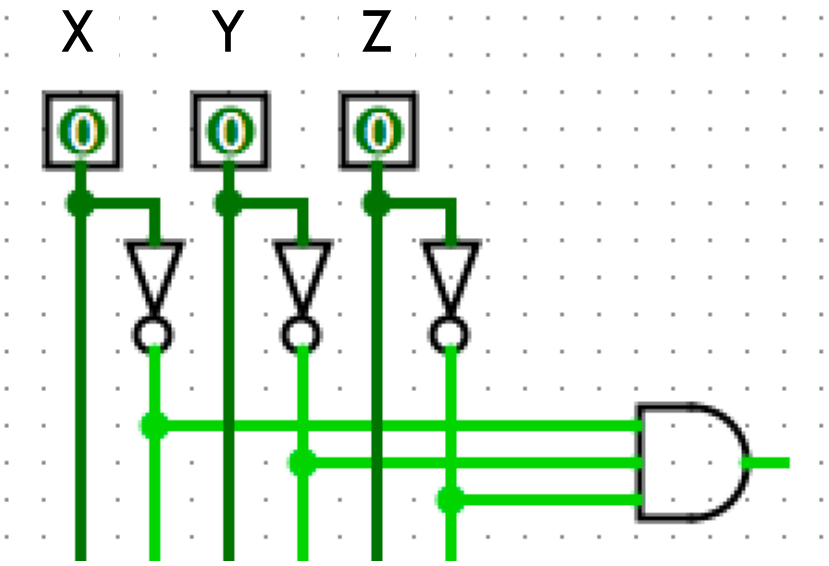


Minterm expansion



When will this AND-gate be 1?

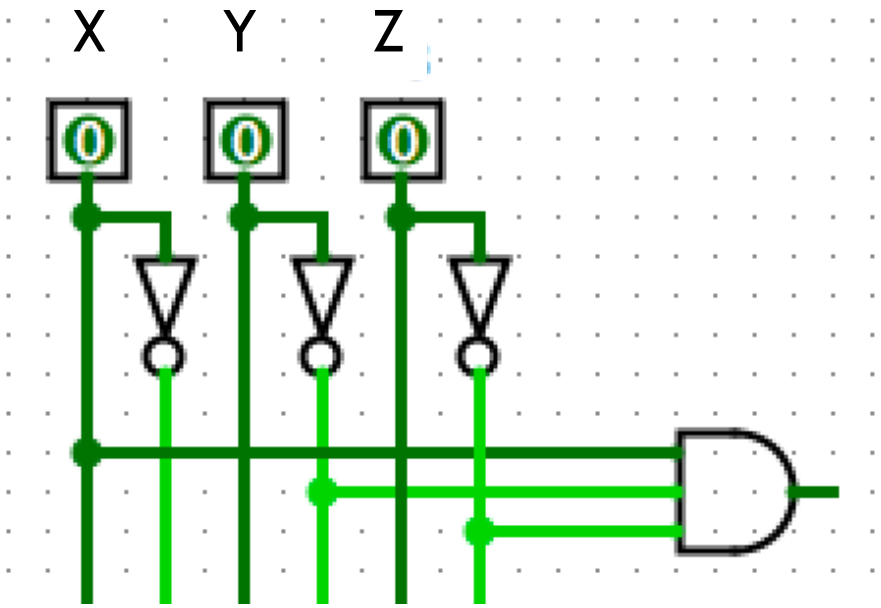
Minterm expansion



Only when X=0, Y=0, Z=0

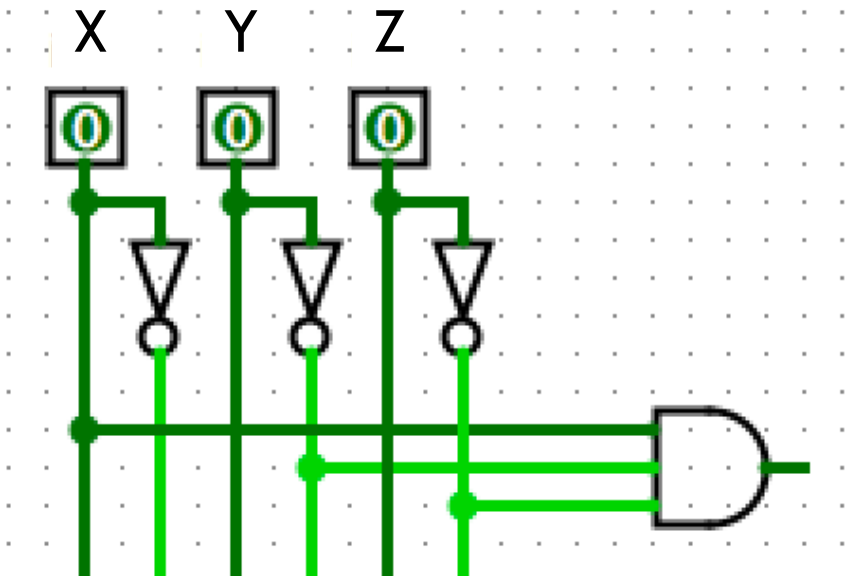
X	Y	Z	OUT	Minterm
0	0	0	1	⌐X ∧ ⌐Y ∧ ⌐Z

Minterm expansion



When will this AND-gate be 1?

Minterm expansion

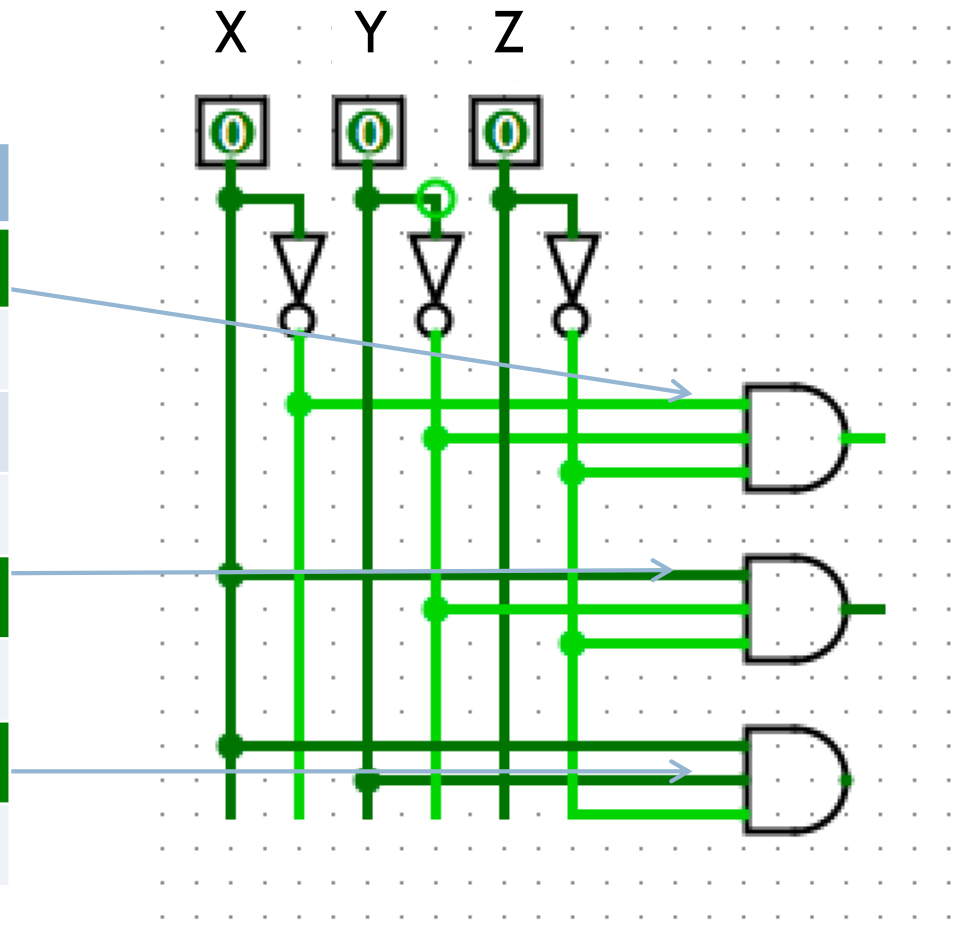


Only when in1=1, in2=0, in3=0

X	Y	Z	OUT	Minterm
1	0	0	1	X ∧ ⌐Y ∧ ⌐Z

Minterm expansion

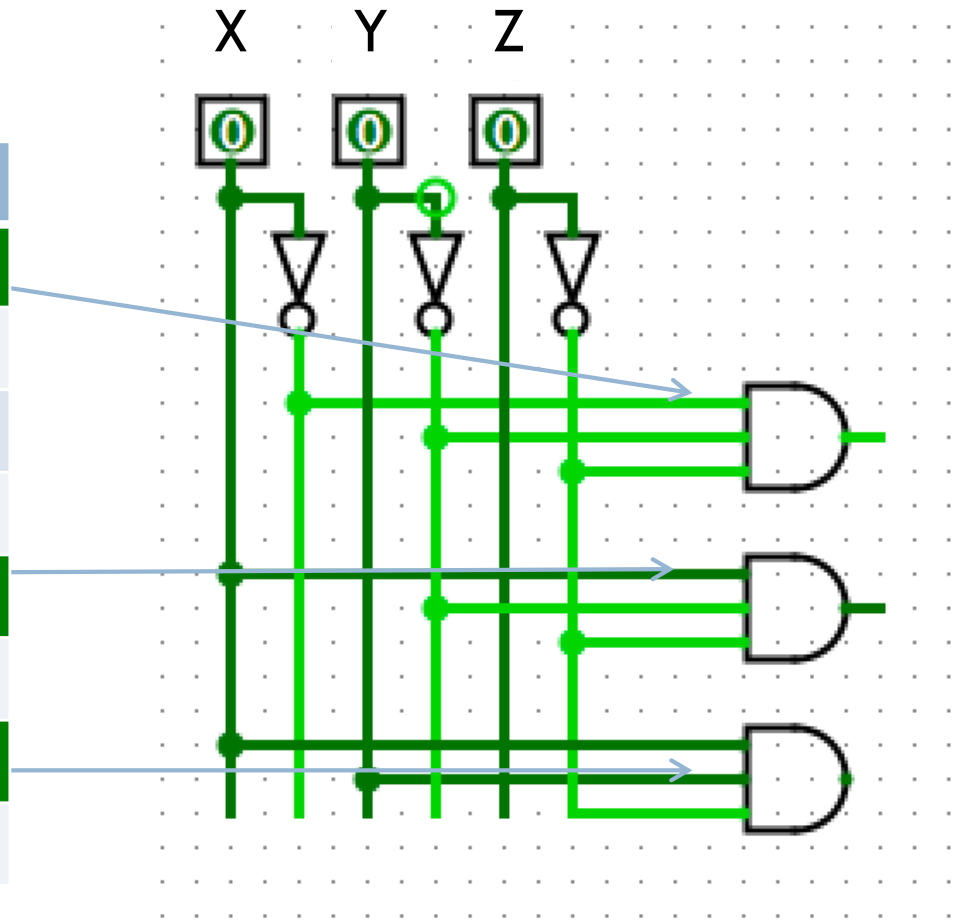
X	Y	Z	OUT	Minterm
0	0	0	1	⌐X ∧ ⌐Y ∧ ⌐Z
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	X ∧ ⌐Y ∧ ⌐Z
1	0	1	0	
1	1	0	1	X ∧ Y ∧ ⌐Z
1	1	1	0	



Make a conjunction for each minterm

Minterm expansion

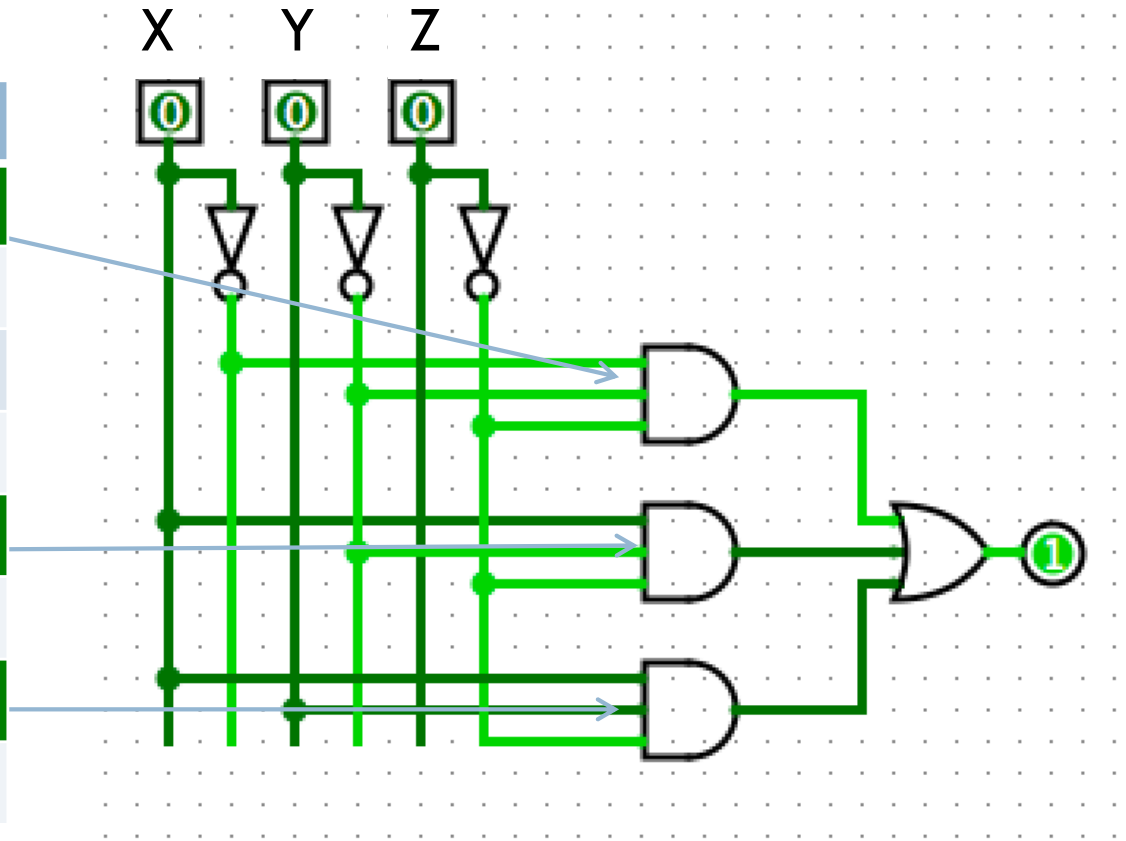
X	Y	Z	OUT	Minterm
0	0	0	1	⌐X ∧ ⌐Y ∧ ⌐Z
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	X ∧ ⌐Y ∧ ⌐Z
1	0	1	0	
1	1	0	1	X ∧ Y ∧ ⌐Z
1	1	1	0	



How do we combine these?

Minterm expansion

X	Y	Z	OUT	Minterm
0	0	0	1	⌐X ∧ ⌐Y ∧ ⌐Z
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	X ∧ ⌐Y ∧ ⌐Z
1	0	1	0	
1	1	0	1	X ∧ Y ∧ ⌐Z
1	1	1	0	



OR-gate: (⌐X ∧ ⌐Y ∧ ⌐Z) ∨ (X ∧ ⌐Y ∧ ⌐Z) ∨ (X ∧ Y ∧ ⌐Z)

K-maps

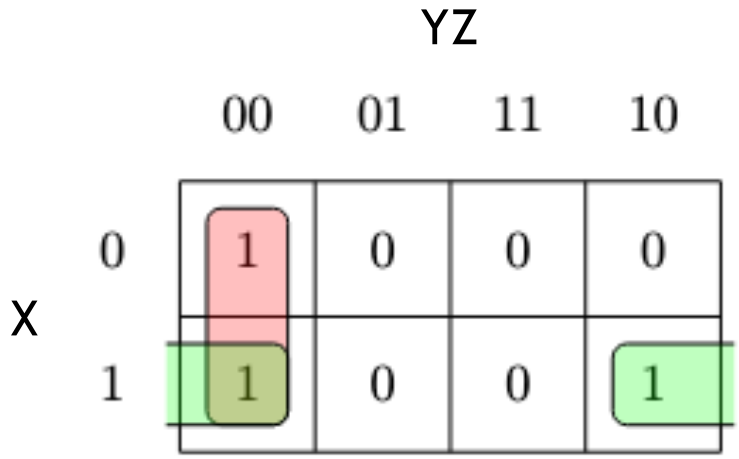
Can you use a K-map to come up with a Boolean expression for this table?

X	Y	Z	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

K-maps

Can you use a K-map to come up with a Boolean expression for this table?

X	Y	Z	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

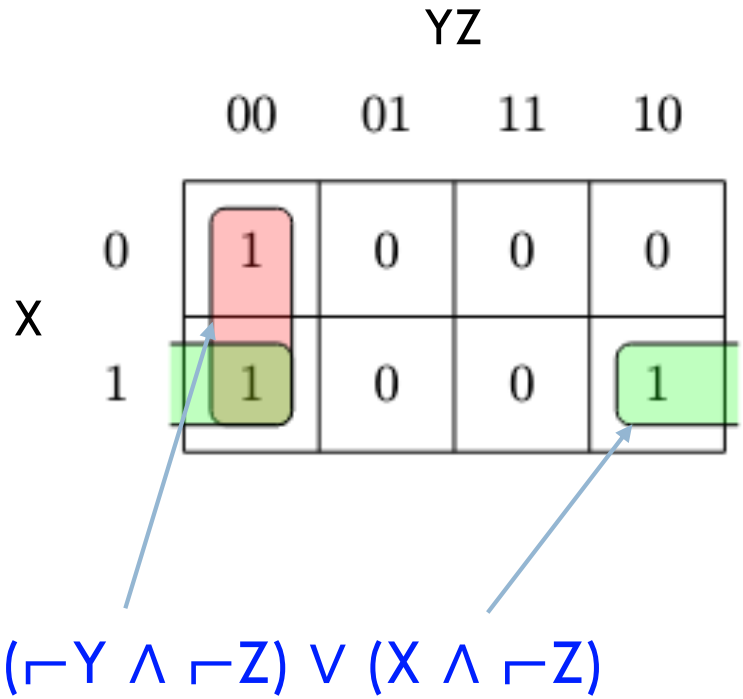


What is the Boolean expression?

K-maps

Can you use a K-map to come up with a Boolean expression for this table?

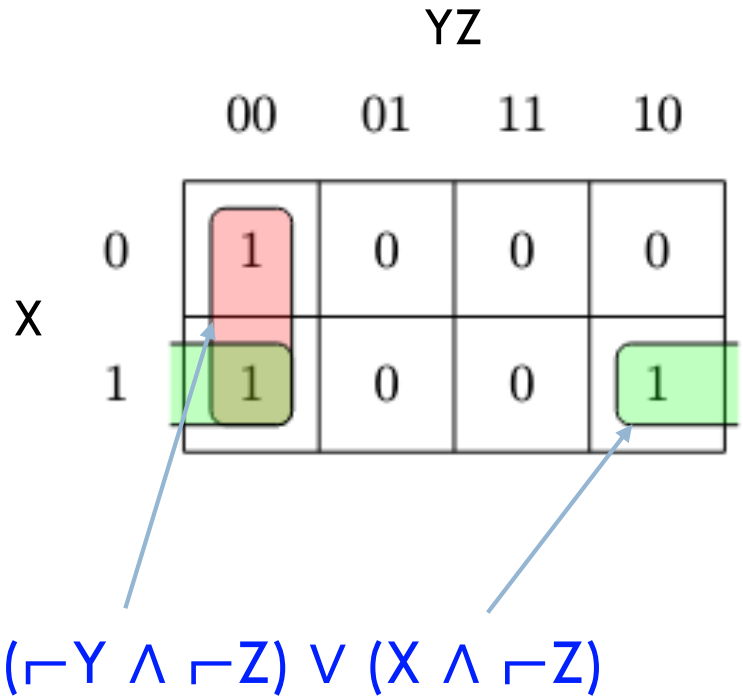
X	Y	Z	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



K-maps

Can you design a circuit?

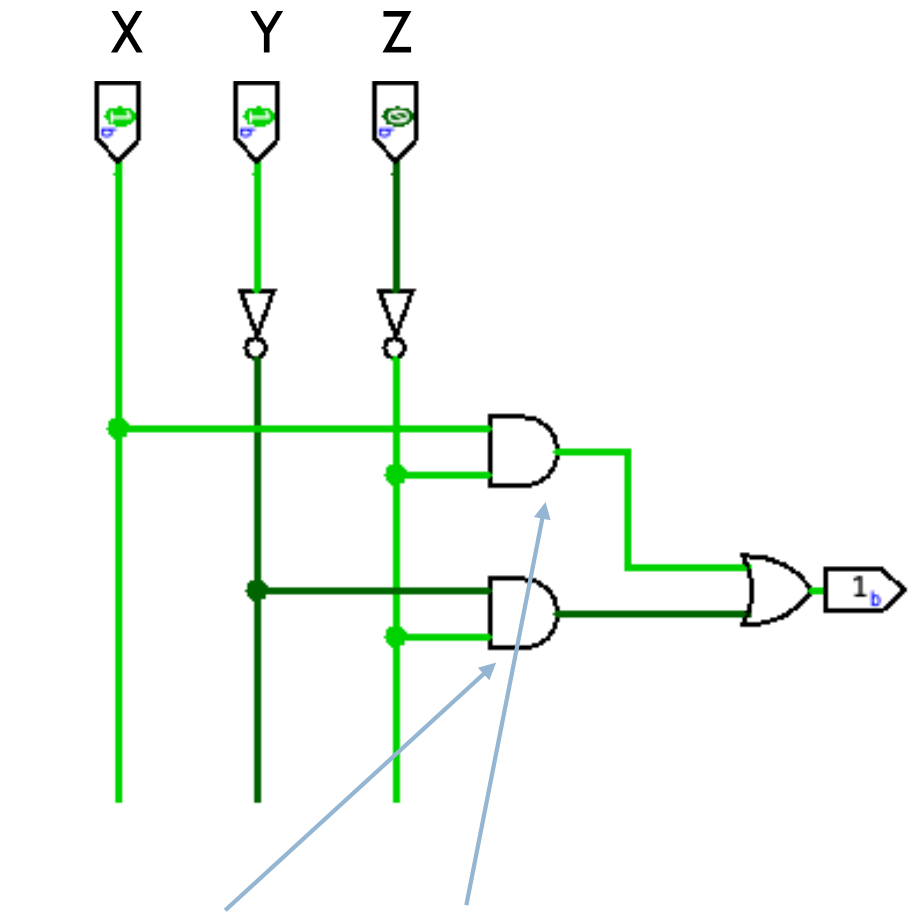
X	Y	Z	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



K-maps

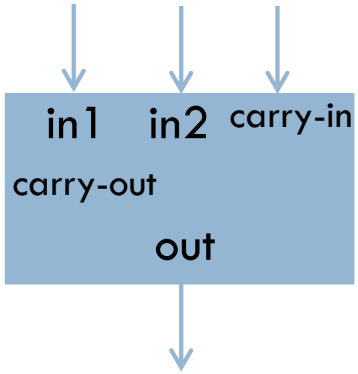
Can you design a circuit?

X	Y	Z	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



Back to addition…

in1	in2	carry-in	carry-out	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



A half-adder: no carry-in

A	B	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

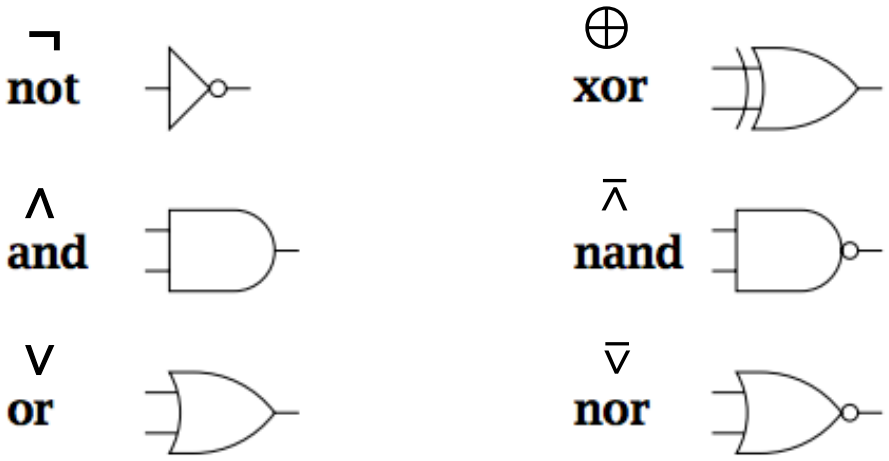
A half-adder: no carry-in

X	Y	¬ X	X ∧ Y	X ∨ Y	X ⊕ Y	X ⊼ Y	X ⊽ Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

A	B	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Hint: solve each output bit independently

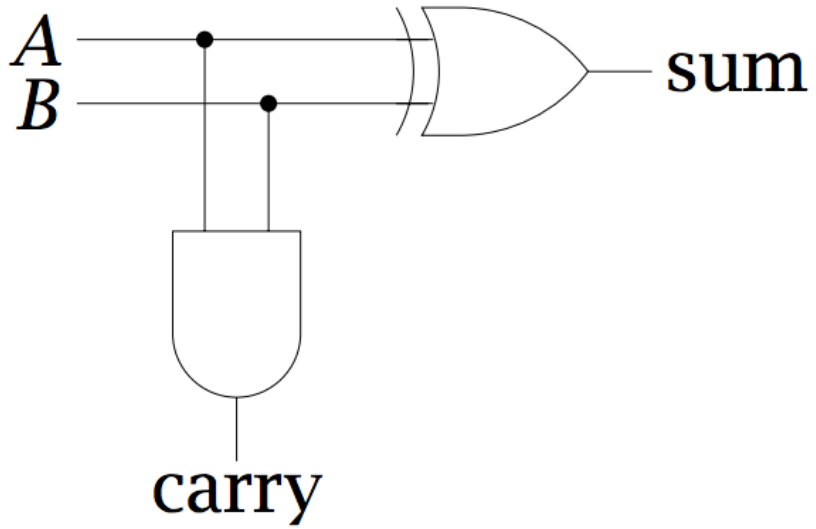
Design a circuit for this



A half-adder: no carry-in

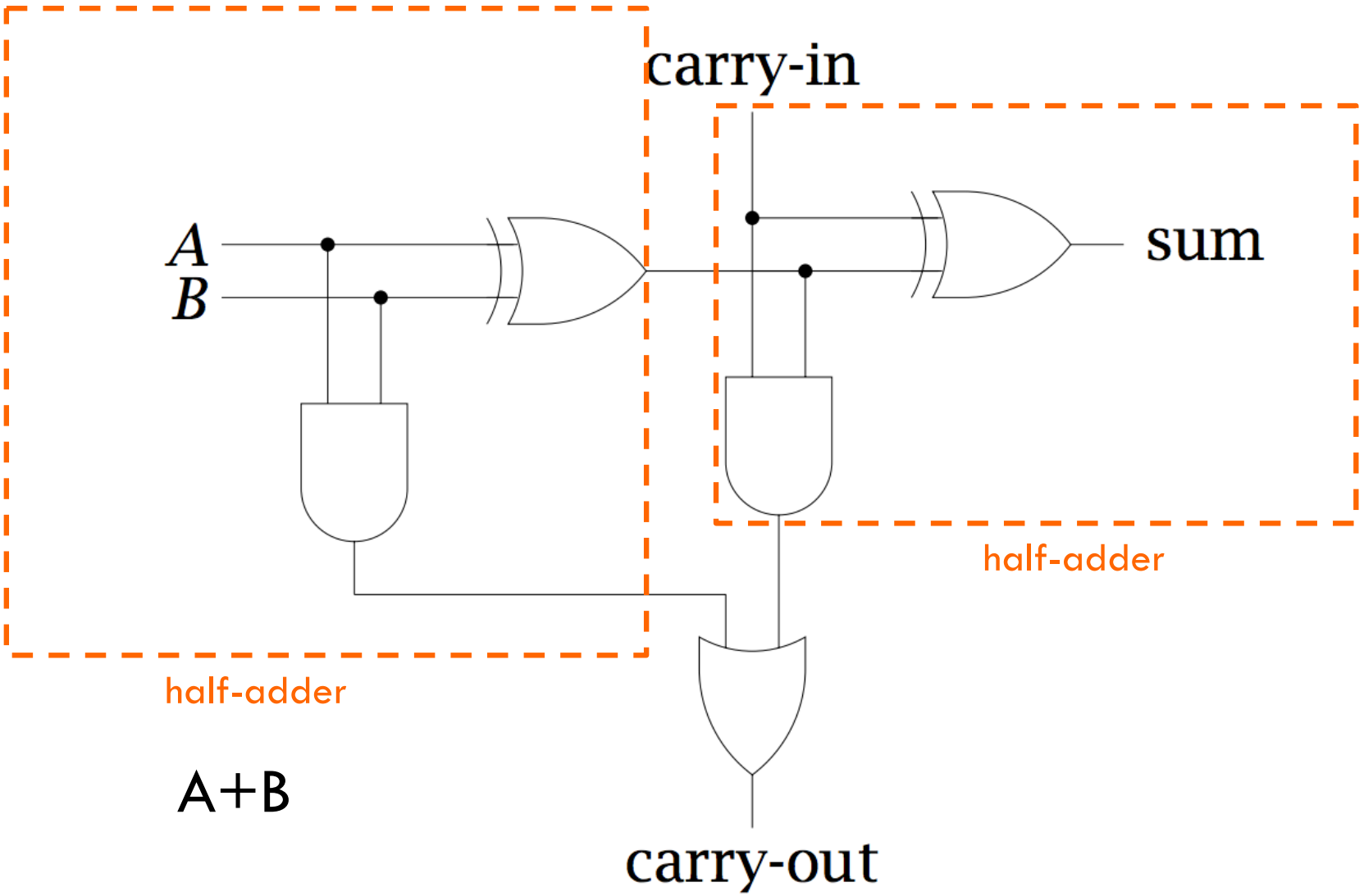
A	B	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

low order bit of A+B

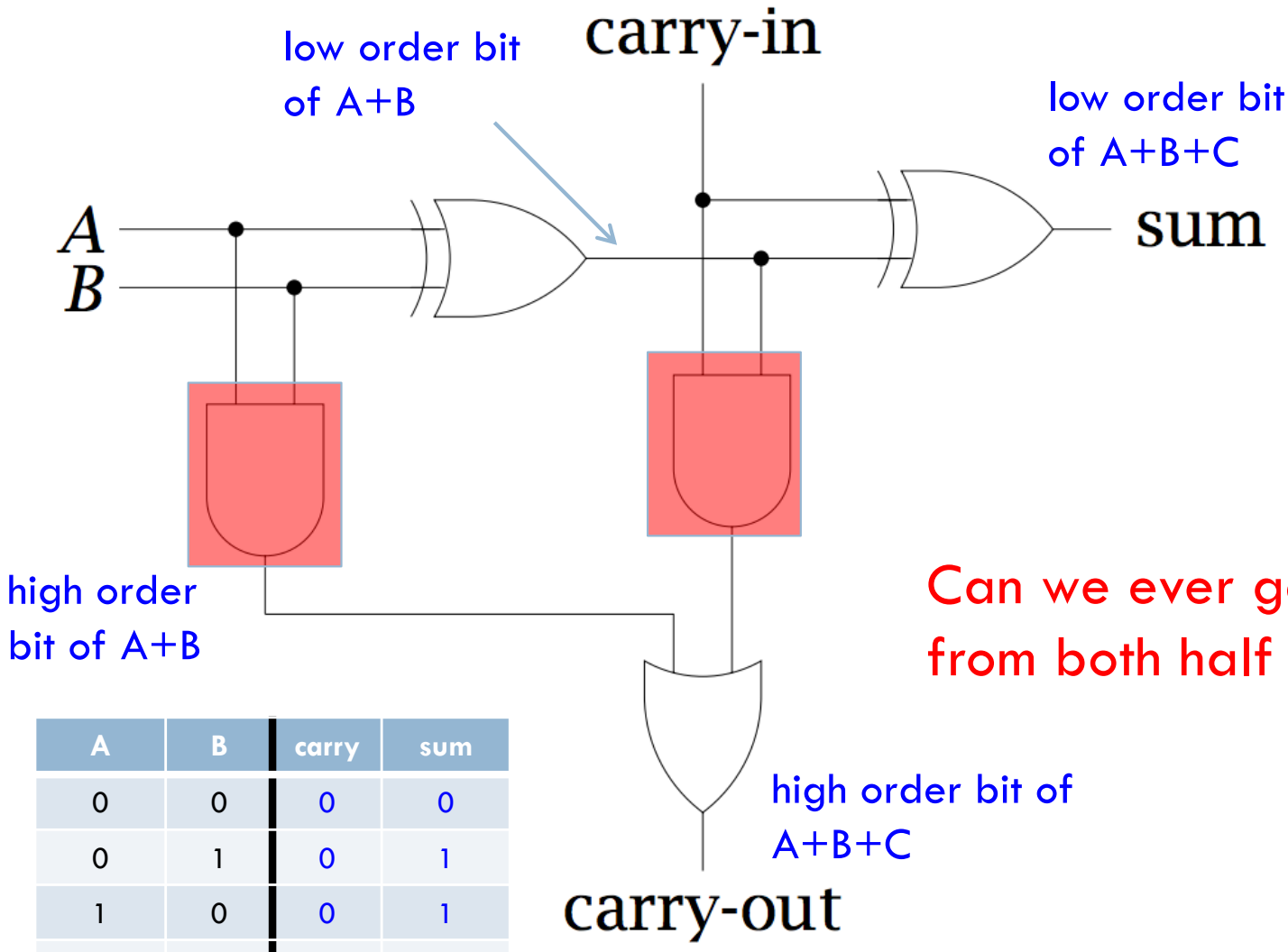


higher order bit of A+B

Implementing a full adder



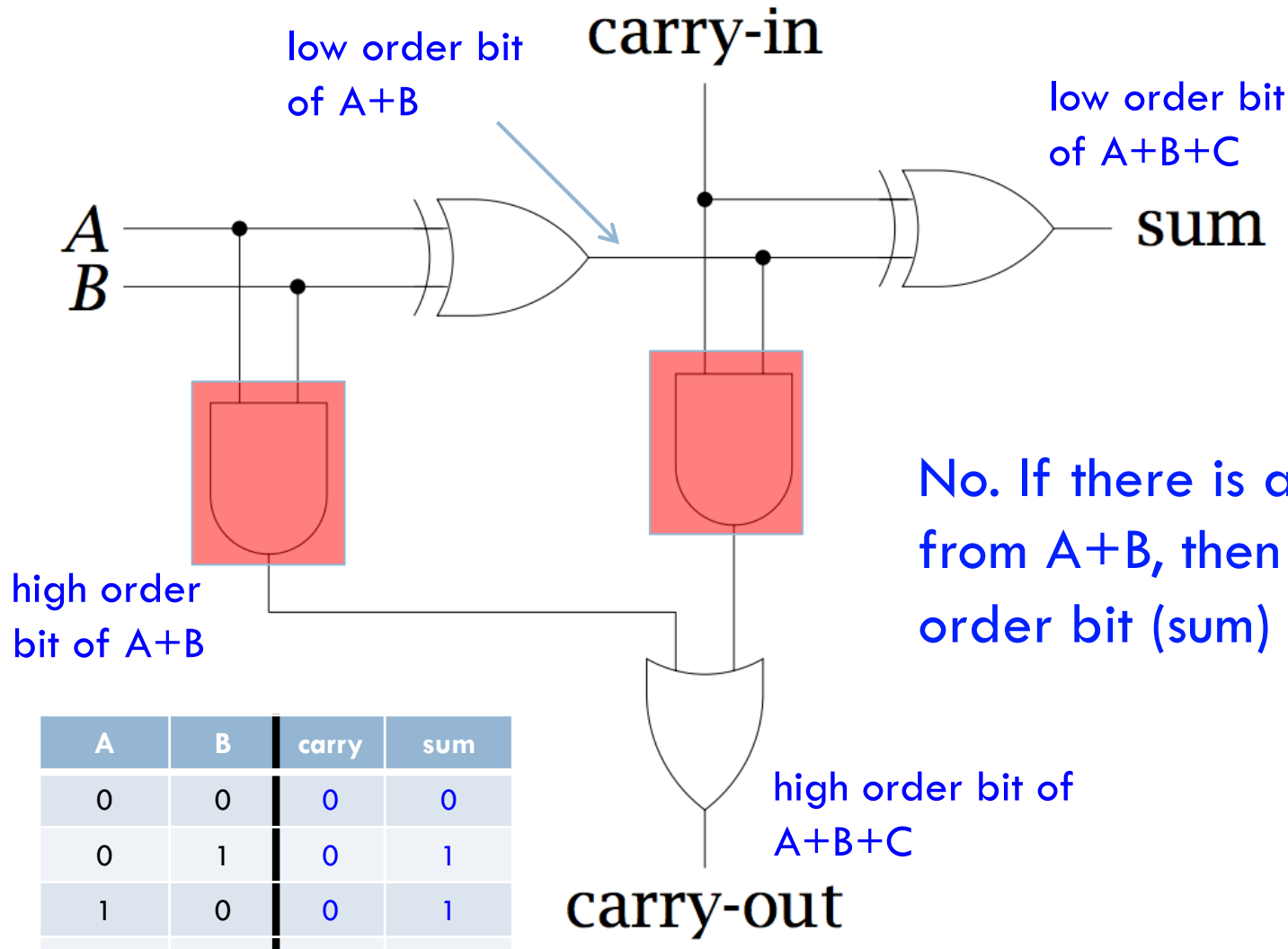
Implementing a full adder



Can we ever get a carry from both half adders?

A	B	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

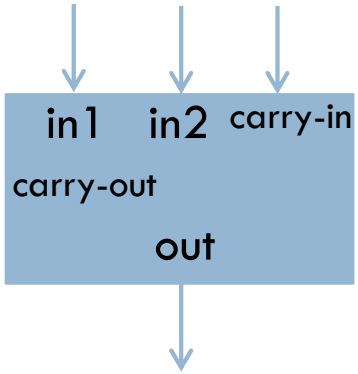
Implementing a full adder



A	B	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

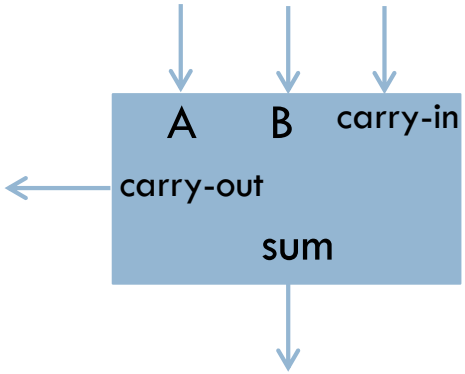
No. If there is a carry from A+B, then the low order bit (sum) must be 0

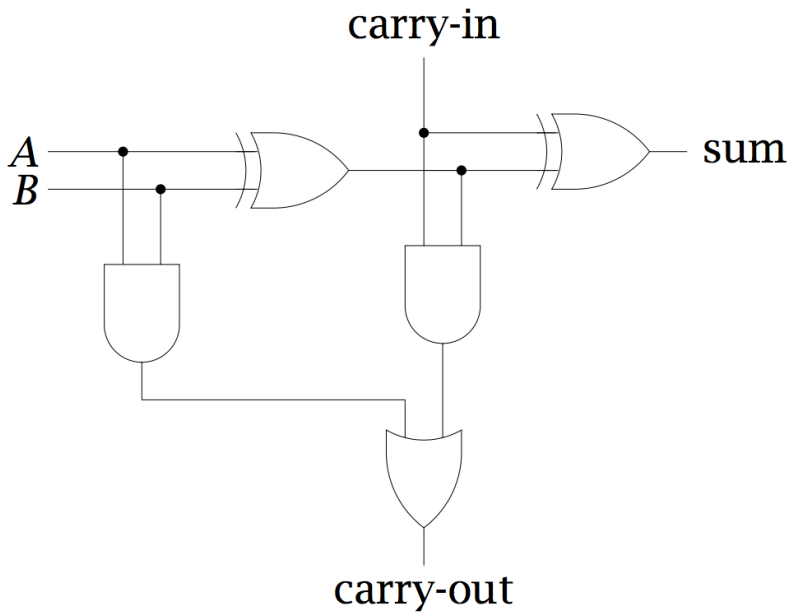
Implementing the component



What goes on inside the component?

Implementing the component





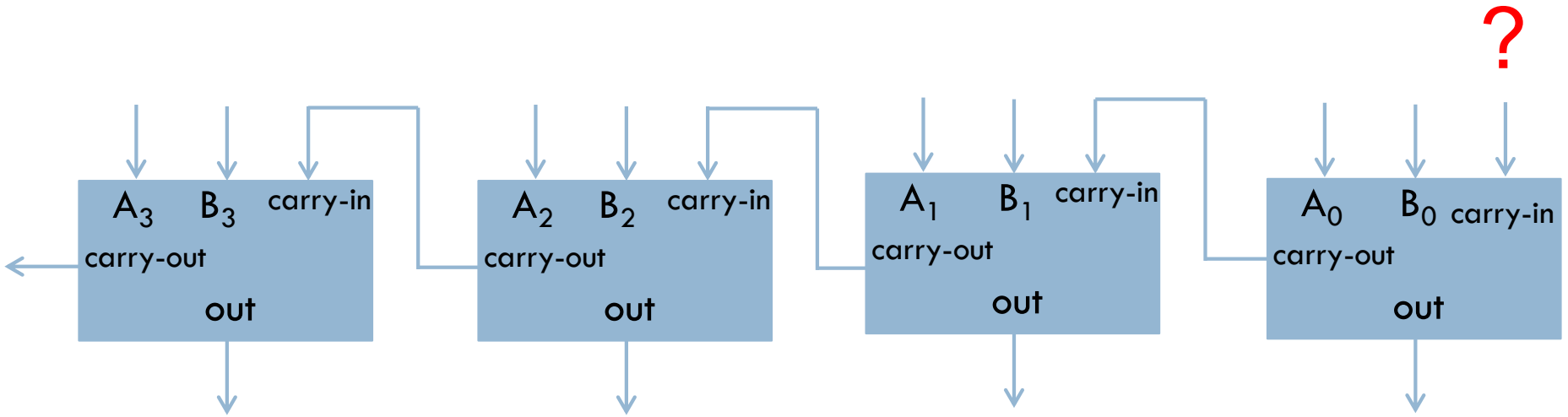
Ripple carry adder

To implement an *n*-bit adder, we chain together *n* full-adders, each adder handles one bit position

$A = A_3\ A_2\ A_1\ A_0$

$B = B_3\ B_2\ B_1\ B_0$

Adder for adding 4-bit numbers



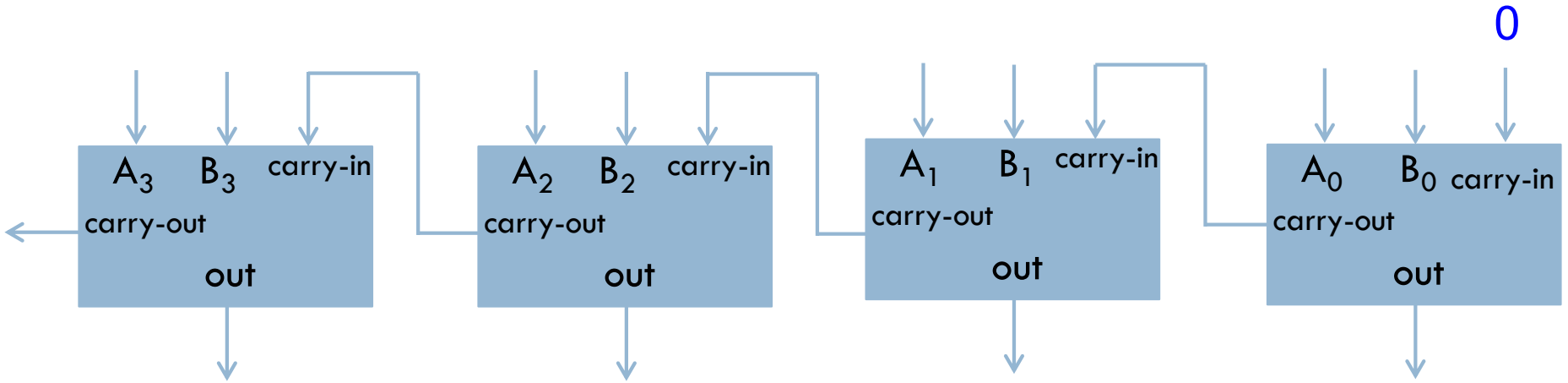
Ripple carry adder

To implement an *n*-bit adder, we chain together *n* full-adders, each adder handles one bit position

$A = A_3\ A_2\ A_1\ A_0$

$B = B_3\ B_2\ B_1\ B_0$

Adder for adding 4-bit numbers



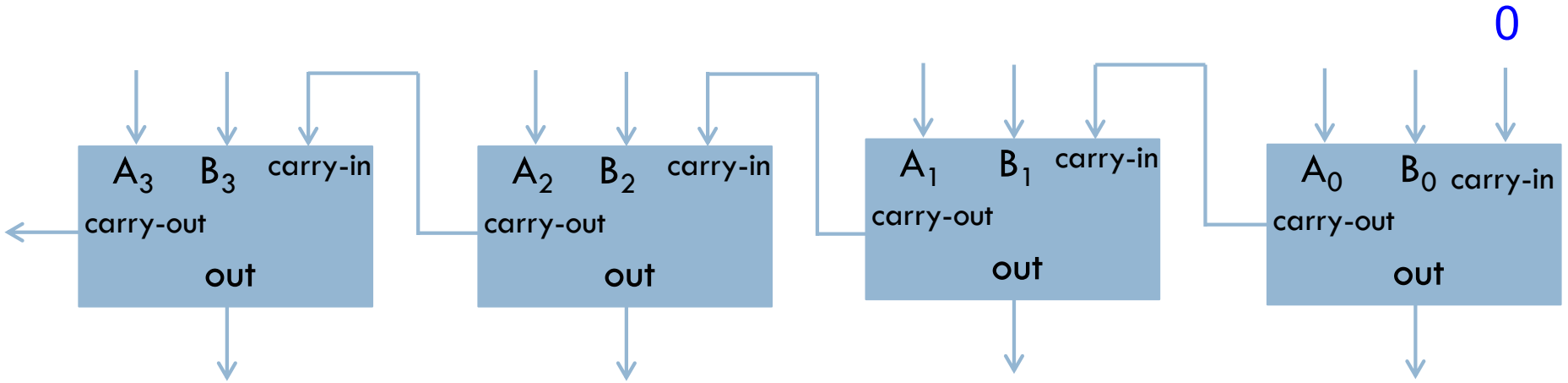
Ripple carry adder

To implement an *n*-bit adder, we chain together *n* full-adders, each adder handles one bit position

$A = A_3\ A_2\ A_1\ A_0$

$B = B_3\ B_2\ B_1\ B_0$

Adder for adding 4-bit numbers



Any downsides (computationally) to ripple carry adder?

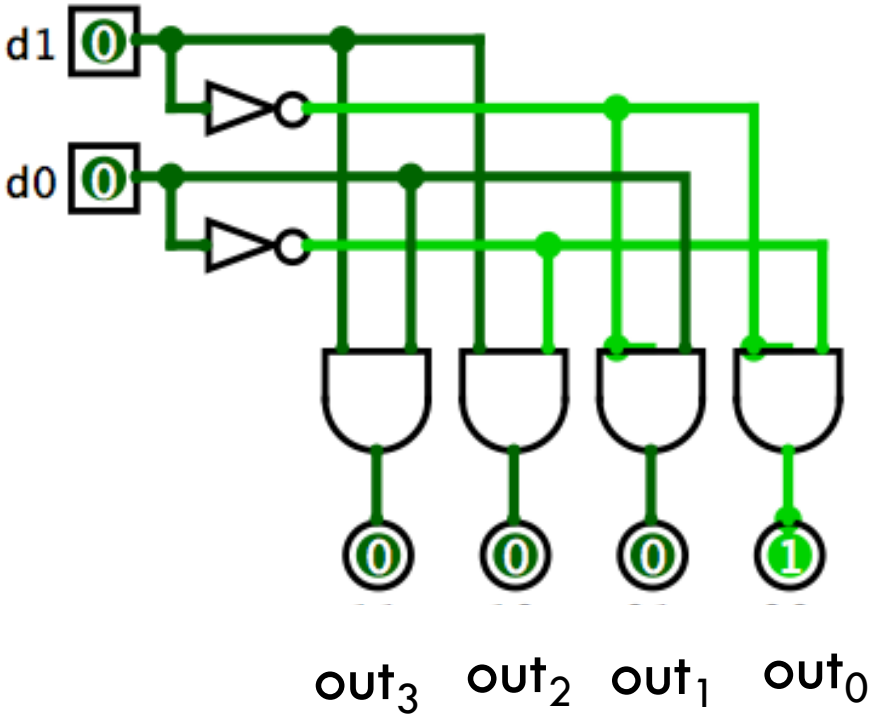
Look at ripple carry adder example

Many circuits

- half-adder
- full-adder (using half-adders)
- ripple-carry adder (using full-adders)

Simulator basics

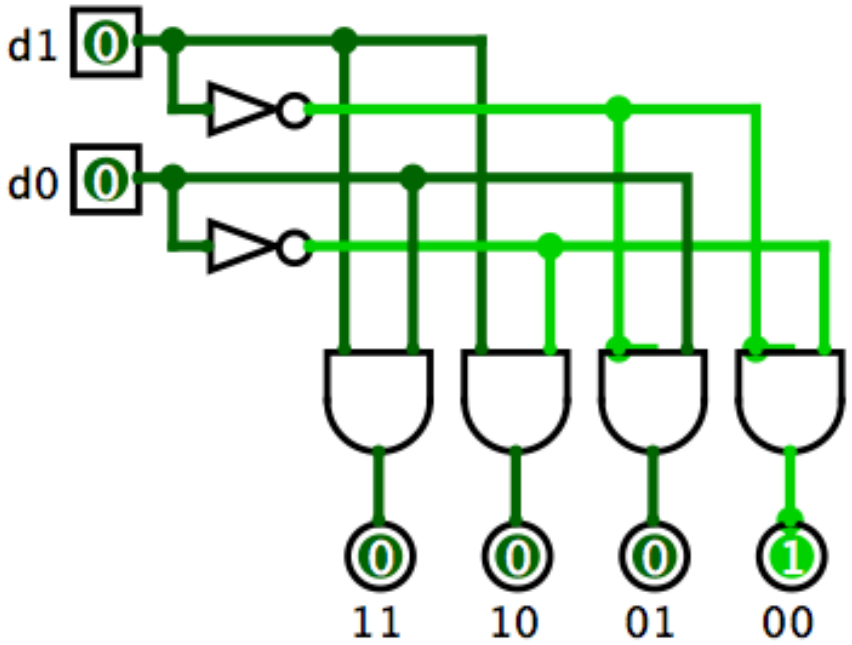
Mystery circuit



d_1	d_0	out_0	out_1	out_3	out_3
0	0				
0	1				
1	0				
1	1				

What does this circuit do?

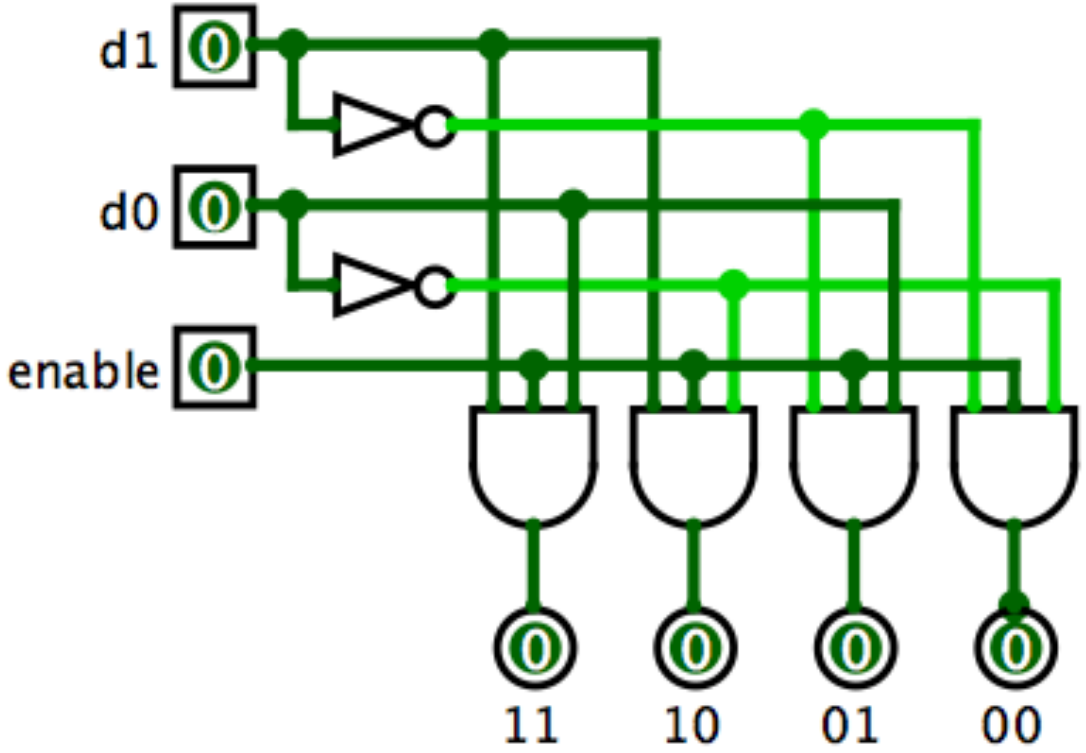
2-bit decoder



d_1	d_0	out_0	out_1	out_3	out_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

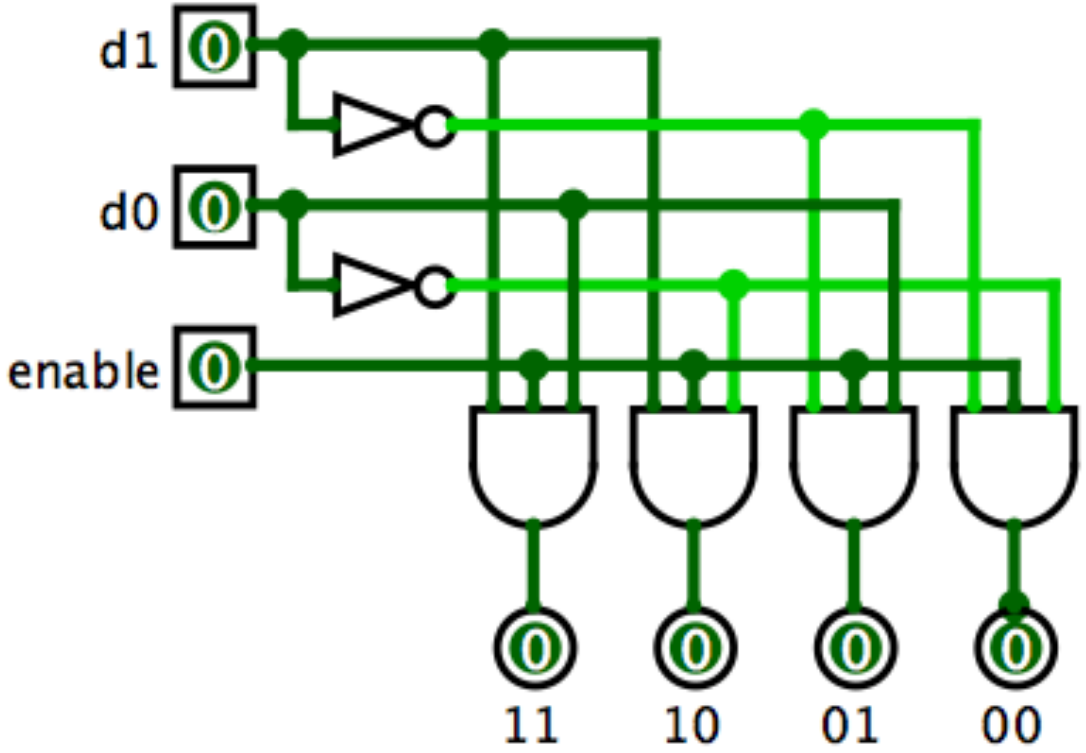
Sends '1' along one of the output lines

2-bit decoder*



What does the extra input do?

2-bit decoder*



When 0, doesn't select any lines, when 1, functions normally

3-bit decoder

3 inputs

How many output lines?

3-bit decoder

3 inputs

8 output lines

Inputs			Outputs							
d_2	d_1	d_0	t_{000}	t_{001}	t_{010}	t_{011}	t_{100}	t_{101}	t_{110}	t_{111}
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

3-bit decoder

3 inputs

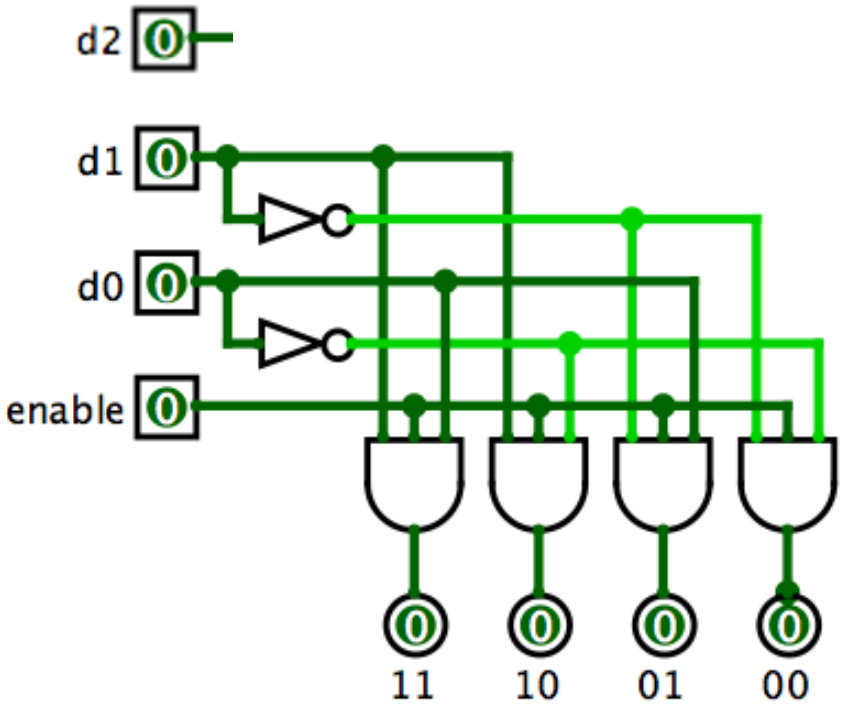
8 output lines

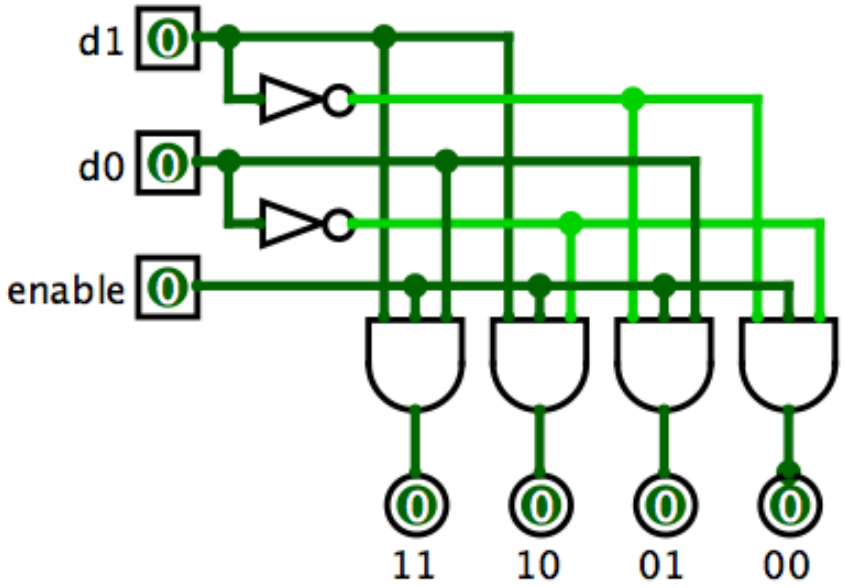
Inputs			Outputs							
d_2	d_1	d_0	t_{000}	t_{001}	t_{010}	t_{011}	t_{100}	t_{101}	t_{110}	t_{111}
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Could make from scratch

Better idea: reuse 2-bit decoders

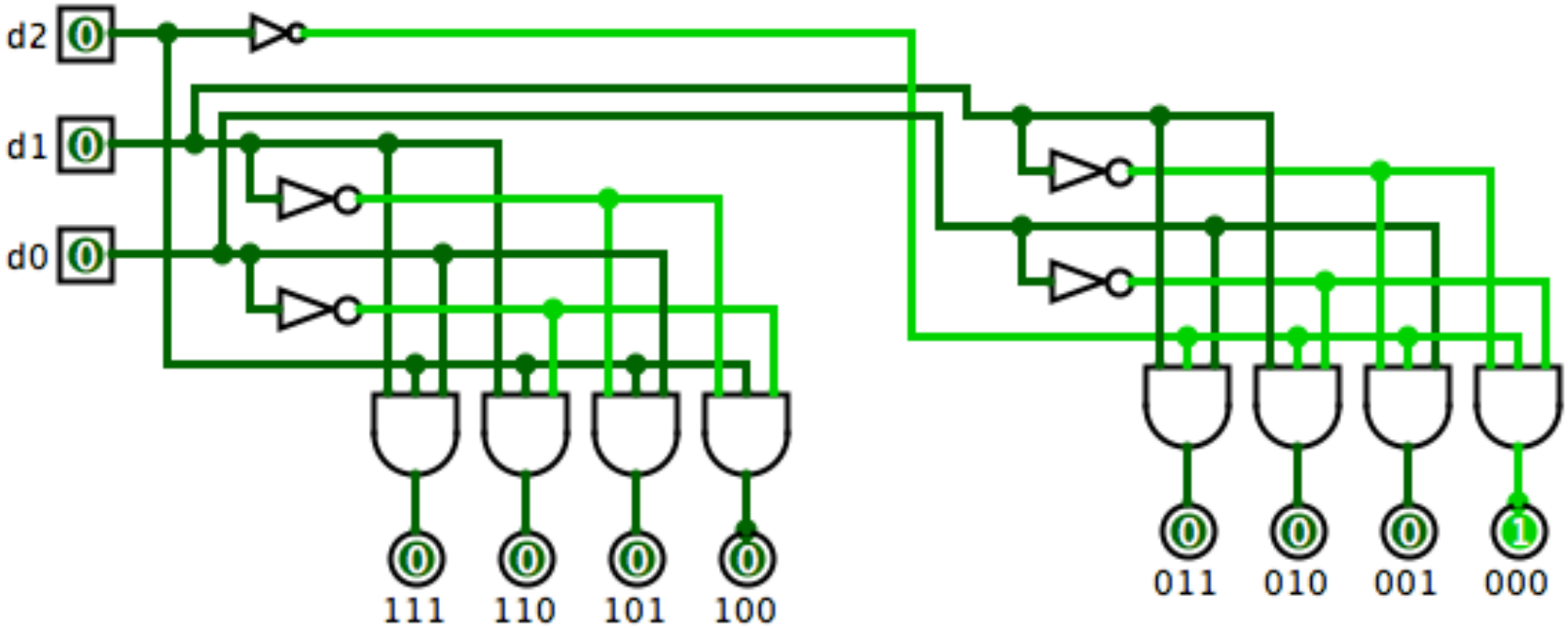
3-bit decoder





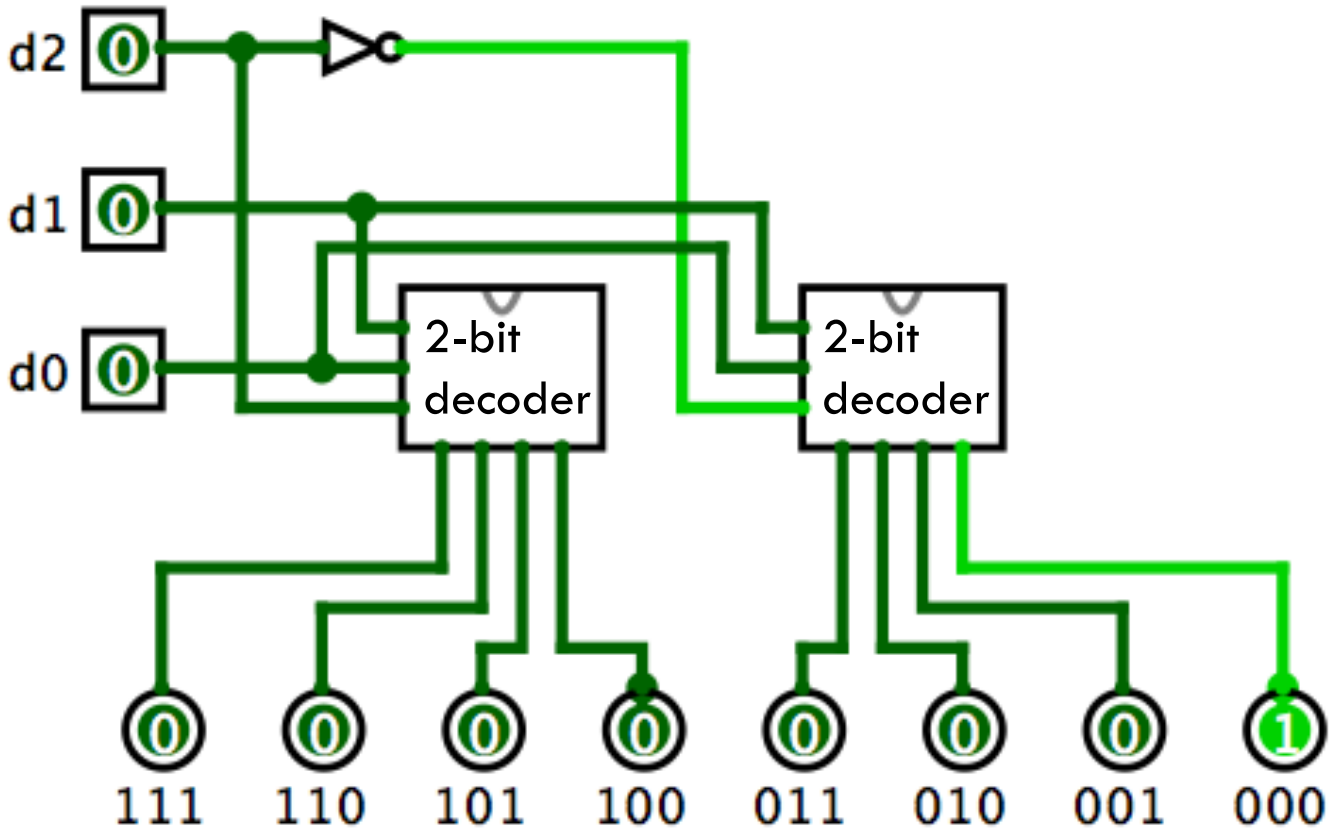
Could we use two 2-bit decoders?

3-bit decoder



d2 gets sent to the enable of the two 2-bit decoders. One as normal and one negated.

3-bit decoder using 2-bit decoders



Look at decoders in simulator

Examples

The Logisim circuit examples can be found at:

http://www.cs.pomona.edu/classes/cs51/circuits/

You can download Logism Evolution at:

https://github.com/logisim-evolution/logisim-evolution