CIRCUITS

David Kauchak
CS 52 – Spring 2017

Admin

- Assignment 5
- Assignment 6 out soon!
- Survey

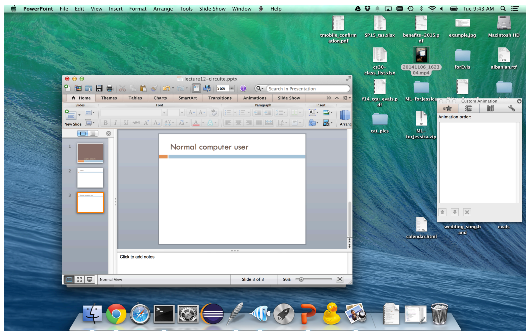
Examples

The Logisim circuit examples can be found at:

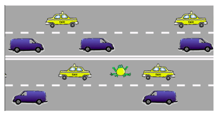
http://www.cs.pomona.edu/~dkauchak/classes/cs52/examples/logisim/

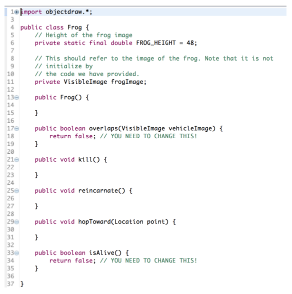
Diving into your computer

Normal computer user



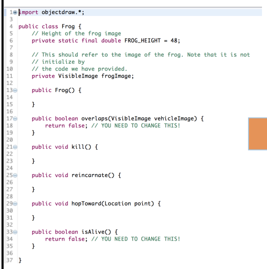
After intro CS





```java
import objectdraw.*;

public class Frog {
    // Height of the frog image
    private static final double FROG_HEIGHT = 48;

    // This should refer to the image of the frog. Note that it is not
    // initialize by
    // the code we have provided.
    private VisibleImage frogImage;

    public Frog() {

    }

    public boolean overlaps(VisibleImage vehicleImage) {
        return false; // YOU NEED TO CHANGE THIS!
    }

    public void kill() {

    }

    public void reincarnate() {

    }

    public void hopToward(Location point) {

    }

    public boolean isAlive() {
        return false; // YOU NEED TO CHANGE THIS!
    }

}
```

After 5 weeks of cs52

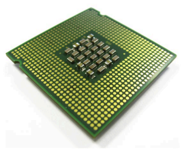


```java
import objectdraw.*;

public class Frog {
    // Height of the frog image
    private static final double FROG_HEIGHT = 48;

    // This should refer to the image of the frog. Note that it is not
    // initialize by
    // the code we have provided.
    private VisibleImage frogImage;

    public Frog() {

    }

    public boolean overlaps(VisibleImage vehicleImage) {
        return false; // YOU NEED TO CHANGE THIS!
    }

    public void kill() {

    }

    public void reincarnate() {

    }

    public void hopToward(Location point) {

    }

    public boolean isAlive() {
        return false; // YOU NEED TO CHANGE THIS!
    }

}
```

```
        loa r1 r0 0         ; get a value for a
        loa r2 r0 0         ; get a value for b
        add r3 r0 r0        ; result = 0;

        ble r1 r0 endloop   ; test if a <= 0
loop
        add r3 r3 r2        ; result += b;
        sbc r1 r1 1         ; a--;
        blt r0 r1 loop      ; return for another iteration

endloop
        sto r0 r3 0         ; write the value of product
        hlt                 ; halt
        end
```

What now?

```
        loa r1 r0 0         ; get a value for a
        loa r2 r0 0         ; get a value for b
        add r3 r0 r0        ; result = 0;

        ble r1 r0 endloop   ; test if a <= 0
loop
        add r3 r3 r2        ; result += b;
        sbc r1 r1 1         ; a--;
        blt r0 r1 loop      ; return for another iteration

endloop
        sto r0 r3 0         ; write the value of product
        hlt                 ; halt
        end
```



Quick recap

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

Quick recap

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

SML: Binary addition

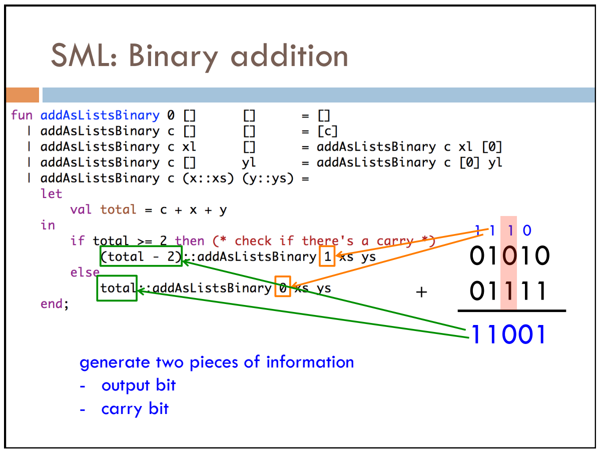
```
fun addAsListsBinary 0 []       []       = []
  | addAsListsBinary c []       []       = [c]
  | addAsListsBinary c xl       []       = addAsListsBinary c xl [0]
  | addAsListsBinary c []       yl       = addAsListsBinary c [0] yl
  | addAsListsBinary c (x::xs) (y::ys) =
    let
        val total = c + x + y
    in
        if total >= 2 then (* check if there's a carry *)
            (total - 2)::addAsListsBinary 1 xs ys
        else
            total::addAsListsBinary 0 xs ys
    end;
```

SML: Binary addition

```
fun addAsListsBinary 0 []       []       = []
  | addAsListsBinary c []       []       = [c]
  | addAsListsBinary c xl       []       = addAsListsBinary c xl [0]
  | addAsListsBinary c []       yl       = addAsListsBinary c [0] yl
  | addAsListsBinary c (x::xs) (y::ys) =
    let
        val total = c + x + y
    in
        if total >= 2 then (* check if there's a carry *)
            (total - 2)::addAsListsBinary 1 xs ys
        else
            total::addAsListsBinary 0 xs ys
    end;
```

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

handle a digit at a time

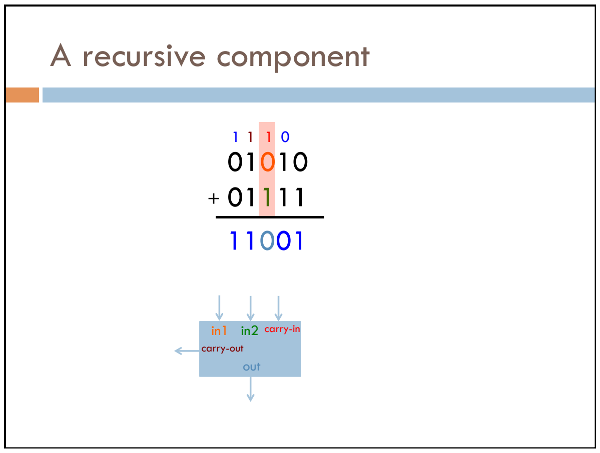


SML: Binary addition

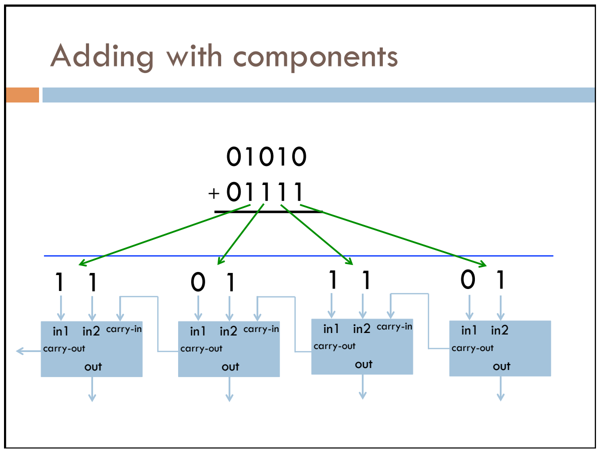
```
fun addAsListsBinary 0 []      []      = []
  | addAsListsBinary c []      []      = [c]
  | addAsListsBinary c xl      []      = addAsListsBinary c xl [0]
  | addAsListsBinary c []      yl      = addAsListsBinary c [0] yl
  | addAsListsBinary c (x::xs) (y::ys) =
    let
        val total = c + x + y
    in
        if total >= 2 then (* check if there's a carry *)
            (total - 2)::addAsListsBinary 1 xs ys
        else
            total::addAsListsBinary 0 xs ys
    end;
```

1 1 1 0
01010
+ 01111
11001

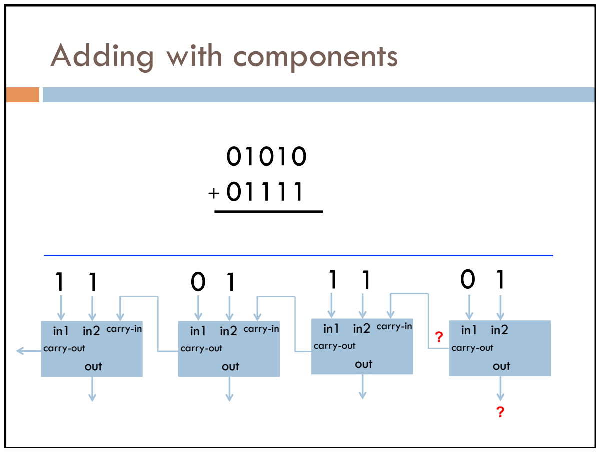
generate two pieces of information
- output bit
- carry bit



A recursive component



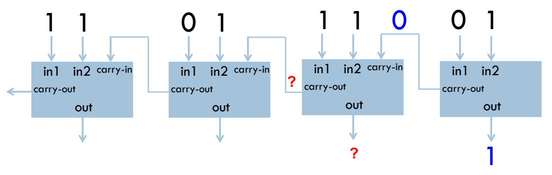
Adding with components



Adding with components

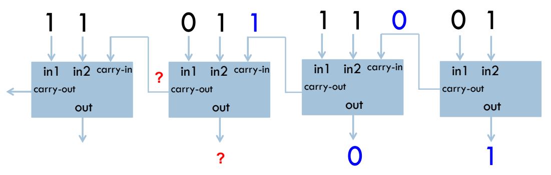
Adding with components

```
    0
 01010
+01111
     1
```



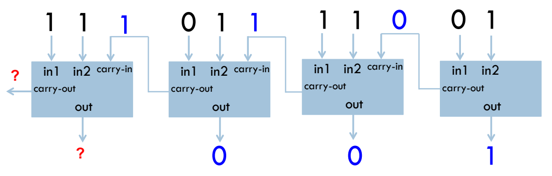
Adding with components

```
   10
 01010
+01111
    01
```



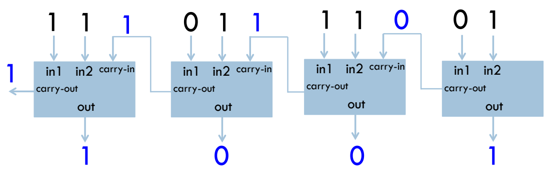
Adding with components

```
  110
 01010
+01111
   001
```

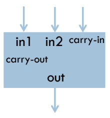


Adding with components

```
 1110
 01010
+01111
 11001
```

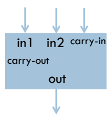


Implementing the component



What goes on inside the component?

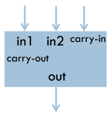
Implementing the component



```
let
    val total = c + x + y
in
    if total >= 2 then (* check if there's a carry *)
        (total - 2)::addAsListsBinary 1 xs ys
    else
        total::addAsListsBinary 0 xs ys
end;
```

Current implementation uses addition!

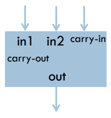
Implementing the component



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

Another implementation

```
fun addAsListsBinary 0 []      []      = []
  | addAsListsBinary c []      []      = [c]
  | addAsListsBinary c xl      []      = addAsListsBinary c xl [0]
  | addAsListsBinary c []      yl      = addAsListsBinary c [0] yl
  | addAsListsBinary c (x::xs) (y::ys) =
    if x = 1 andalso y = 1 andalso c = 1 then
        1::(addAsListsBinary 1 xs ys)
    else if (x = 1 andalso y = 1) orelse
            (x = 1 andalso c = 1) orelse
            (y = 1 andalso c = 1) then
        0::(addAsListsBinary 1 xs ys)
    else if x = 1 orelse y = 1 orelse c = 1 then
        1::(addAsListsBinary 0 xs ys)
    else
        0::(addAsListsBinary 0 xs ys);
```

- Don't use addition anymore
- Translated the problem into a boolean logic problem

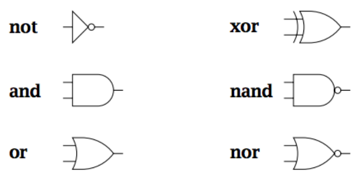
What are some boolean operators?

A	B	A and B	A or B	not A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

What are some boolean operators?

A	B	A and B	A or B	not A	A nand B	A nor B	A xor B
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0

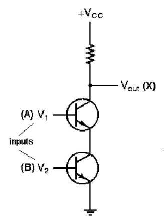
Gates

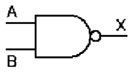


Gates have inputs and outputs
- values are 0 or 1

They are **hardware** components!

Gates as hardware

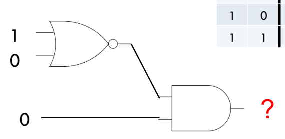


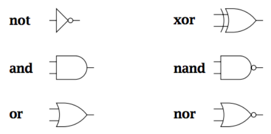


A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Utilizing gates

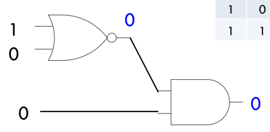
A	B	A and B	A or B	not A	A nand B	A nor B	A xor B
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0

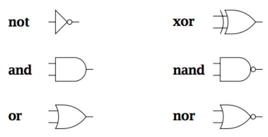




Utilizing gates

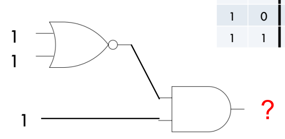
A	B	A and B	A or B	not A	A nand B	A nor B	A xor B
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0

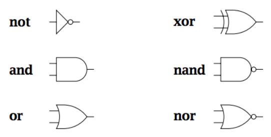




Utilizing gates

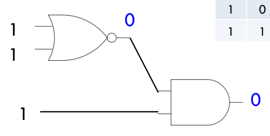
A	B	A and B	A or B	not A	A nand B	A nor B	A xor B
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0



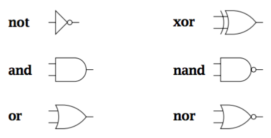


Utilizing gates

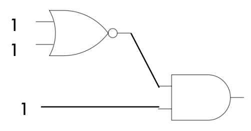
A	B	A and B	A or B	not A	A nand B	A nor B	A xor B
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0



When is this circuit 1?



Utilizing gates



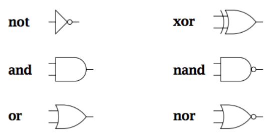
in1	in2	in3	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Designing more interesting circuits

A	B	A and B	A or B	not A	A nand B	A nor B	A xor B
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0

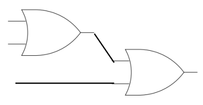
in1	in2	in3	OUT
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

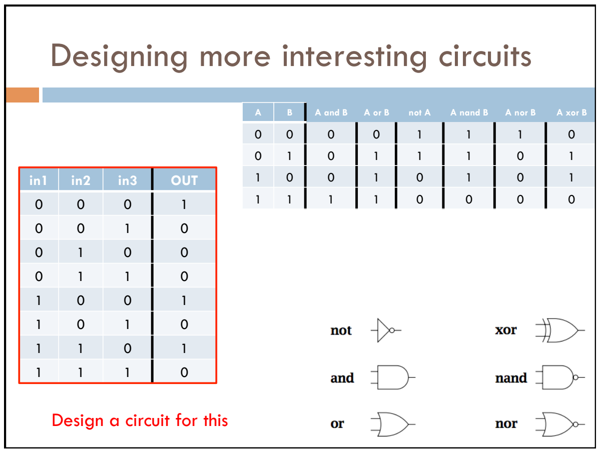
Design a circuit for this



Designing more interesting circuits

in1	in2	in3	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

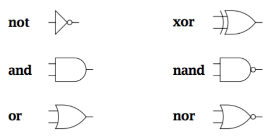




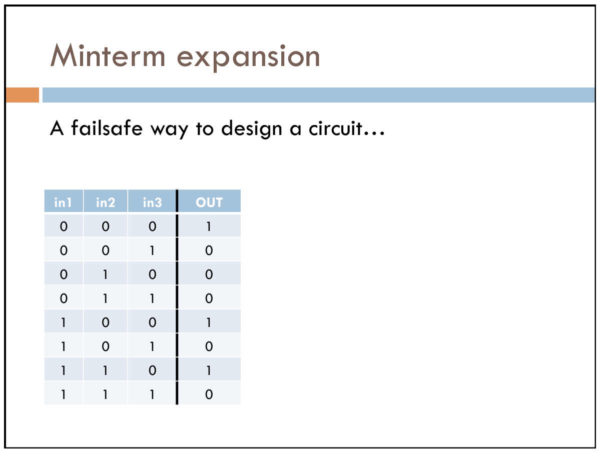
Designing more interesting circuits

in1	in2	in3	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

A	B	A and B	A or B	not A	A nand B	A nor B	A xor B
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0



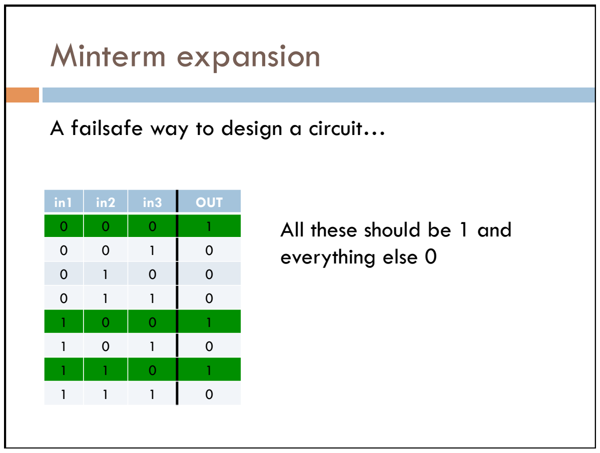
Design a circuit for this



Minterm expansion

A failsafe way to design a circuit…

in1	in2	in3	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

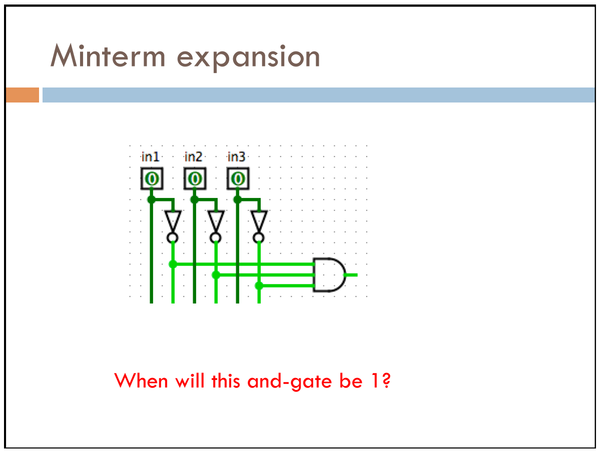


Minterm expansion

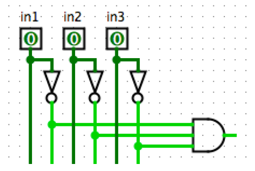
A failsafe way to design a circuit…

in1	in2	in3	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

All these should be 1 and everything else 0

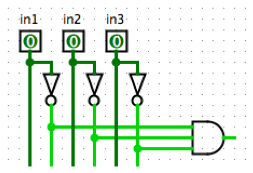


Minterm expansion



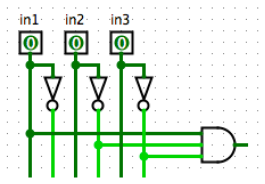
When will this and-gate be 1?

Minterm expansion



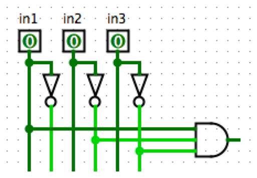
Only when in1=0, in2=0, in3=0

Minterm expansion



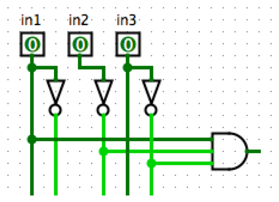
When will this and-gate be 1?

Minterm expansion



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

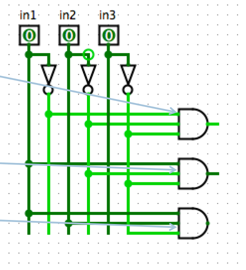
Minterm expansion



Does this help us?

Minterm expansion

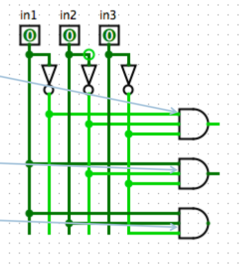
in1	in2	in3	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



Only these three inputs can be true!

Minterm expansion

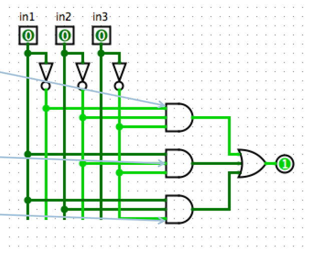
in1	in2	in3	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



How do we combine these?

Minterm expansion

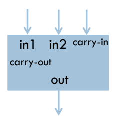
in1	in2	in3	OUT
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



Or-gate!

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

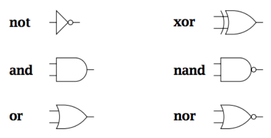
A half-adder: no carry-in

A	B	A and B	A or B	not A	A nand B	A nor B	A xor B
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0

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

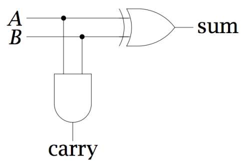
Hint: solve each output bit independently

Design a circuit for this



A half-adder: no carry-in

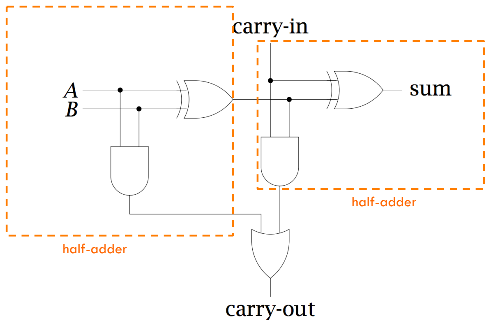
A	B	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



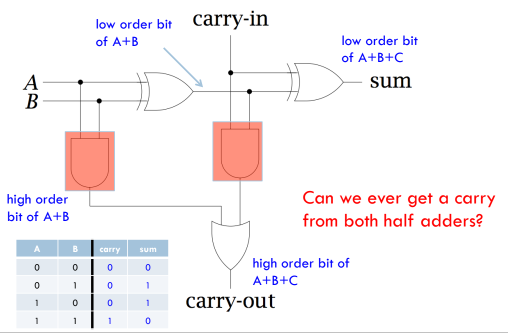
low order bit of A+B

higher order bit of A+B

Implementing a full adder



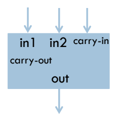
Implementing a full adder



Can we ever get a carry from both half adders?

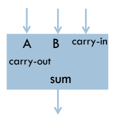
A	B	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

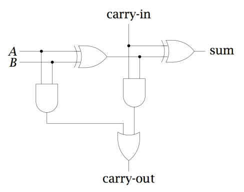
Implementing the component



What goes on inside the component?

Implementing the component





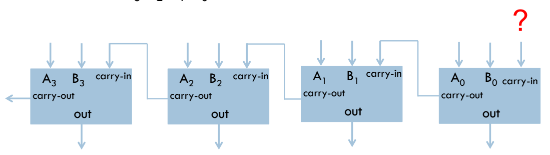
Ripple carry adder

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



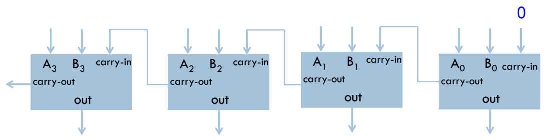
Ripple carry adder

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

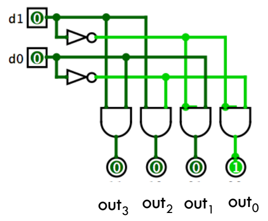


Look at ripple carry adder example

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

Simulator basics

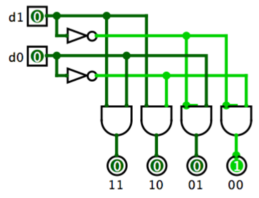
Mystery circuit



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?

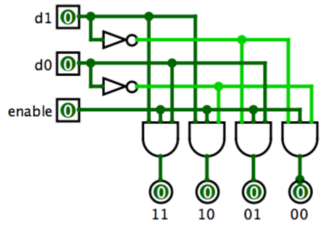
2-bit decoder



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

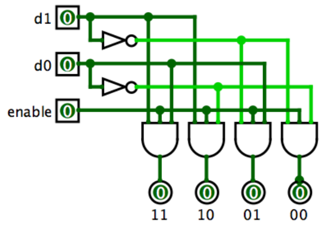
Sends '1' along one of the output lines

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

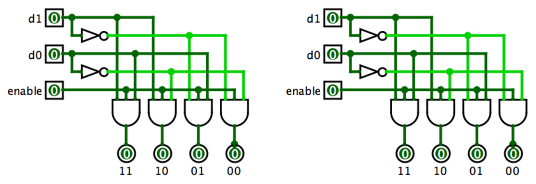
3 inputs

8 output lines

Could make from scratch

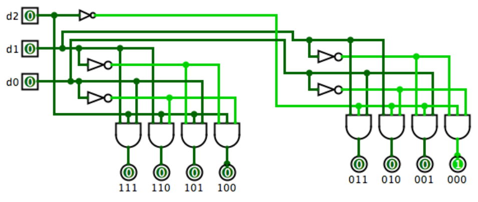
Better idea: reuse 2-bit decoders

3-bit decoder



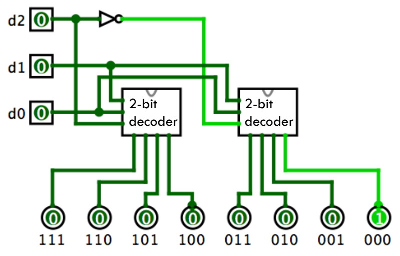
Could we use two 2-bit decoders?

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

Barrel shifters

Examples

The Logisim circuit examples can be found at:

http://www.cs.pomona.edu/~dkauchak/classes/cs52/examples/logisim/