Lecture5

Page 1

Skills:

① Design an adder.

Outline

① Discuss the adder

② Review number representation.

③ Masks

Add two binary

Constrain we only want to use logic operations

Lable the position of the binary numbers

Input 1

x_n x_3 x_2 x_1 x_0

0 ..... 0 0 0 0

Input 2

y_n... y_4 y_3 y_2 y_1 y_0

0 0 0 0 0 0

output

z_n .... z_3 z_2 z_1 z_0

0 0 0 0 0

Let's consider the case of add two binary number that are 1 bit long

$x_0 + y_0$

Let write ~~of~~ out the truth table for this addition

x_0	y_0	z_0	c_0
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	1

Special case because the result is 10

we call this extra one the carry bit

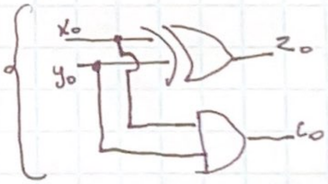
Question can we design a circuit that outputs

z_0

$x_0 \oplus y_0 = z_0$

xor

could you design



* Gate reminder

But what about the carry bit

$x_0 \wedge y_0 = c_0$

What about binary number that are longer than 2 bits

Consider case with two binary number

carry (out) ← carry (in)

c_2 ↗ c_1

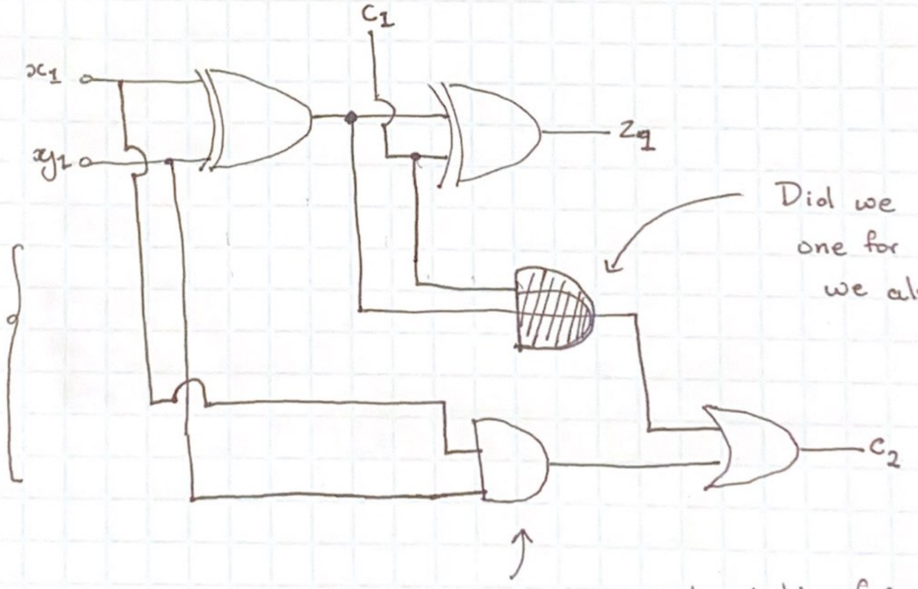
```
   1
  1 11
 + 11
 ----
  110
```

Example 2.1

Let's write out the truth table.

c_1	x_1	y_2	z_1	c_2
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

← The case we looked from example 2.1

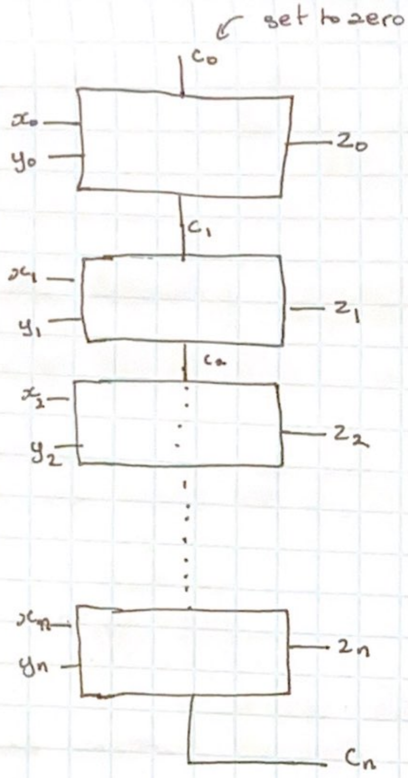


What about c_2

Did we calculate one for z b/c did we also have a carry

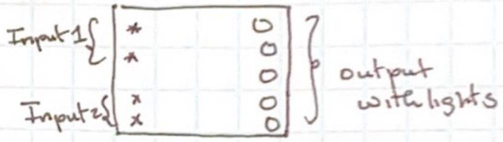
same carry calculation ~~form~~ from before.

Then how do we scale to number of bits



Commonly reffered to as a ripple carry adder.

Now we final have our adding box



- review of ~~two~~ bitwise operators. & Masking
- Maybe sum bit puzzles.

Operators.

&	Bitwize and
|	Bitwize or
^	Bitwize xor.
>>	Bit wize to the right no sign extension
<<	Bitwize shift to the left no sign extension.

Masking

$00\underbrace{111}0000$ construct a sequence of one & zero that you can use to extract or flip information / binary values

Consider the following <u>examples.</u>

x = 110111 ← Set the forth bit of this number to one

x | 0010000

↖ we could $\overset{\text{or}}{\ }$ $\overset{\text{with}}{\ }$ the following mask,

```
  110111
or001000
  ------
  11 1111
```

↖ we've~~se~~ set the forth bit to one.

How could we construct the mask

~~y=~~ 001000

↓

~~y=~~ 1<<4

x |= 1<<4 ← Set the forth bit of x to one

x = x|<<4

<u>Clear</u> (we normal use ands)

x = 1111

clear the first, thrid and forth bits

we could do that by anding with 0010

```
 1111
^0010
 ----
 0010
```

x &|= 1<<1

Clear (and)

x = 1111 clear just the second bit

we could and with 1101

$$\begin{array}{r} \cancel{0010} \\ 1101 \\ \wedge\,1111 \\ \hline 1101 \end{array}$$

↖ Just clears the second bit

How do we create
the mask 1101

y = 0001

~~y<<1y~~ = 0010 ← y<<1

~y = 1101 ← ¬y (not of y) write ~y in c.

Use or to set
Use and to clear

But how would you ~~flip~~ flip.

1101 → 1111
1111 → 1101 } Think about this for a second.

mask = 0010
x = 1101

$$x \wedge \text{mask} = \begin{array}{r} 1101 \\ \oplus\,0010 \\ \hline 1111 \end{array}$$

↖ flipped

~~mask~~ ~~1101~~
mask = 0010
x = 1111

$$x \wedge \text{mask} = \begin{array}{r} 0010 \\ \oplus\,1111 \\ \hline 1101 \end{array}$$

↑
flipped

Parity bit

even odd.

0110 1 0 1

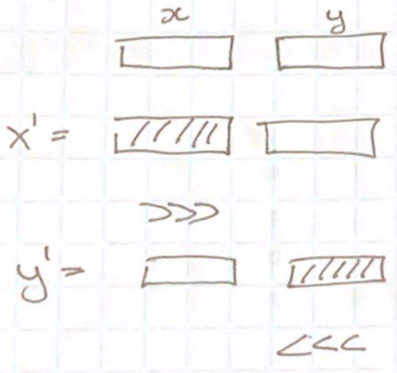
~~even parity~~ number

parity { 0 if even number of 1
1 if odd number of 1

0
1

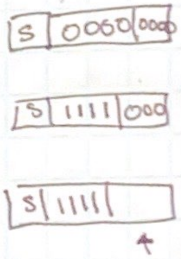
```
parity = 0
    repeat 32
        parity ^= (x & 1)
        x>>=1
```

Puzzle.



or together.

*0 = 1
!0 = 0



two complement
flip +1
add.