University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

A Power-of-Two Checker

Powers of Two are Easy to Spot in Binary

Let's do another bit-sliced design.

Can we check whether an unsigned number represents a power of two?

What does a power of two look like in bits?

For **5-bit unsigned**, the powers of 2 are...

`00001`, `00010`, `00100`, `01000`, `10000`

A power of two has exactly one 1 bit
(with place value 2^N for some N, of course!).

The Answers are Not Always Enough

So our design will answer the following:
Is $A = a_{N-1}a_{N-2}...a_1a_0$ a power of two?

How many answers are possible?

Two: Yes, and No.

A trick question:

How many bits do we need to pass between slices?

That's right: TWO bits.

What Extra Information Do We Need?

Why not just one? An answer only needs 1 bit!

Say that we pass bits from right to left.

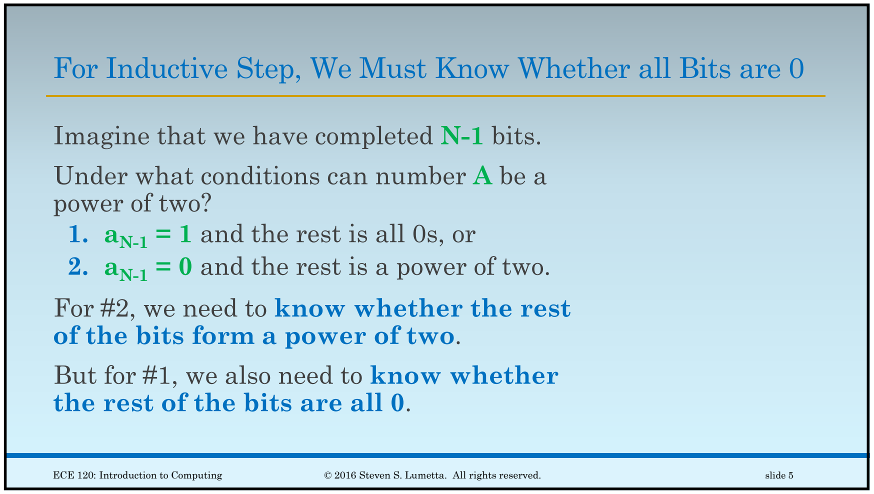
If the bits **$a_{N-2}...a_1a_0$** represent a power of two,
is $a_{N-1}a_{N-2}...a_1a_0$ be a power of two?

Iff a_{N-1} = 0.

What if **$a_{N-2}...a_1a_0$** does **not** represent a power of two?

In that case, **we can't tell whether $a_{N-1}a_{N-2}...a_1a_0$ is a power of two or not**!

What else do we need to know?



For Inductive Step, We Must Know Whether all Bits are 0

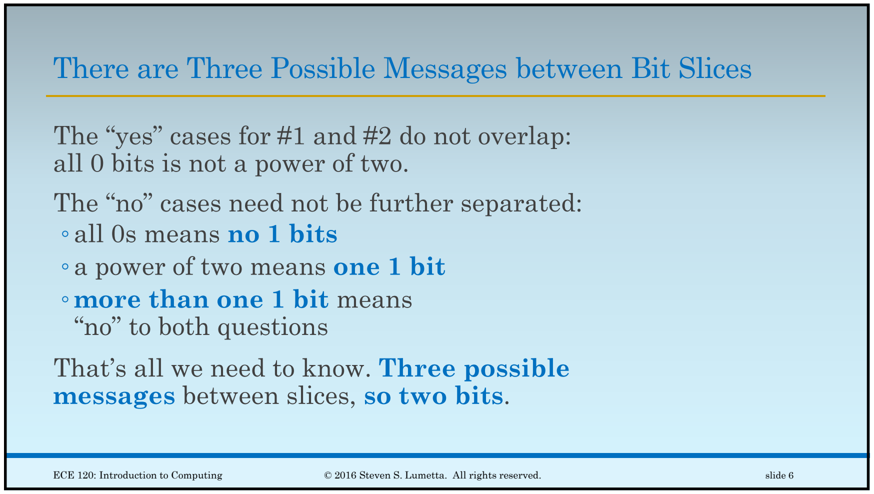
Imagine that we have completed **N-1** bits.

Under what conditions can number **A** be a power of two?

1. $a_{N-1} = 1$ and the rest is all 0s, or
2. $a_{N-1} = 0$ and the rest is a power of two.

For #2, we need to **know whether the rest of the bits form a power of two**.

But for #1, we also need to **know whether the rest of the bits are all 0**.



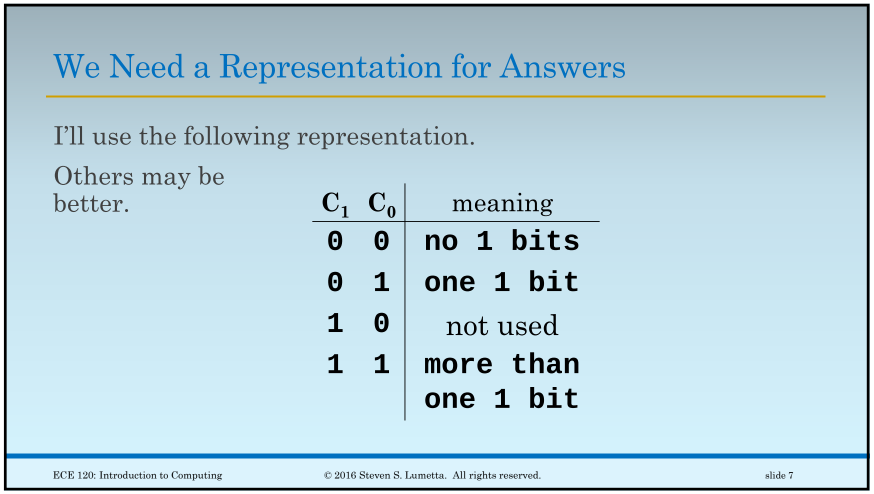
There are Three Possible Messages between Bit Slices

The "yes" cases for #1 and #2 do not overlap: all 0 bits is not a power of two.

The "no" cases need not be further separated:

- all 0s means **no 1 bits**
- a power of two means **one 1 bit**
- **more than one 1 bit** means "no" to both questions

That's all we need to know. **Three possible messages** between slices, **so two bits**.

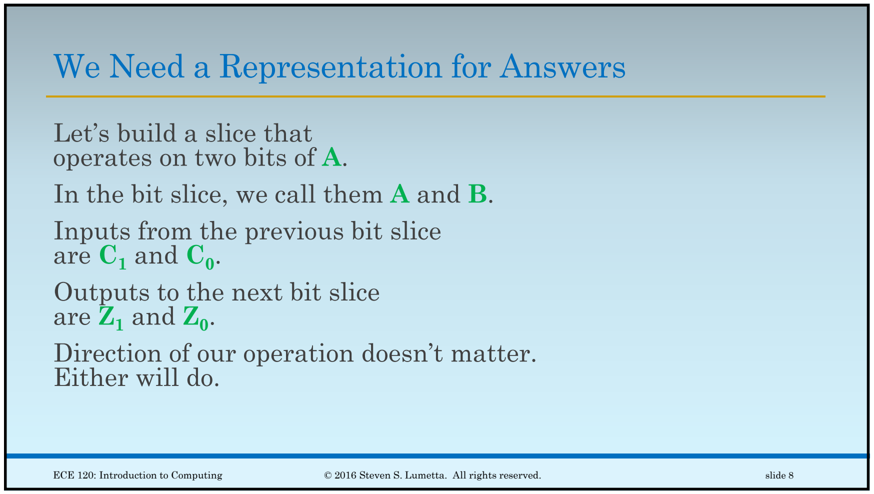


We Need a Representation for Answers

I'll use the following representation.

Others may be better.

C_1	C_0	meaning
0	0	no 1 bits
0	1	one 1 bit
1	0	not used
1	1	more than one 1 bit



We Need a Representation for Answers

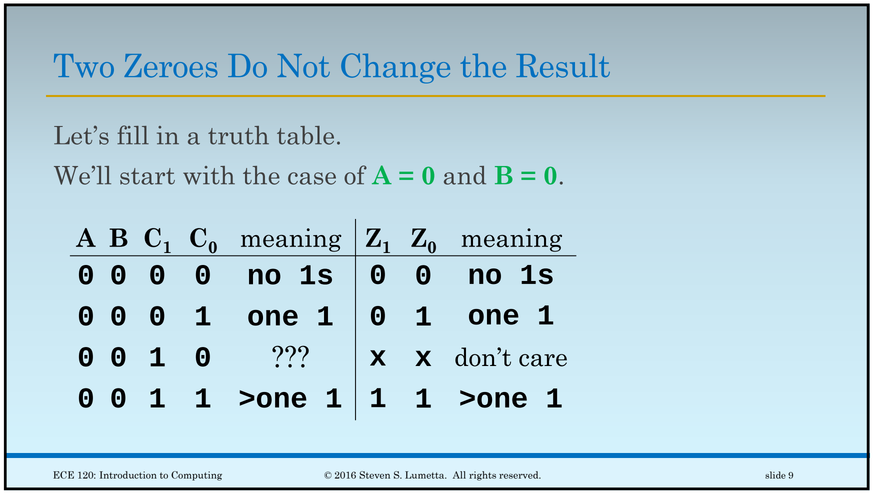
Let's build a slice that operates on two bits of **A**.

In the bit slice, we call them **A** and **B**.

Inputs from the previous bit slice are C_1 and C_0.

Outputs to the next bit slice are Z_1 and Z_0.

Direction of our operation doesn't matter. Either will do.

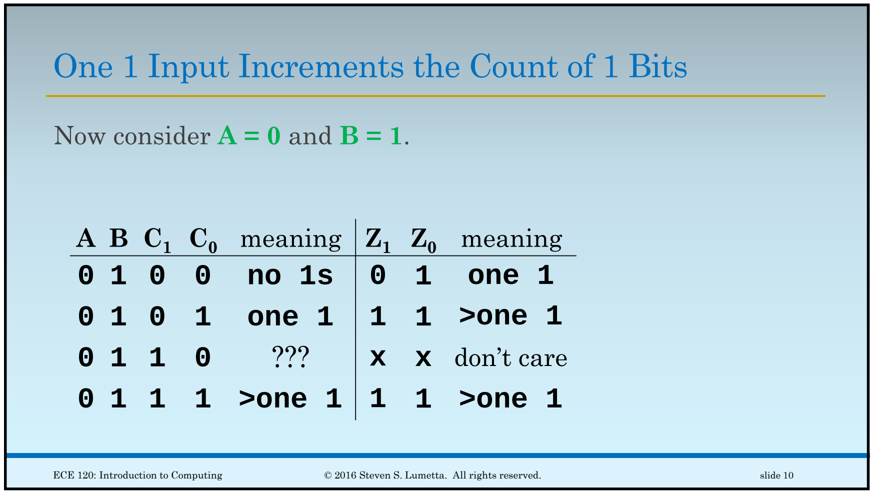


Two Zeroes Do Not Change the Result

Let’s fill in a truth table.

We’ll start with the case of **A = 0** and **B = 0**.

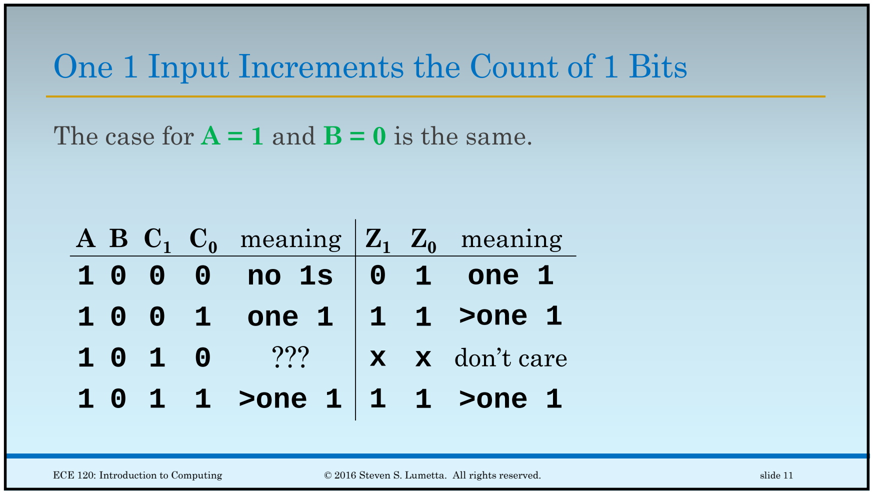
A	B	C_1	C_0	meaning	Z_1	Z_0	meaning
0	0	0	0	no 1s	0	0	no 1s
0	0	0	1	one 1	0	1	one 1
0	0	1	0	???	x	x	don’t care
0	0	1	1	>one 1	1	1	>one 1



One 1 Input Increments the Count of 1 Bits

Now consider **A = 0** and **B = 1**.

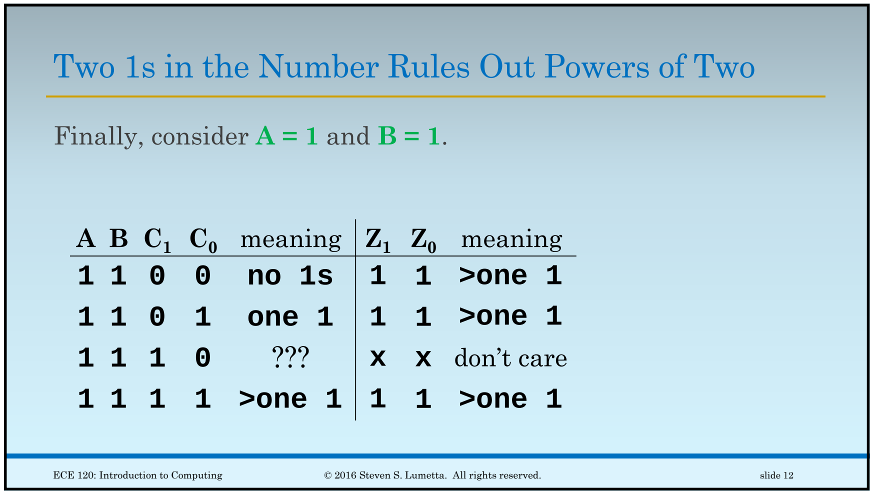
A	B	C_1	C_0	meaning	Z_1	Z_0	meaning
0	1	0	0	no 1s	0	1	one 1
0	1	0	1	one 1	1	1	>one 1
0	1	1	0	???	x	x	don’t care
0	1	1	1	>one 1	1	1	>one 1



One 1 Input Increments the Count of 1 Bits

The case for **A = 1** and **B = 0** is the same.

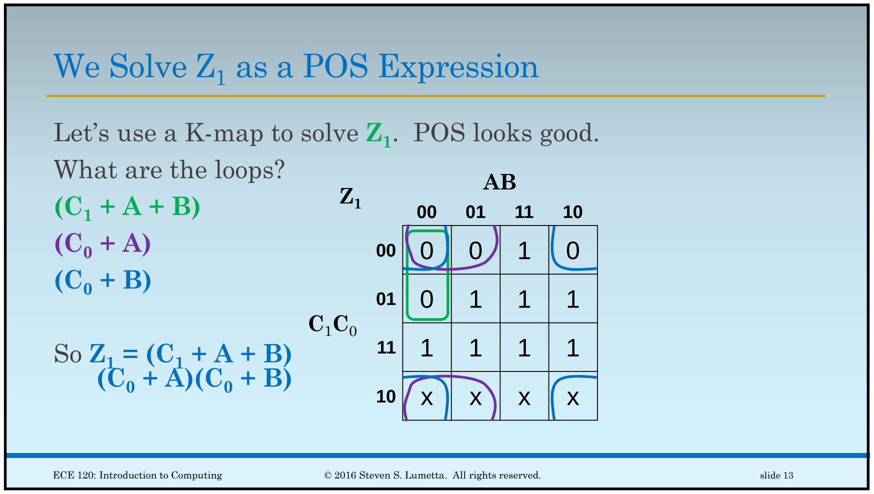
A	B	C_1	C_0	meaning	Z_1	Z_0	meaning
1	0	0	0	no 1s	0	1	one 1
1	0	0	1	one 1	1	1	>one 1
1	0	1	0	???	x	x	don’t care
1	0	1	1	>one 1	1	1	>one 1



Two 1s in the Number Rules Out Powers of Two

Finally, consider **A = 1** and **B = 1**.

A	B	C_1	C_0	meaning	Z_1	Z_0	meaning
1	1	0	0	no 1s	1	1	>one 1
1	1	0	1	one 1	1	1	>one 1
1	1	1	0	???	x	x	don’t care
1	1	1	1	>one 1	1	1	>one 1



We Solve Z_1 as a POS Expression

Let's use a K-map to solve Z_1. POS looks good.

What are the loops?

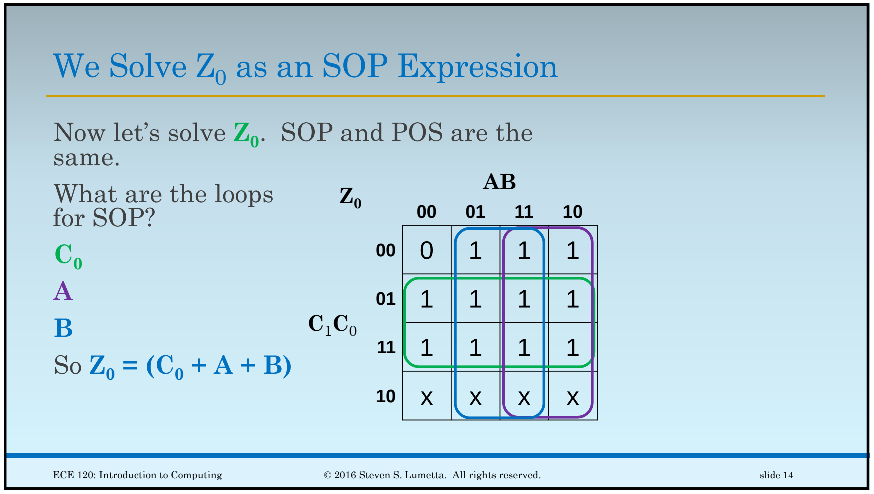
$(C_1 + A + B)$

$(C_0 + A)$

$(C_0 + B)$

So $Z_1 = (C_1 + A + B)(C_0 + A)(C_0 + B)$

Z_1	AB 00	01	11	10
C_1C_0 00	0	0	1	0
01	0	1	1	1
11	1	1	1	1
10	x	x	x	x



We Solve Z_0 as an SOP Expression

Now let's solve Z_0. SOP and POS are the same.

What are the loops for SOP?

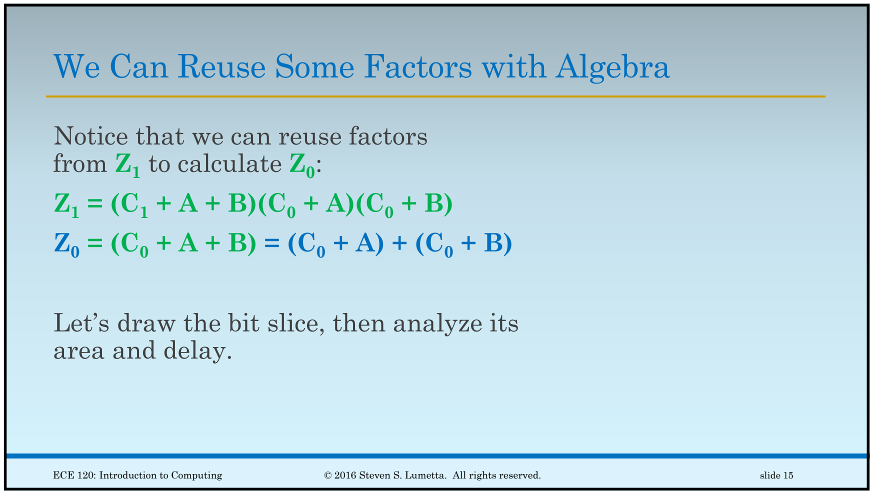
C_0

A

B

So $Z_0 = (C_0 + A + B)$

Z_0	AB 00	01	11	10
C_1C_0 00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	x	x	x	x



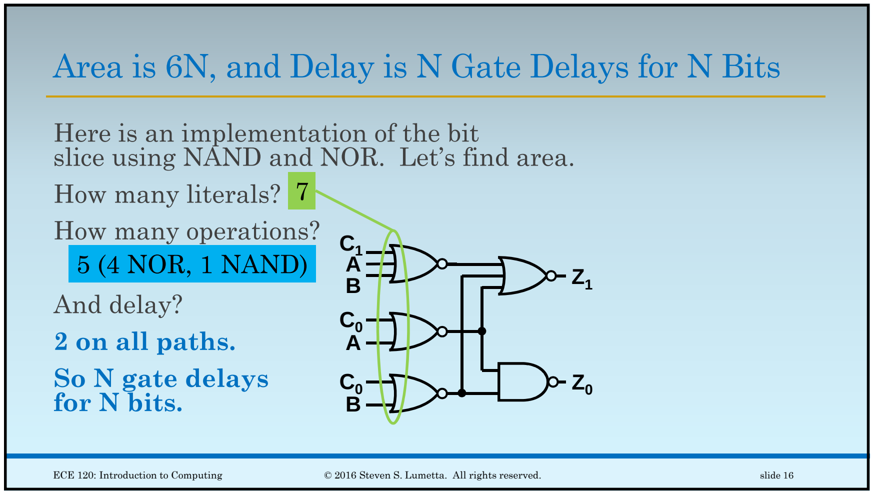
We Can Reuse Some Factors with Algebra

Notice that we can reuse factors from Z_1 to calculate Z_0:

$Z_1 = (C_1 + A + B)(C_0 + A)(C_0 + B)$

$Z_0 = (C_0 + A + B) = (C_0 + A) + (C_0 + B)$

Let's draw the bit slice, then analyze its area and delay.



Area is 6N, and Delay is N Gate Delays for N Bits

Here is an implementation of the bit slice using NAND and NOR. Let's find area.

How many literals? 7

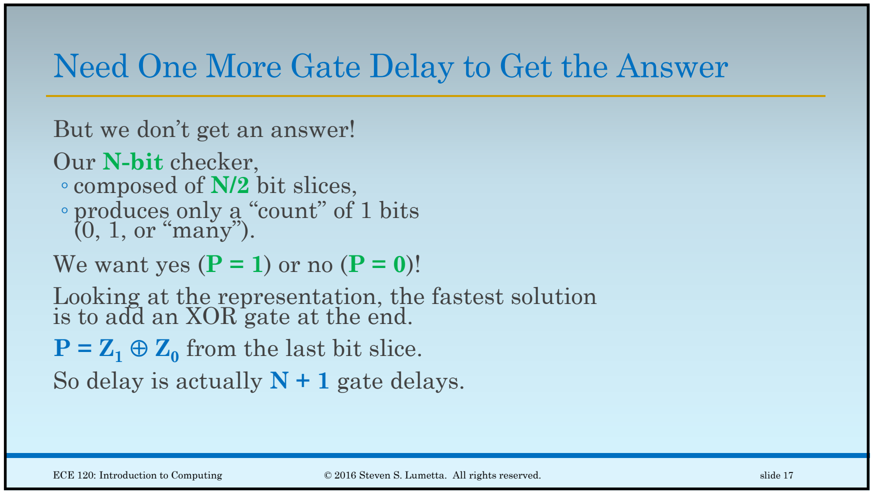
How many operations?

5 (4 NOR, 1 NAND)

And delay?

2 on all paths.

So N gate delays for N bits.



Need One More Gate Delay to Get the Answer

But we don't get an answer!

Our **N-bit** checker,

- composed of **N/2** bit slices,
- produces only a "count" of 1 bits (0, 1, or "many").

We want yes (**P = 1**) or no (**P = 0**)!

Looking at the representation, the fastest solution is to add an XOR gate at the end.

$\mathbf{P = Z_1 \oplus Z_0}$ from the last bit slice.

So delay is actually **N + 1** gate delays.