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

ECE 120: Introduction to Computing

Bit-Sliced Comparator

How Do You Compare Unsigned Numbers?

Let's develop a bit-sliced design to
compare two unsigned numbers.

Which 8-bit unsigned number is bigger?

```
0 1 1 0 1 0 0 0
0 1 0 1 0 1 1 1
```

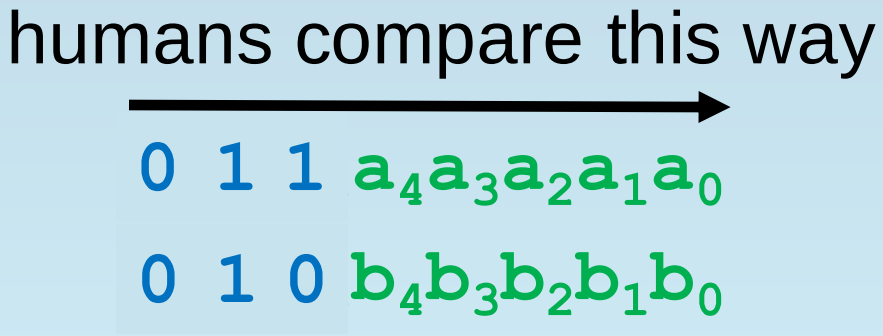
How did you know?

Did you start on the left or the right?

Humans Go from Left to Right

Usually, humans start on the left. Why?

As soon as we notice a difference, we're done!



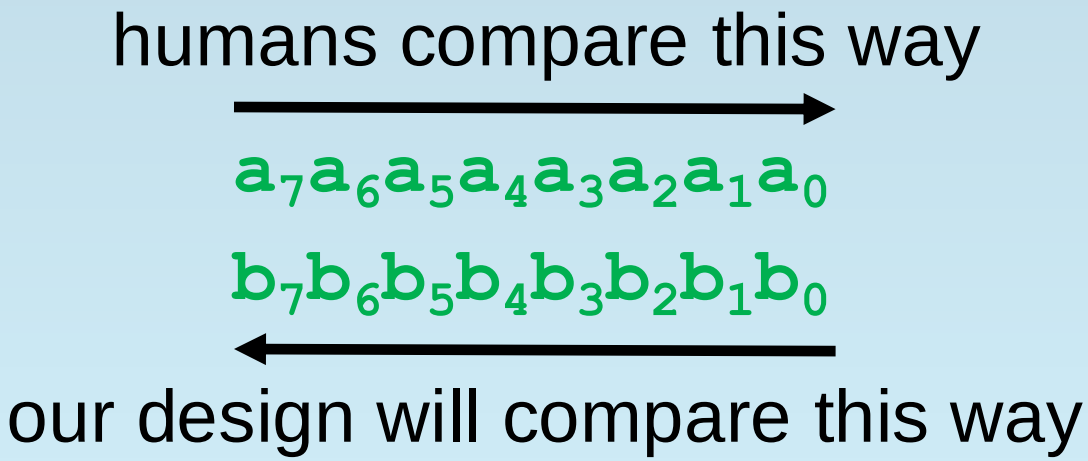
Bit-sliced hardware cannot stop in the middle.

The information flows from one end to the other.

Output wires produce the answer (in bits).

Our Design Compares from Right to Left

Our comparator design will start on the right.



From least significant to most significant bit.

Three Possible Answers for Comparison of A and B

When comparing two numbers, **A** and **B**,
we have **three possible outcomes**:

A < B

A = B

A > B

To decide the answer for **N+1** bits, we need:

- the answer for **N** (less significant) bits,
- one bit of **A**, and
- one bit of **B**.

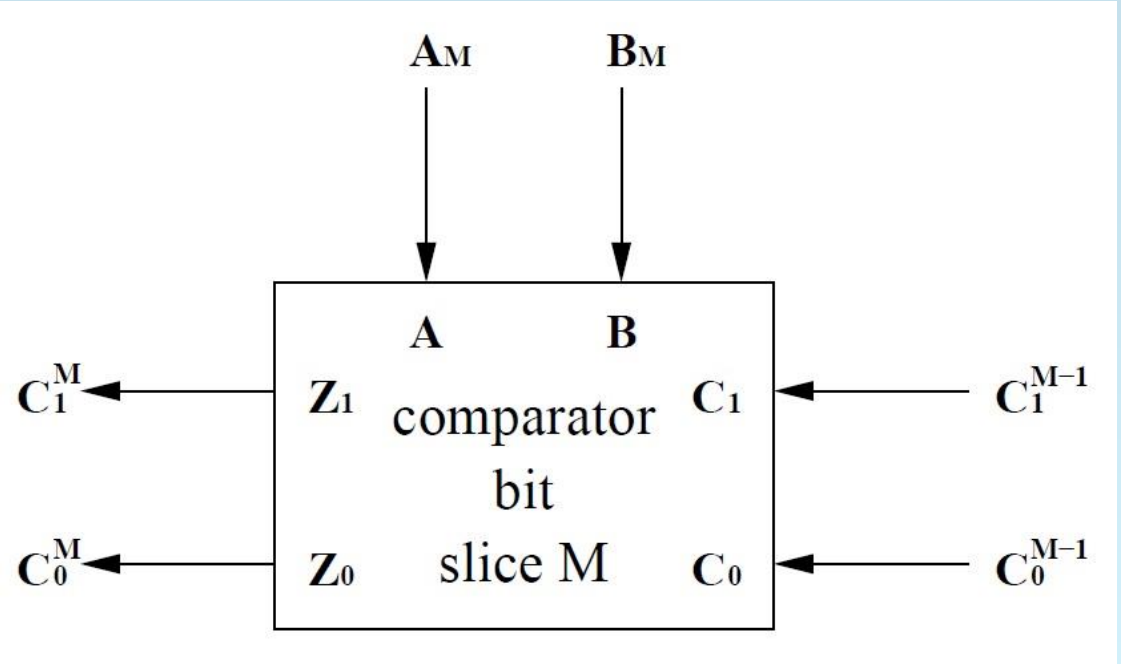
An Abstract Model of the Comparator Bit Slice

A question for you:

How many bits must pass between slices?

Two!

This figure shows an abstract model of our bit slice.



We Need a Representation for Answers

Another question for you:

How do we represent the three possible answers?

Any way we want!

Our choice of representation will affect the amount of logic we need.

Here's a good one…

C_1	C_0	meaning
0	0	A = B
0	1	A < B
1	0	A > B
1	1	not used

A Single Bit Requires Two Minterms on A, B

Let's start by solving a single bit.

In this case, there are no less significant bits.

So we consider only **A** and **B**.

Fill in the meanings, then the bits.

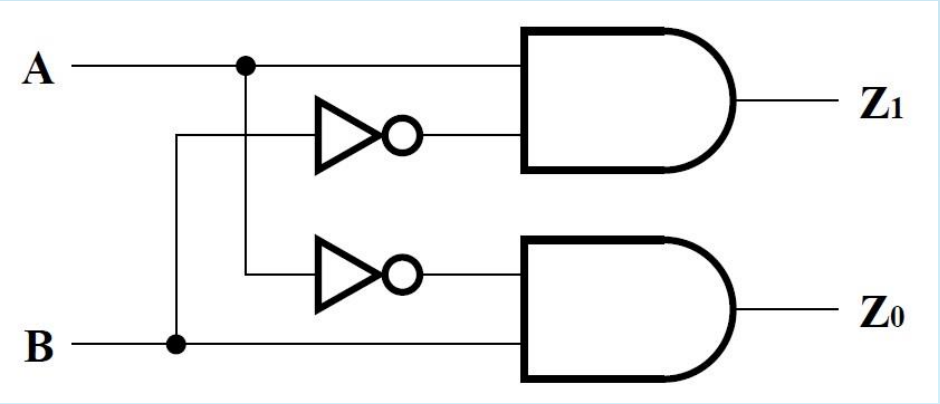
Note that Z_1 and Z_0 are minterms.

A	B	Z_1	Z_0	meaning
0	0	0	0	A = B
0	1	0	1	A < B
1	0	1	0	A > B
1	1	0	0	A = B

Comparing Two Bits is Fairly Easy

An implementation for a single bit appears below.

This structure forms the core of our bit slice, since it compares one bit of **A** with one bit of **B**.



When A and B are Equal, Pass Along the Answer

Now for the full problem.

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	A = B	0	0	A = B
0	0	0	1	A < B	0	1	A < B
0	0	1	0	A > B	1	0	A > B
0	0	1	1	???	x	x	don't care

When A and B are Equal, Pass Along the Answer

Is there any difference when **A = 1** and **B = 1**?

No, outputs are the same as the last case.

A	B	C_1	C_0	meaning	Z_1	Z_0	meaning
1	1	0	0	A = B	0	0	A = B
1	1	0	1	A < B	0	1	A < B
1	1	1	0	A > B	1	0	A > B
1	1	1	1	???	x	x	don't care

When A and B Differ, Override the Previous Answer

What about case of **A = 0** and **B = 1**?

Always output A < B (for valid inputs).

A	B	C_1	C_0	meaning	Z_1	Z_0	meaning
0	1	0	0	A = B	0	1	A < B
0	1	0	1	A < B	0	1	A < B
0	1	1	0	A > B	0	1	A < B
0	1	1	1	???	x	x	don't care

When A and B Differ, Override the Previous Answer

And the case of **A = 1** and **B = 0**?

Always output A > B (for valid inputs).

A	B	C_1	C_0	Meaning	Z_1	Z_0	meaning
1	0	0	0	A = B	1	0	A > B
1	0	0	1	A < B	1	0	A > B
1	0	1	0	A > B	1	0	A > B
1	0	1	1	???	x	x	don't care

Z_1 is a Majority Function

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

What are the loops?

AB'

AC_1

$B'C_1$

So

$Z_1 = AB' + AC_1 + B'C_1$

Z_1

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

Z_0 is Also a Majority Function

Now let's use a K-map to solve Z_0.

What are the loops?

A'B

A'C_0

BC_0

So

Z_0 = A'B + A'C_0 + BC_0

Z_0

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

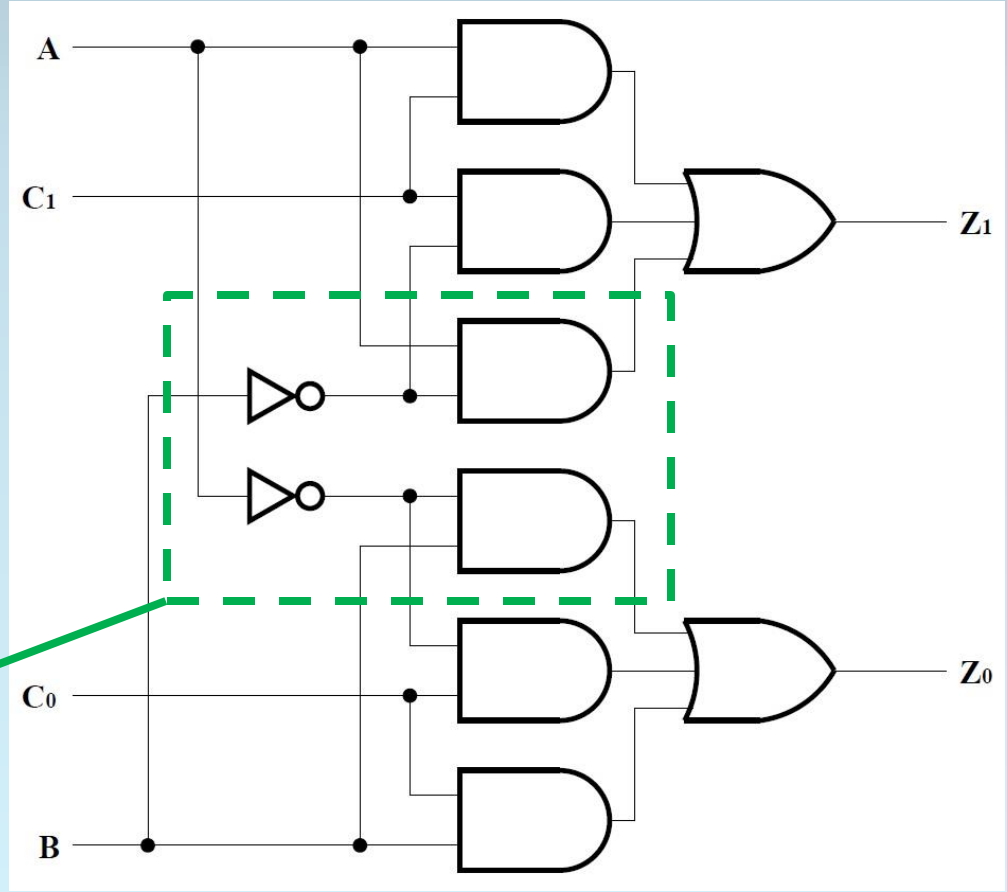
Full Implementation as SOP Expressions

$Z_1 = AB' + AC_1 + B'C_1$

$Z_0 = A'B + A'C_0 + BC_0$

Notice the symmetry.

And the single-bit core.



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

ECE 120: Introduction to Computing

Analyzing and Optimizing the Bit-Sliced Comparator

Area Heuristic for One Comparator Bit Slice is 20

Let's analyze area and delay.

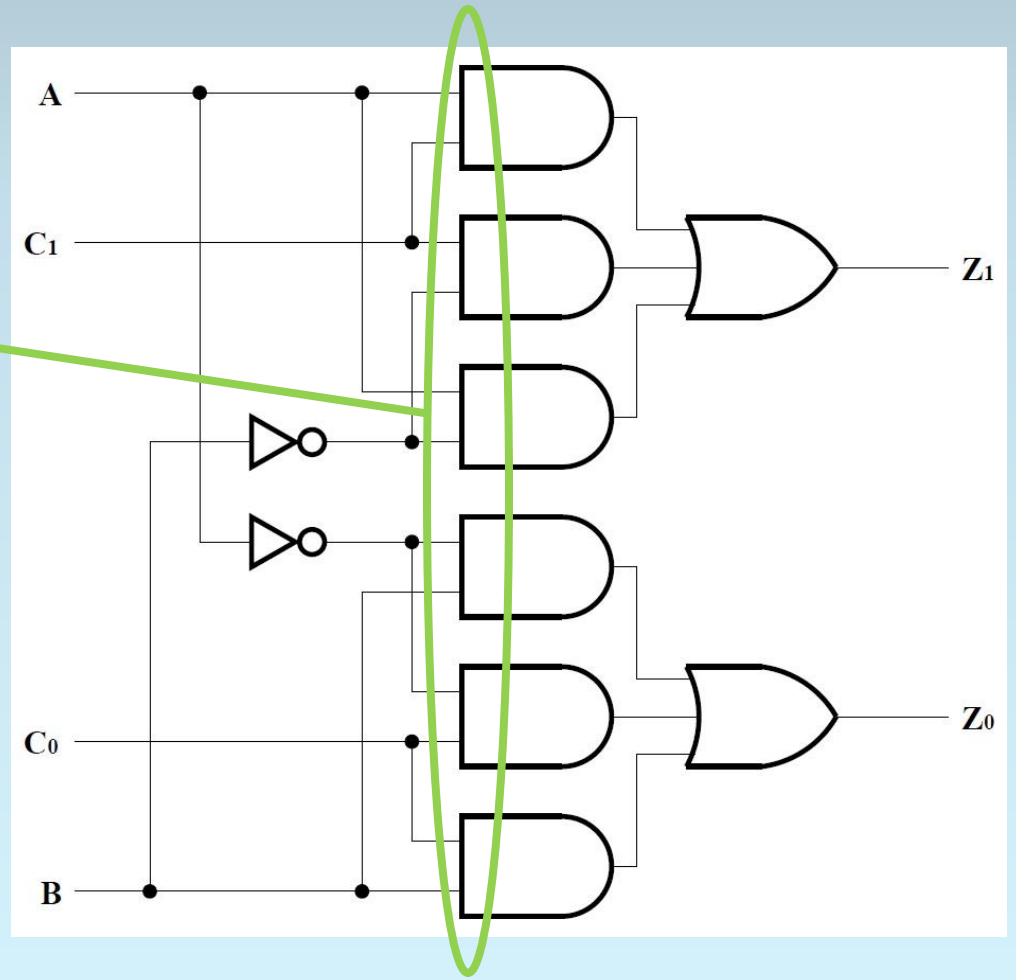
How many literals?

12

How many operators?

8 (6 AND, 2 OR)

So **area is 20**.



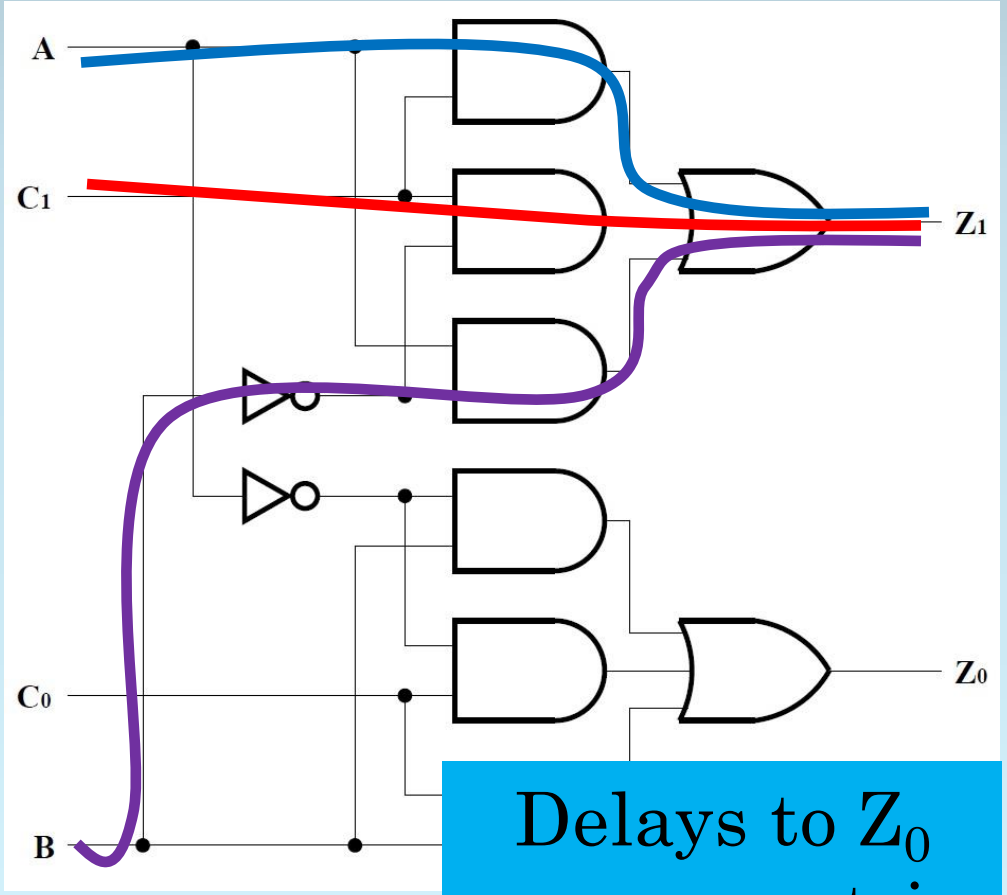
How Many Gate Delays to Z_1?

A to Z_1:
2 gate delays

C_1 to Z_1:
2 gate delays

C_0 to Z_1:
not relevant

B to Z_1:
2 gate delays
(ignoring NOT)

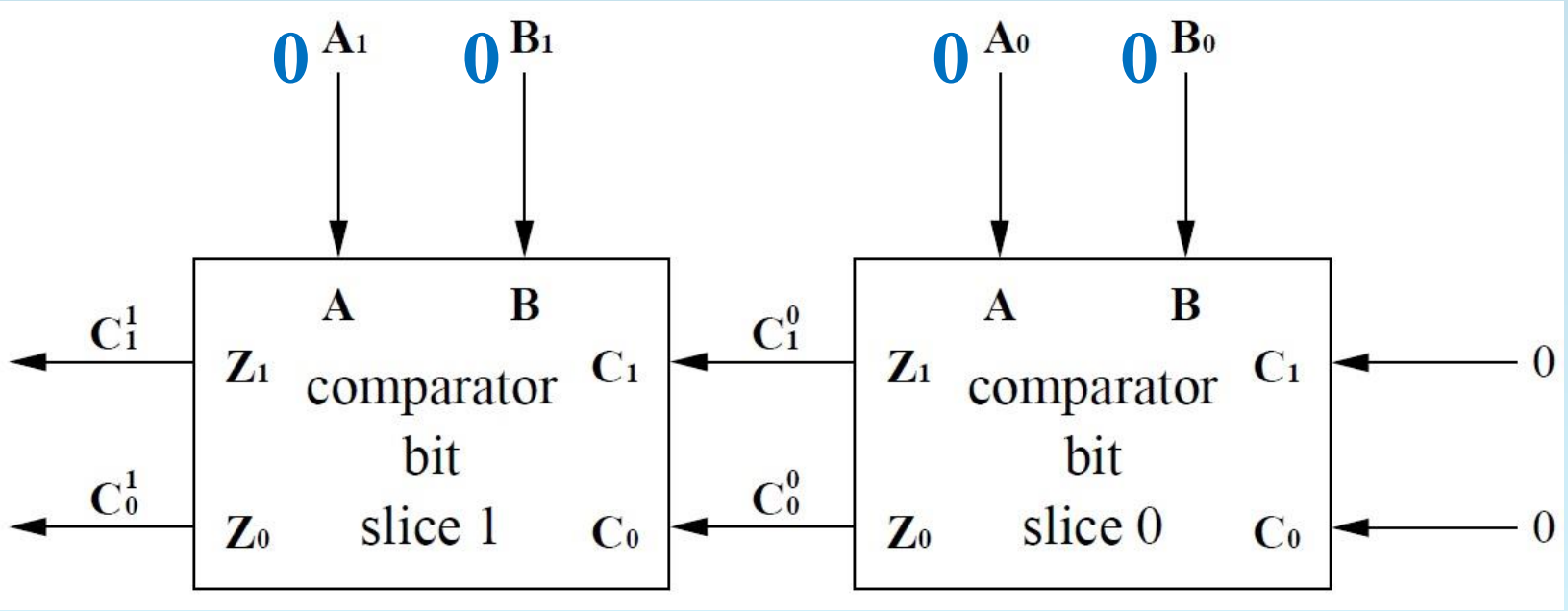


Delays to Z_0 are symmetric.

Extending from One Bit Slice to N Bit Slices

What happens in an **N-bit** design?

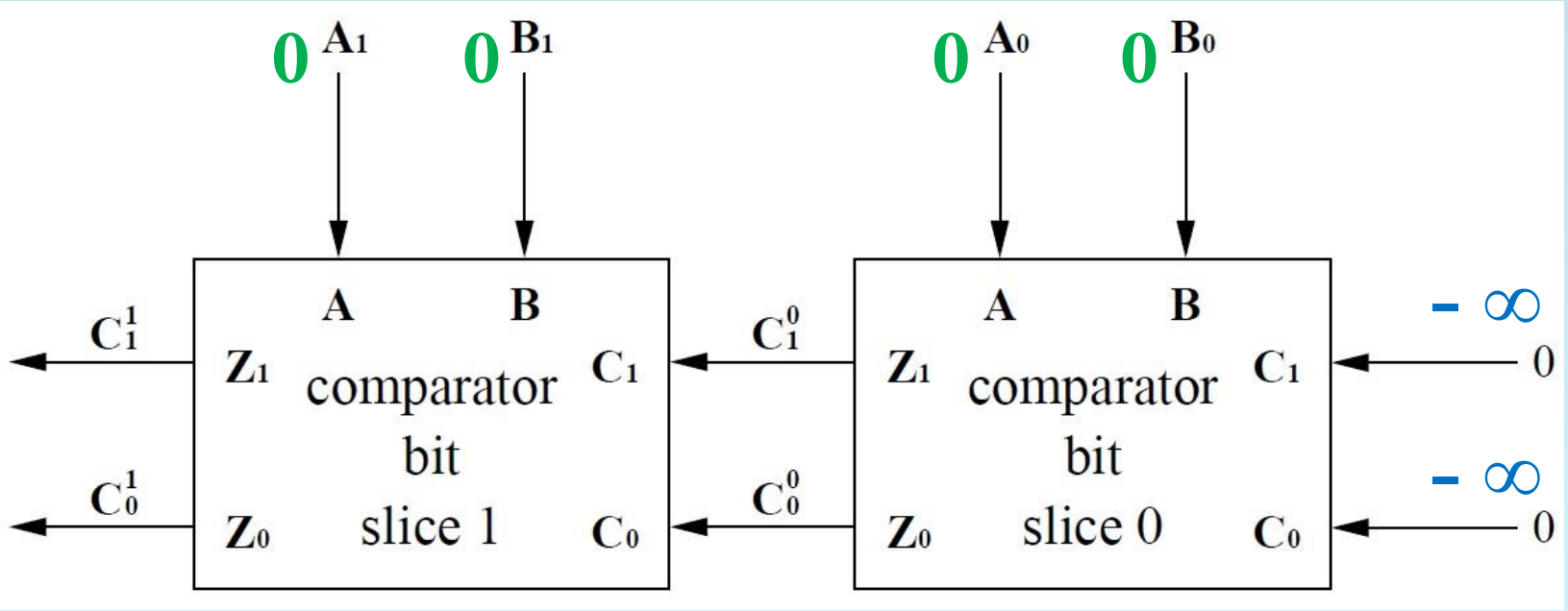
Say that **A and B are available at time 0**.



Constant Inputs are Available Arbitrarily Early

What about the 0s on the right?

Available "forever" … (**time -∞**).



Use Bit Slice Timing to Calculate Times Between Slices

Now we must

- use the delays that we found for one bit slice
- to calculate times for inter-slice **C** values.

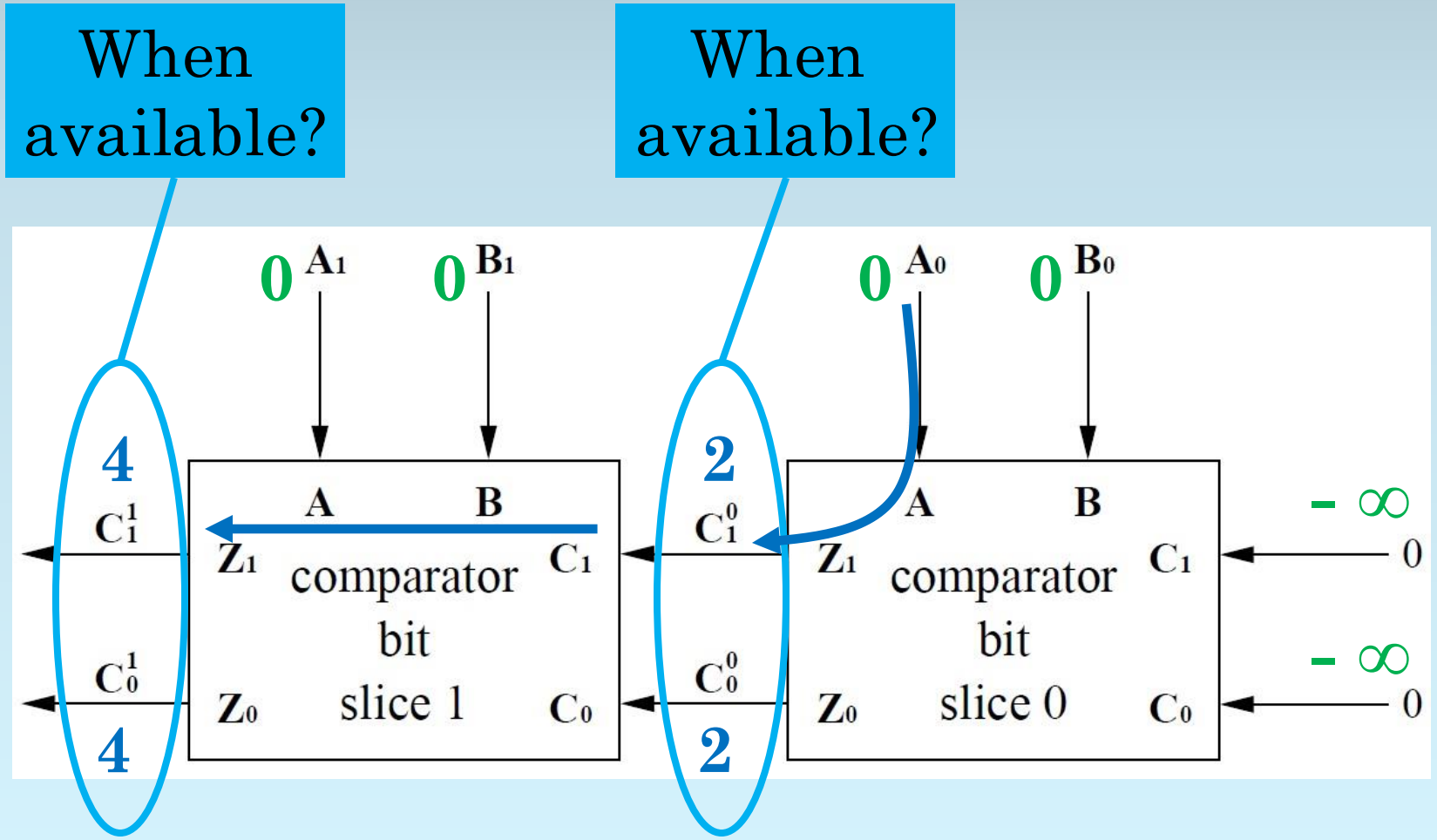
Recall that

- all **A** and **B** bits are available at **time 0**,
- so the **C to Z delays are the most important**.

We found

- **C_1 to Z_1: 2 gate delays**
- **C_0 to Z_0: 2 gate delays**

Calculate the Time at Which C^M Becomes Available



A More Detailed Version of Our Calculations

Grey is "not relevant," and green is maximum (time at which $\mathbf{Z_i}$ is available).

(bit slice 0)	**A**	**B**	**C_1**	**C_0**
input available at	0	0	-∞	-∞
delay from input to Z_1	+2	+2	+2	
Z_1 not available until	2	2	-∞	
delay from input to Z_0	+2	+2		+2
Z_0 not available until	2	2		-∞

A More Detailed Version of Our Calculations

Grey is "not relevant," and green is maximum (time at which $\mathbf{Z_i}$ is available).

(bit slice 1)	**A**	**B**	**C_1**	**C_0**
input available at	0	0	2	2
delay from input to Z_1	+2	+2	+2	
Z_1 not available until	2	2	4	
delay from input to Z_0	+2	+2		+2
Z_0 not available until	2	2		4

Generalize the Result to an N-Bit Comparator

C_1^0 and C_0^0 are available at time 2 (2 gate delays).*

C_1^1 and C_0^1 are available at time 4.

When are C_1^{N-1} and C_0^{N-1} available (these are the answer for an **N-bit** comparator)?

N-bit answer is available at time 2N.

*In the notes, the inverters are counted, so paths from A and B are slightly longer, and all timings are increased by 1.

We May be Able to Improve Our Comparator Design

Can we do better?

(You should ask: better in what sense?)

Can we reduce delay?

- **Unlikely** with a bit-sliced design.
- Not easy to implement most functions with one gate.

Can we reduce area?

- **Maybe** …
- Let's do some algebra.

Use Algebra to Find Common Subexpressions (A'B, AB')

Start with $Z_1 = AB' + AC_1 + B'C_1$

then use distributivity to pull out C_1:

$$Z_1 = AB' + (A + B')C_1$$

and rewrite the $(A + B')$ factor as a NAND:

$$Z_1 = AB' + (A'B)'C_1$$

Similarly, $Z_0 = A'B + (AB')'C_0$

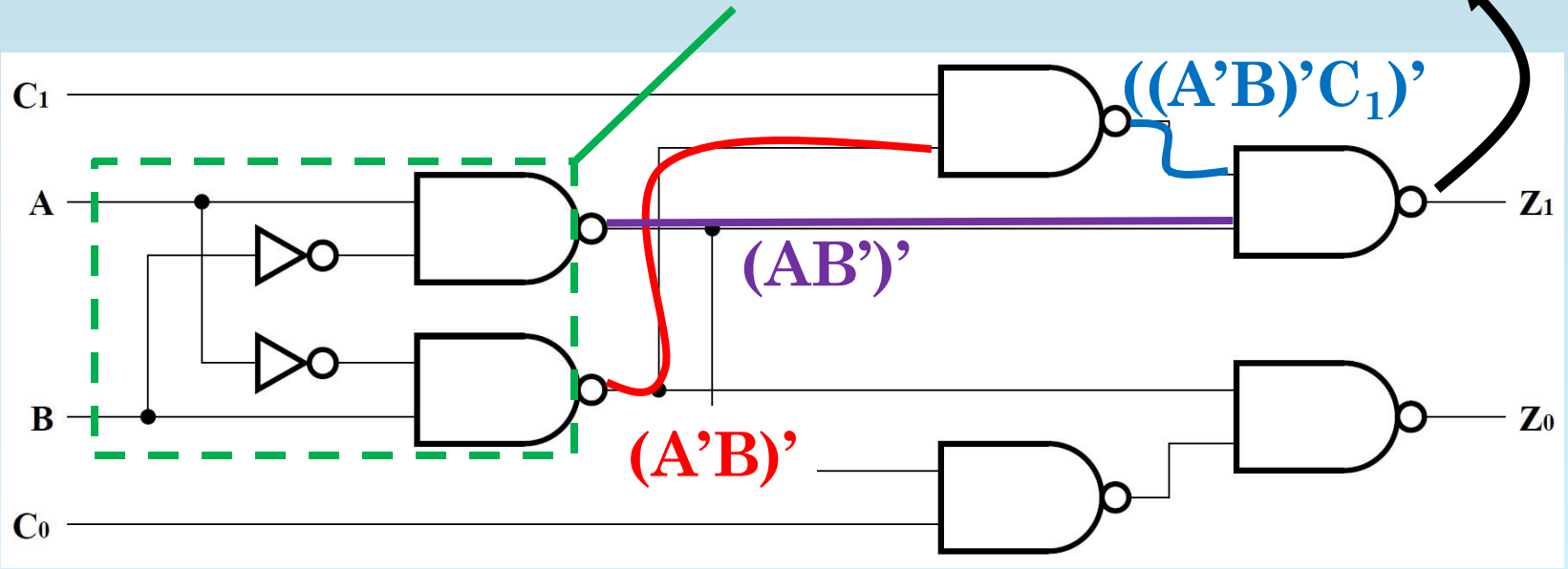
Notice that we now reuse AB' and $A'B$.

The New Implementation Uses Fewer Gates

The diagram below shows the new equations using NAND gates.

$$Z_1 = [\ (AB')'\ ((A'B)'C_1)'\ ]'$$
$$= AB' + (A'B)'C_1$$

The single-bit core is here.

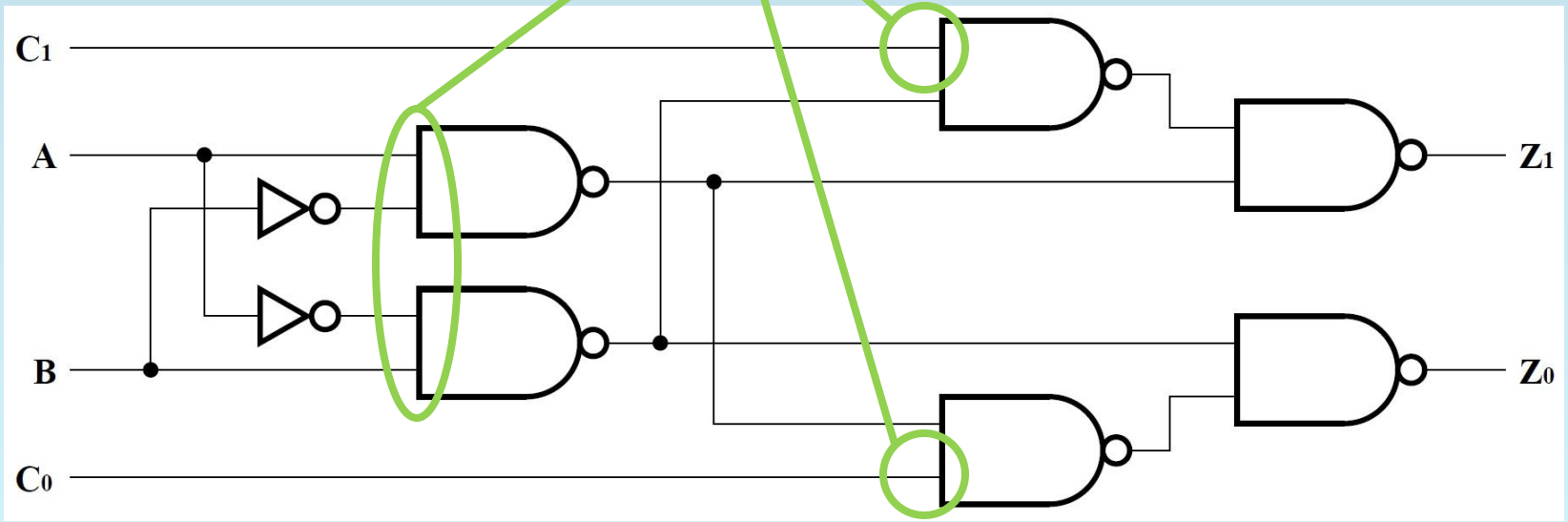


Area Heuristic for the New Design is 12

Let's analyze area for the new design.

How many literals? 6

How many operators? 6 (NAND)



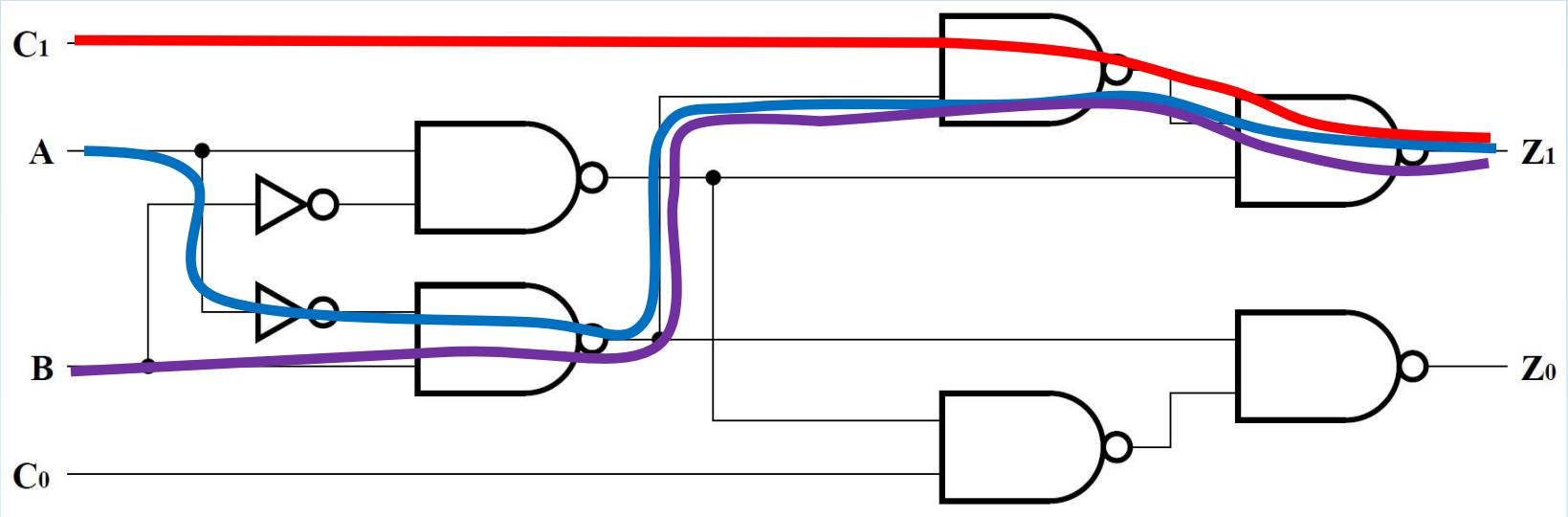
Delay Analysis for the New Design

A to Z_1: 3 gate delays (ignoring NOT)

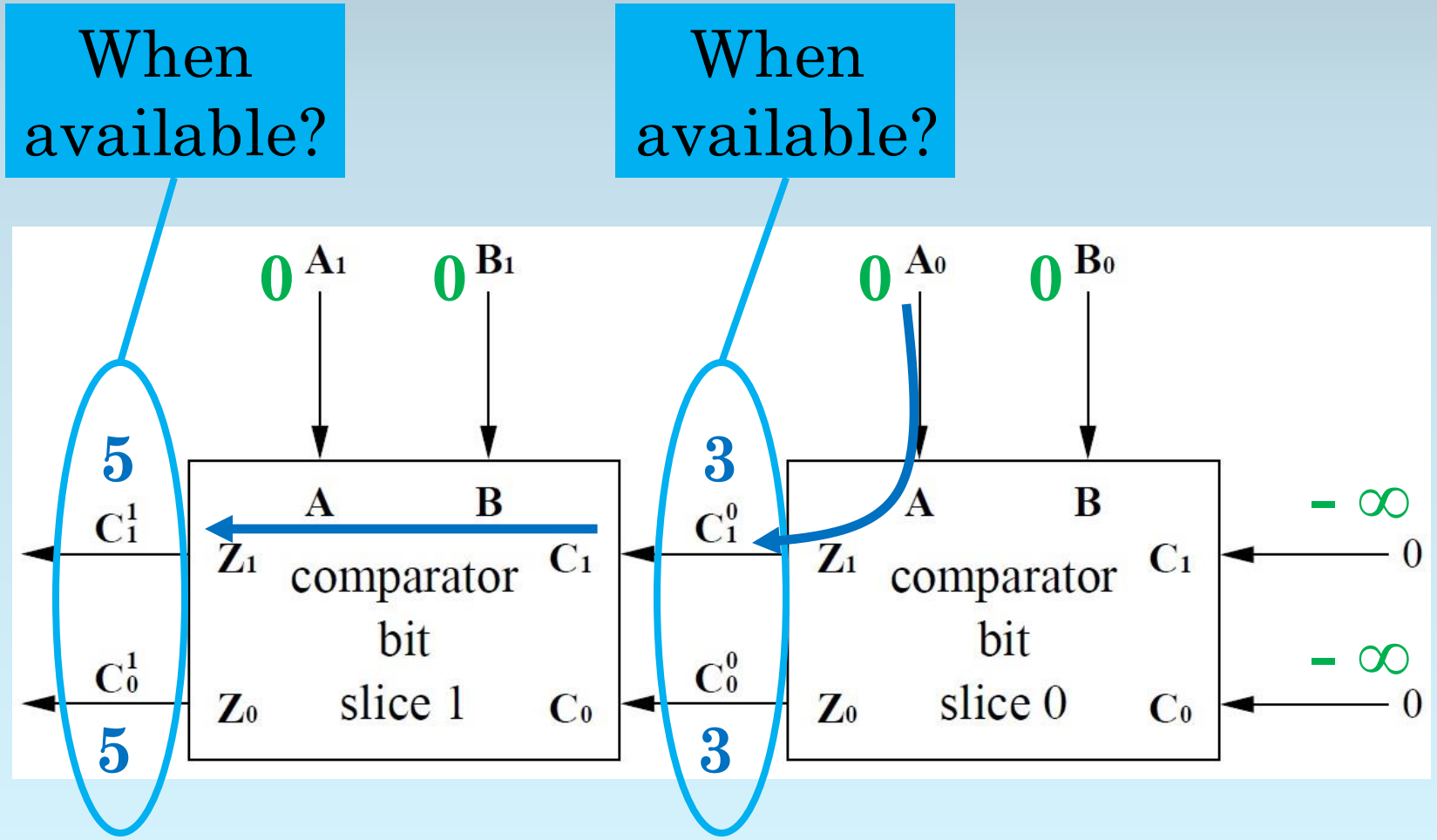
C_1 to Z_1: 2 gate delays

B to Z_1: 3 gate delays

50% slower?!



Calculate the Time at Which C^M Becomes Available



A More Detailed Version of Our Calculations

Grey is "not relevant," and green is maximum (time at which $\mathbf{Z_i}$ is available).

(bit slice 0)	**A**	**B**	**C_1**	**C_0**
input available at	0	0	-∞	-∞
delay from input to Z_1	+3	+3	+2	
Z_1 not available until	3	3	-∞	
delay from input to Z_0	+3	+3		+2
Z_0 not available until	3	3		-∞

A More Detailed Version of Our Calculations

Grey is "not relevant," and green is maximum (time at which $\mathbf{Z_i}$ is available).

(bit slice 1)	**A**	**B**	**C_1**	**C_0**
input available at	0	0	3	3
delay from input to Z_1	+3	+3	+2	
Z_1 not available until	3	3	5	
delay from input to Z_0	+3	+3		+2
Z_0 not available until	3	3		5

The Slice-to-Slice Paths are the Important Ones

C_1^0 and C_0^0 are available at time 3 (2 gate delays).*

C_1^1 and C_0^1 are available at time 5.

When are C_1^{N-1} and C_0^{N-1} available (these are the answer for an **N-bit** comparator)?

N-bit answer is available at time 2N+1.

*In the notes, the inverters are counted, so paths from A and B are slightly longer, and all timings are increased by 1.

Overall: Much Better Area for Slightly More Delay

So the new design

- **reduces area by about 40%**
  (area 12N compared to area 20N).
- **increases delay by 1**
  (2N+1 gate delays compared to
  2N gate delays).

Can We Do Even Better?

Yes, but it's not as easy.

For example, we can design a slice
- that compares multiple bits of **A** and **B**.
- See Notes 2.4.6 for an example.

We can also solve the full **N-bit** problem.

In other words, trade **more human work and complexity for better area and delay**.

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

ECE 120: Introduction to Computing

A Comparator for 2's Complement

Comparing 2's Complement Is Different from Unsigned

Let's design a comparator for **2's complement** numbers.

Is the function the same as with **unsigned** (like addition)?

For **unsigned**, **1001 > 0101**.

Is the same true with 2's complement?

No.

Should we just start over?

Start with the Sign Bits

Let's try a little harder first…

If we compare two non-negative numbers,

- the approach IS the same.
- Right?

Maybe we can just use some extra logic to handle the sign bits?

Consider All Possible Combinations of Sign Bits

Let's make a table based on the sign bits:

A_s	B_s	interpretation	solution
0	0	$A \geq 0$ AND $B \geq 0$	use unsigned comparator
0	1	$A \geq 0$ AND $B < 0$	$A > B$
1	0	$A < 0$ AND $B \geq 0$	$A < B$
1	1	$A < 0$ AND $B < 0$	unknown

Interpret 2's Complement as Unsigned

Remember our "simple" rule for translating **2's complement** bit patterns to decimal?

The pattern $\mathbf{A = a_{N-1}a_{N-2} \ldots a_1a_0}$

has value $\mathbf{V_A = -a_{N-1}2^{N-1} + a_{N-2}2^{N-2} + \ldots + a_02^0}$

Let **A** be negative ($\mathbf{a_{N-1} = 1}$).

Interpreted as **unsigned**, the same bits have value $\mathbf{V_A + 2^N}$.*

*The statement is true by definition of 2's complement, actually.

Negative Numbers Can be Compared Directly

What happens if we feed two negative 2's complement numbers into our unsigned comparator?

We compare $V_A + 2^N$ with $V_B + 2^N$.

And we get an answer: <, =, or >.

Let's say that we find $V_A + 2^N < V_B + 2^N$.

In that case, $V_A < V_B$, so **we have the right answer for 2's complement**.

The same result holds for other answers.

We Need Special Logic for the Sign Bits

Now we can complete our table:

A_s	B_s	interpretation	solution
0	0	$A \geq 0$ AND $B \geq 0$	use unsigned comparator
0	1	$A \geq 0$ AND $B < 0$	$A > B$
1	0	$A < 0$ AND $B \geq 0$	$A < B$
1	1	$A < 0$ AND $B < 0$	use unsigned comparator

Simply Flip the Wires on the Most Significant Bit

Can we just flip the wires on the sign bits?

For $A_s = 0$ and $B_s = 1$,
- we feed in **$A_{N-1} = 1$** and **$B_{N-1} = 0$**, and
- the unsigned comparator produces **$A > B$**.

For $A_s = 1$ and $B_s = 0$,
- we feed in **$A_{N-1} = 0$** and **$B_{N-1} = 1$**, and
- the unsigned comparator produces **$A < B$**.

What about when $A_s = B_s$?

Flipping the bits then has no effect!

Answers are also correct in those cases.

One Comparator with a Control Signal can Do Both

Can we use a single comparator
to perform both kinds of comparisons?

Yes, if we
- add a control signal **S**
- to tell the comparator whether to do **unsigned** (**S=0**) or **2's complement** (**S=1**) comparison.

Simply **XOR'ing the most significant bits of A and B with S suffices**.
- This approach leverages flexibility in the problem to reduce the logic needed.
- Analyze the design to understand how it works.