

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

ECE 120: Introduction to Computing

Serialization of Bit-Sliced Designs

An Abstract Model of a Bit Slice

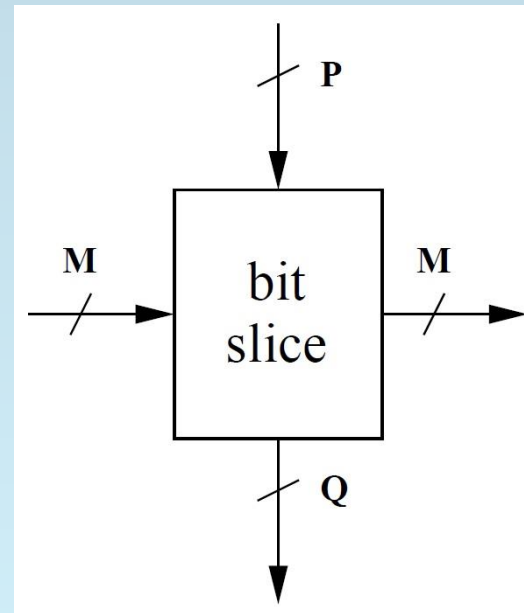
Consider a bit-sliced design.

Each bit slice computes a function of

- **P** input bits, and
- **M** bits from the previous bit slice.

And each bit slice produces

- **Q** output bits, and
- **M** bits for the next bit slice.



Use Flip-Flops to Serialize a Bit-Sliced Design

How do we handle **N** bits?

Previously, we used **N** copies of the bit slice.

But now we know how to store bits.

So we could instead

- use **one copy of the bit slice**, and
- **pass the bit slice's M outputs back as inputs** in the next clock cycle.

Such an implementation is a **serial design** because it handles one bit at a time.

Boundary Conditions Add Some Complexity

But it's not quite so simple.

What about the first bit slice?

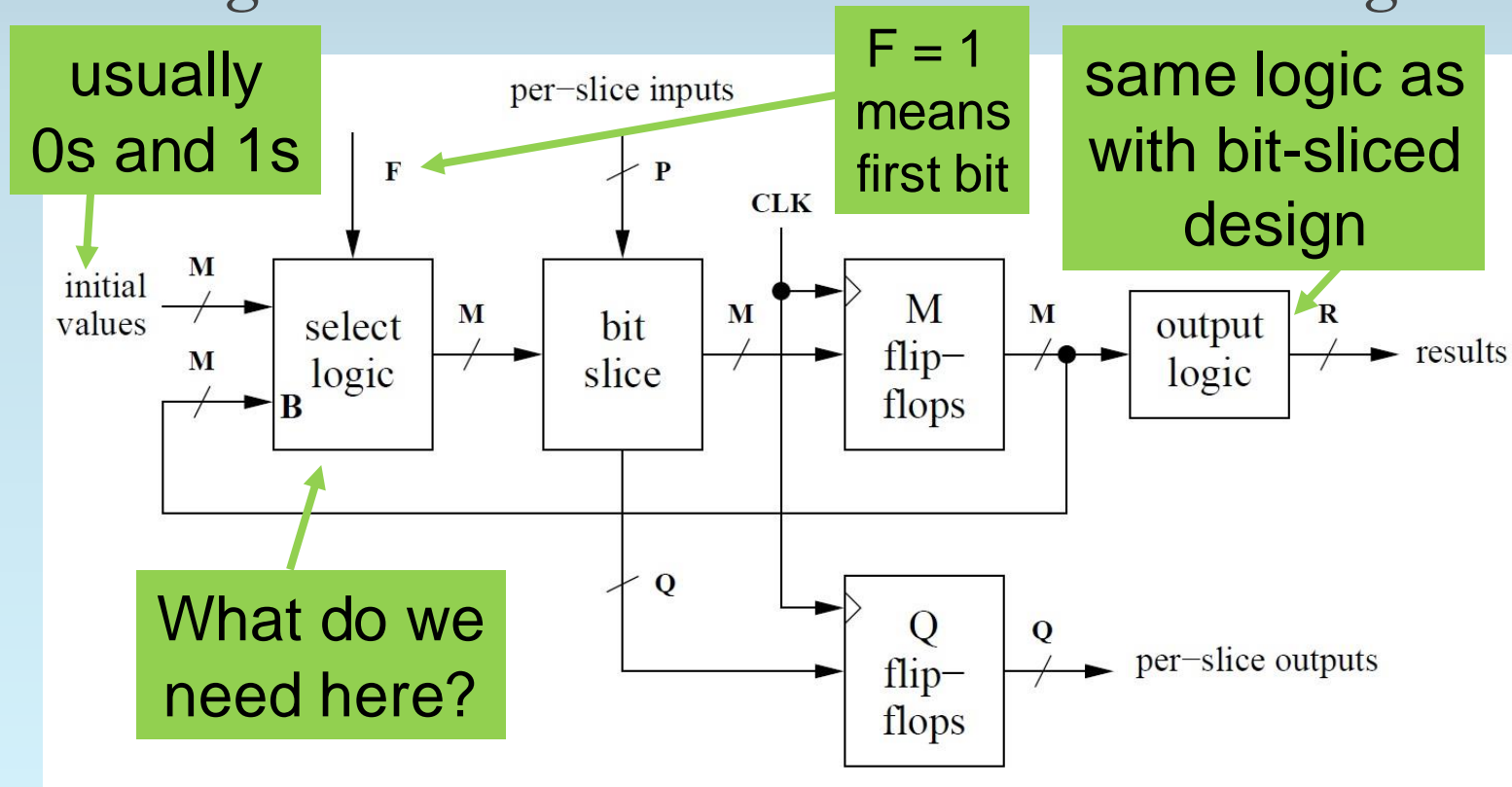
That bit slice has no previous bit slice,
so instead the **M-bit input is 0s and 1s.**

And **what about the last bit slice?**

The **M-bit** output to the next slice is not
always acceptable as an answer. We
sometimes **need additional output logic.**

Not Much Work to Serialize a Design

This figure shows an abstract serial design.



Muxes Suffice, But Let's Optimize the Design a Little

M 2-to-1 muxes controlled by F suffice for the selection logic.

But since the initial values are usually 0s and 1s, **we can optimize.**

M flip-flops feed their stored values back into the selection logic. Let's call these bits **B_i**.

And let's call the **M** bits produced for the bit slice **C_i**.

Need One NOR Gate When Initializing to 0

Start by assuming a 0 bit in place of B_i for the first bit (when $F = 1$).

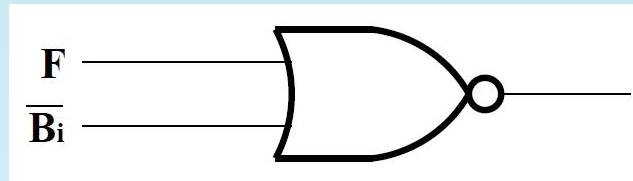
And write a truth table for C_i .

From the table, we can write

$$C_i = F'B_i = (F + B_i)'$$

Remembering that B_i' is available from the flip-flop output, a single NOR gate suffices.

F	C_i
0	B_i
1	0



Need NAND and NOT When Initializing to 1

Now assume a 1 bit in place of B_i for the first bit (when $F = 1$).

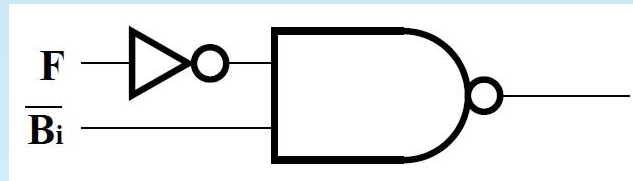
And write a truth table for C_i .

From the table, we can write

$$C_i = F + B_i = (F'B_i)'$$

Remembering that B_i' is available from the flip-flop output, a NAND gate and an inverter for F suffices.

F	C_i
0	B_i
1	1



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

ECE 120: Introduction to Computing

Example of Serialization

Review of General Bit-Slice Model

General model parameters

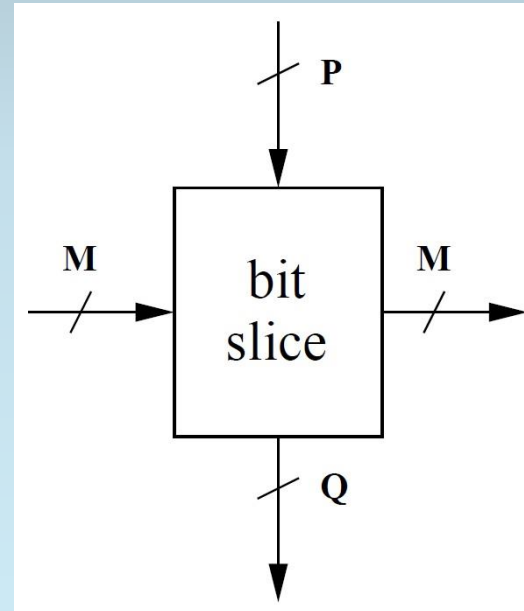
N-bit operands

P bits of input from operands

Q bits of output produced

M bits between bit slices

R bits of final output (not shown; produced by output logic operating on **M** bits from last bit slice).



N-bit Bit-Slice Comparator

Comparator parameters

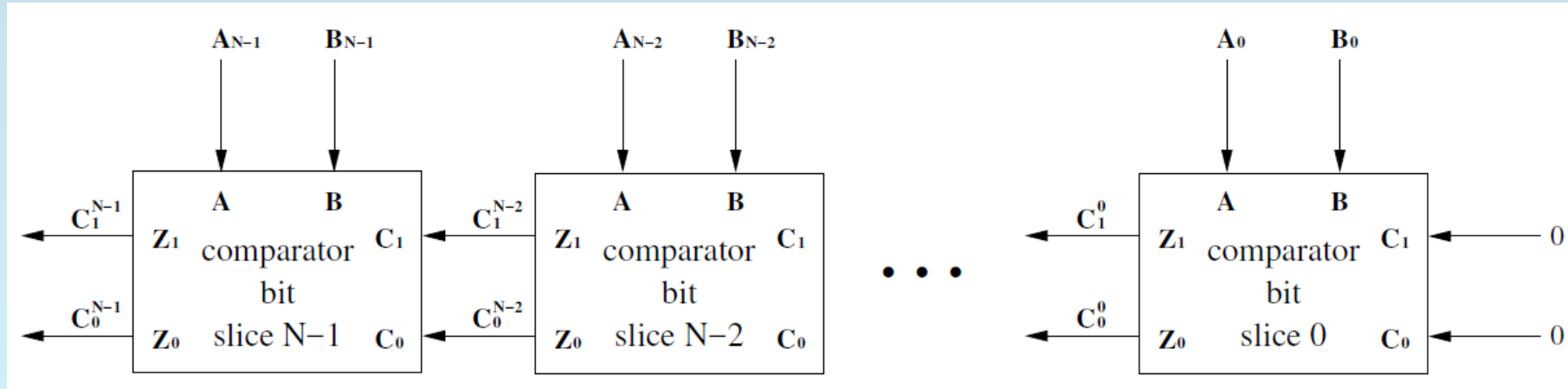
N-bit operands $A_{N-1}A_{N-2}\dots A_1A_0$ and $B_{N-1}B_{N-2}\dots B_1B_0$

P = 2

Q = 0

M = 2

R = 2



Parameter Values for a Serial Comparator

Comparator parameters

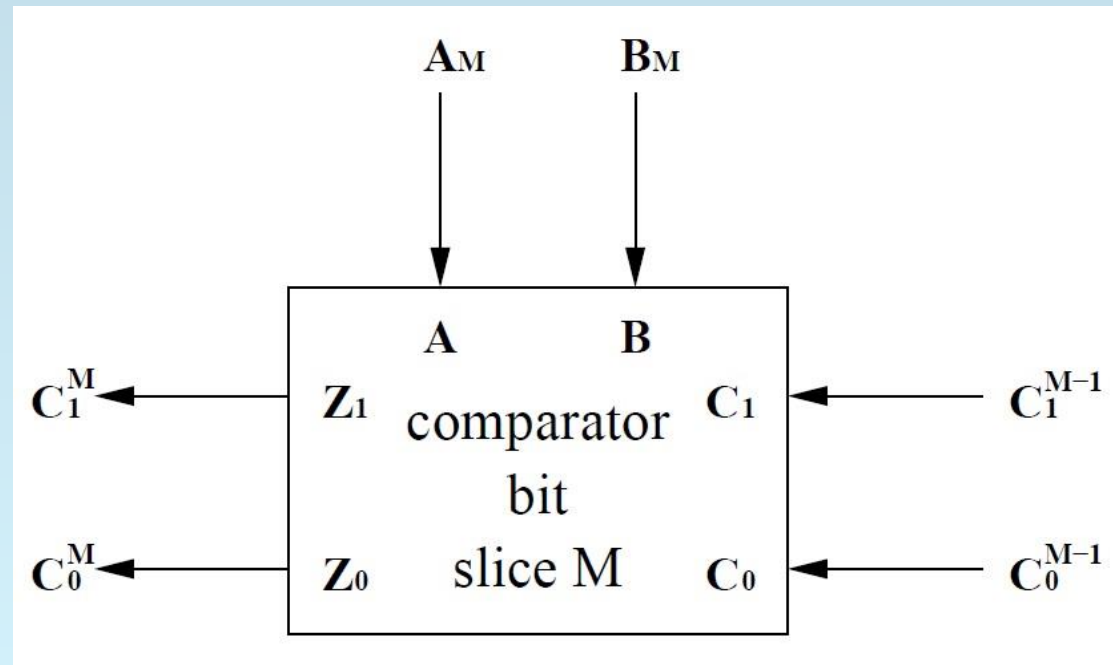
N-bit operands

P = 2

Q = 0

M = 2

R = 2



Initialization of a Serial Comparator

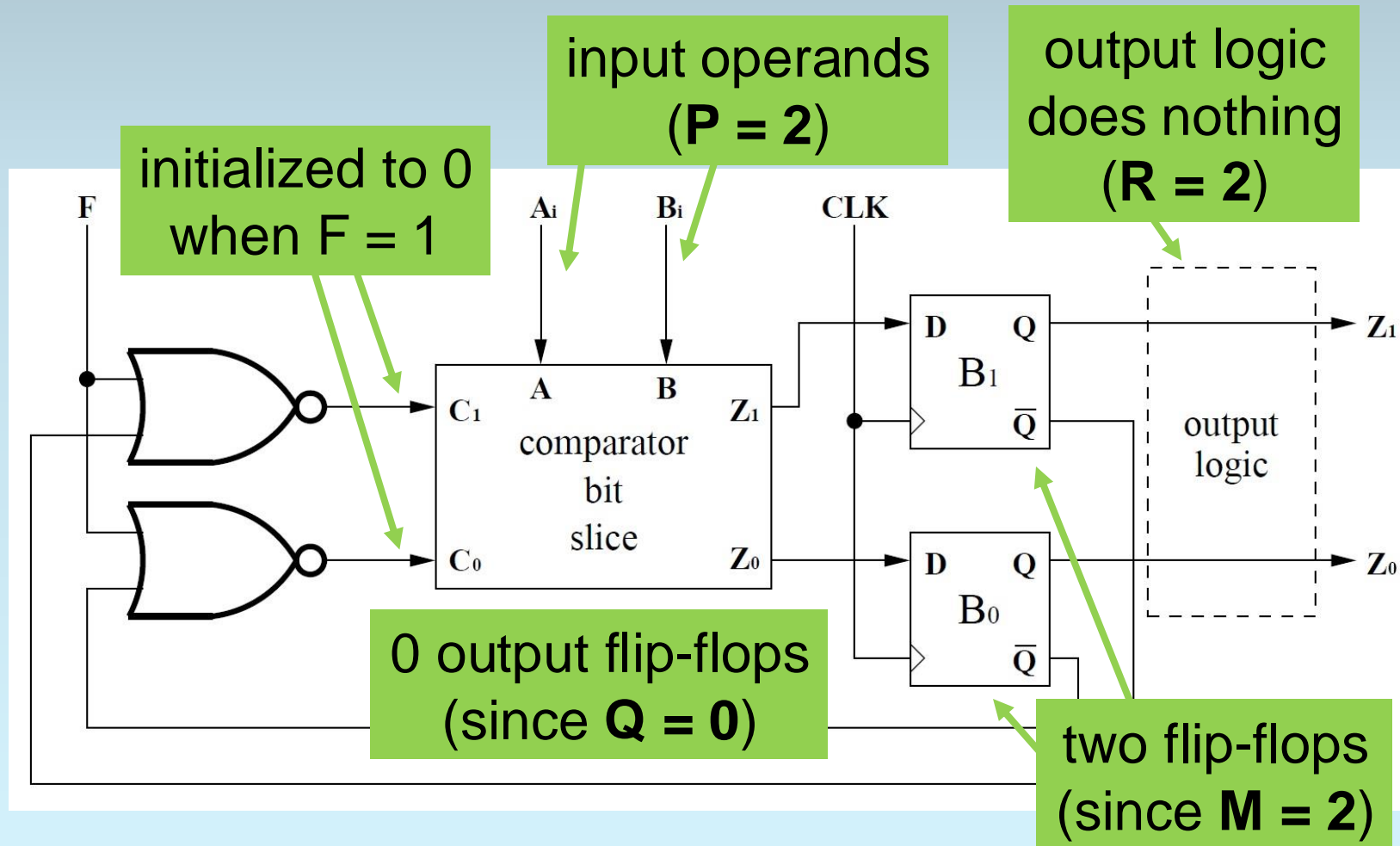
Our comparator bit slice uses the representation shown here to pass information between slices.

What values should be passed to the first bit slice?

A = B, so $C_1C_0 = 00$

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

Example: A Serial Comparator



Discrete Time Implies Delayed Results: $N = 4$

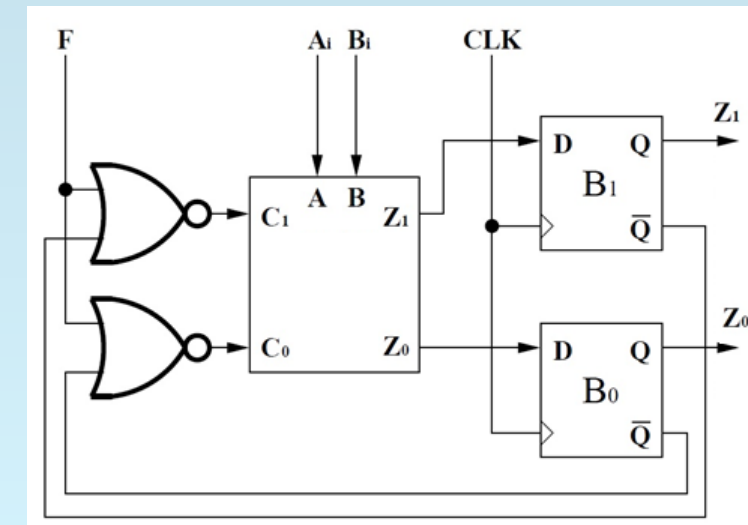
yellow = inputs

(green = bit slice labels)

cycle #	F	A	B	B ₁	B ₀	C ₁	C ₀	Z ₁	Z ₀
0	1	0	1	bits	bits	0	0	0	1
1	0	1	1	0	1	0	1	0	1
2	0	1	0	0	1	0	1	1	0
3	0	0	1	1	0	1	0	0	1
4	x	x	x	0	1	0	?	?	?

blue = outputs

C ₁	C ₀	meaning
0	0	A = B
0	1	A < B
1	0	A > B
1	1	not used

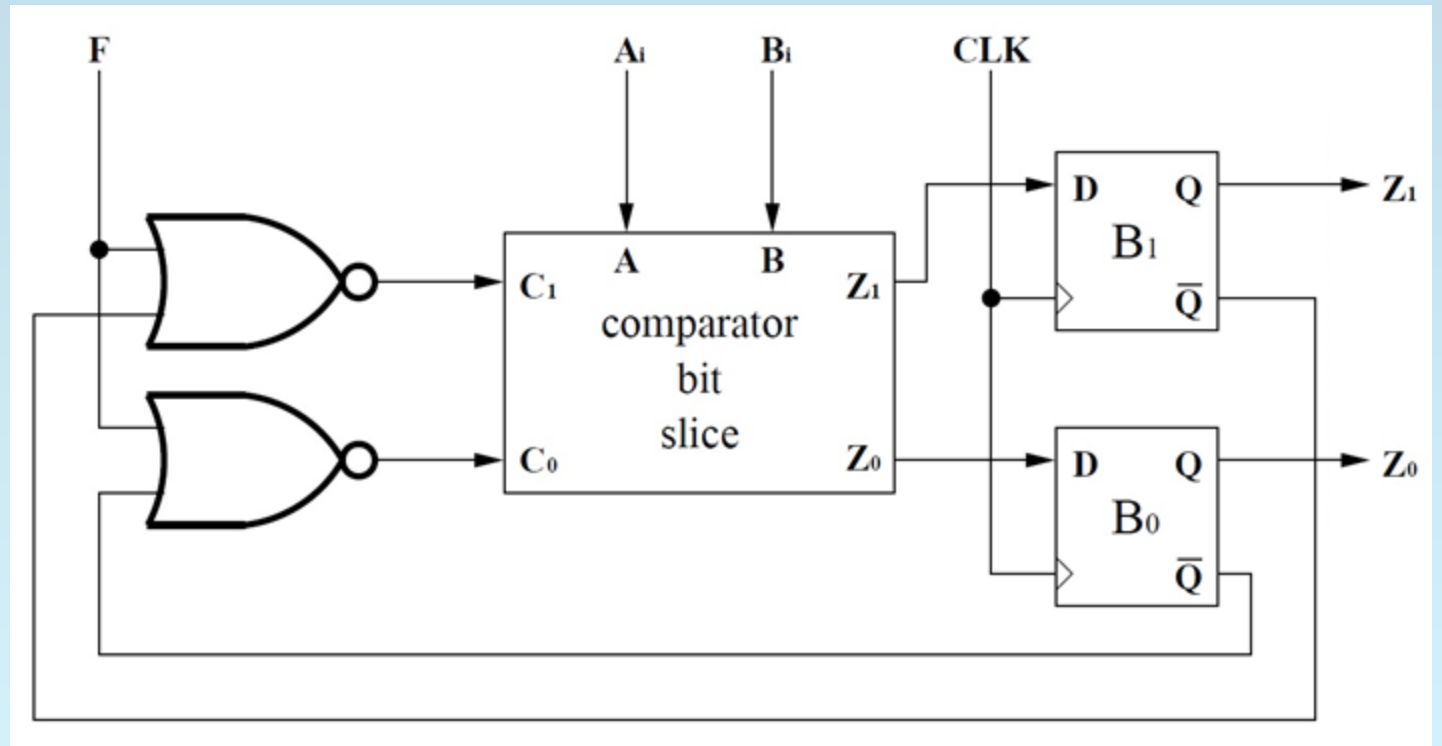


A Serial Comparator Consists of Three Parts

Let's analyze the area of a serial comparator.

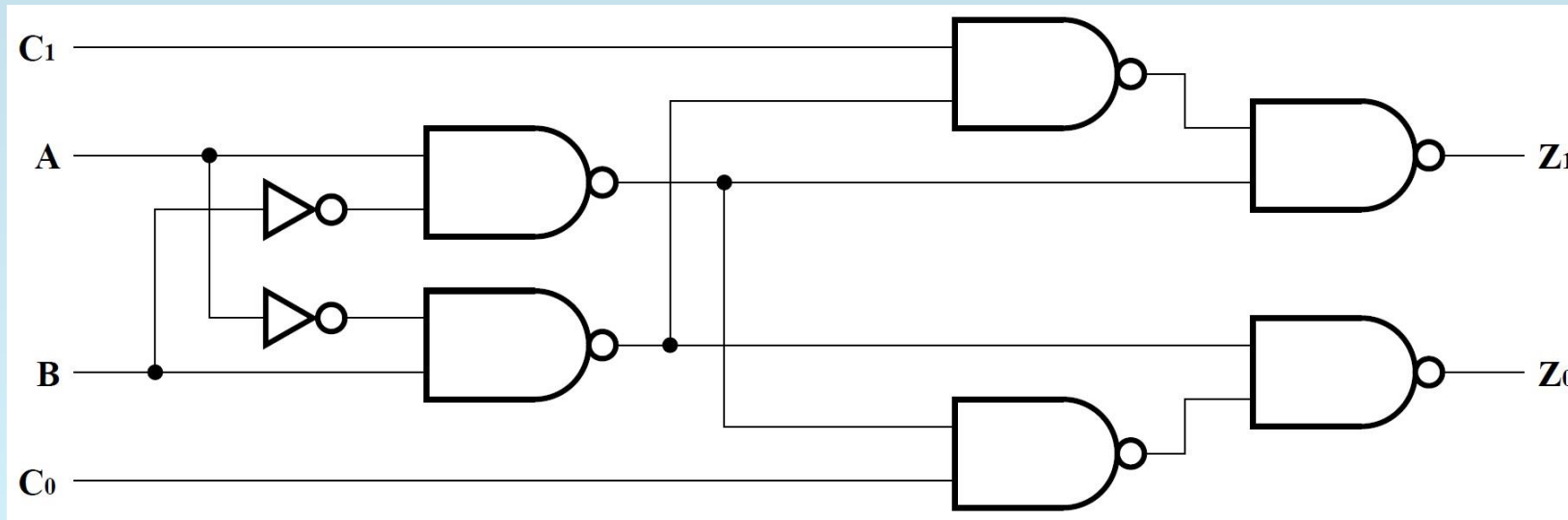
We have:

- one bit slice,
- two flip-flops, and
- two 2-input NOR gates (selection logic).



A Serial Comparator Contains One Bit Slice

Assume the smaller version of the bit slice.
So we need **six 2-input NAND gates** and **two inverters**.




A Serial Comparator Consists of Three Parts

Let's analyze the area of a serial comparator.

We have:

- one bit slice,
- two flip-flops, and
- two 2-input NOR gates (selection logic).



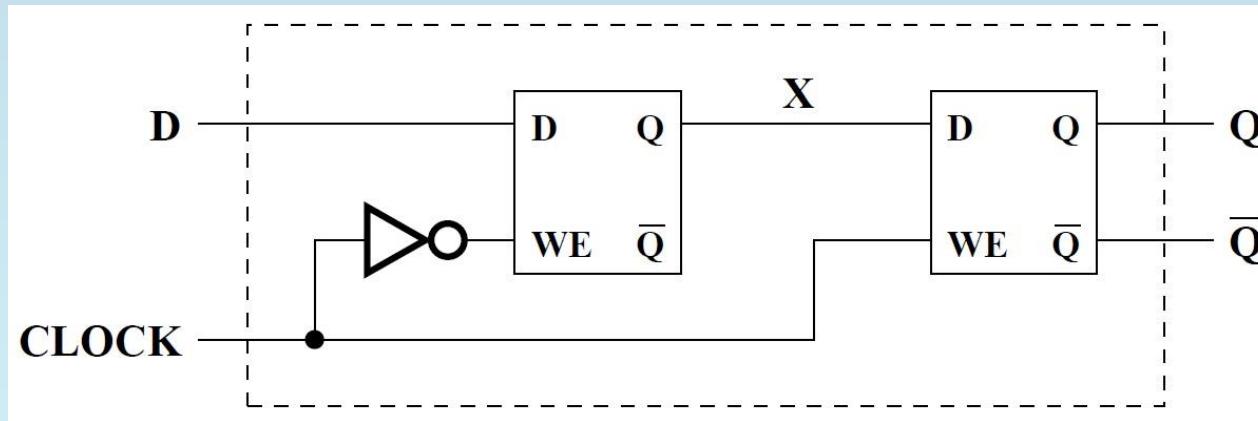
**six 2-input NAND
and two inverters**

A Serial Comparator Uses Two Flip-Flops

A flip-flop is two latches and an inverter.

If we use NOR gates for the first latch, we don't need the extra inverter.*

So **eight 2-input gates** and **two inverters** (each).




*And real designs, optimized at the transistor level, are even smaller.

A Serial Comparator Consists of Three Parts

Let's analyze the area of a serial comparator.

We have:

- one bit slice,
- two flip-flops, and
- two 2-input NOR gates (selection logic).



six 2-input NAND
and two inverters



16 2-input gates
and four inverters

Total: $6+16+2 = 24$ 2-input gates and
 $2+4 = 6$ inverters.

Serial Design is Smaller for $N \geq 4$

To handle **N-bit** operands,
a bit-sliced design requires:

- **$6N$ 2-input gates**, and
- **$2N$ inverters**.

A serial design (independent of **N**) requires

- **24 2-input gates**, and
- **6 inverters**.

The serial design is smaller for $N \geq 4$.

Serial Designs are Slower than Bit-Sliced Designs

The tradeoff? Serial designs are **slower** than bit-sliced designs.

Why?

There are three reasons:

1. All paths matter.
2. Selection logic and flip-flops add to delay.
3. Other logic may further reduce the speed of the common clock.

Let's look at each in more detail.

All Paths Matter in a Serial Design

In an **N-bit bit-sliced design**,

- All external inputs appear at time 0,
- So only the **slice-to-slice paths** in the bit slice **contribute to the multiplier on N**.
- **Other paths contribute only constant time** to the overall delay in the design.

In a **serial design**, all paths matter.

- All input bits arrive in the cycle in which they are consumed, so
- **long paths from any input can slow down the design** overall.

Flip-Flops and Selection Logic Add to Delay

Flip-flops take time

- To store values,
- To produce values.

And the selection logic sits between the flip-flops and the bit-slice inputs.

The clock cycle

- must be long enough
- to account for all of these delays.

Clock Speed is Determined by the Slowest Logic

The longest path through combinational logic determines the speed of the common clock.

In practice,

- engineers identify complex and/or important elements and
- work hard to make them fast or
- to split them into several cycles.

Even if a serial design's logic needs only 0.1 clock cycles, operating on **N-bit** operands still takes **N** clock cycles.

Assume Four Gate Delays On Either Side of Clock Edge

Let's analyze the delay of a serial comparator.

We can count gate delays

- in the bit slice, and
- for the selection logic.

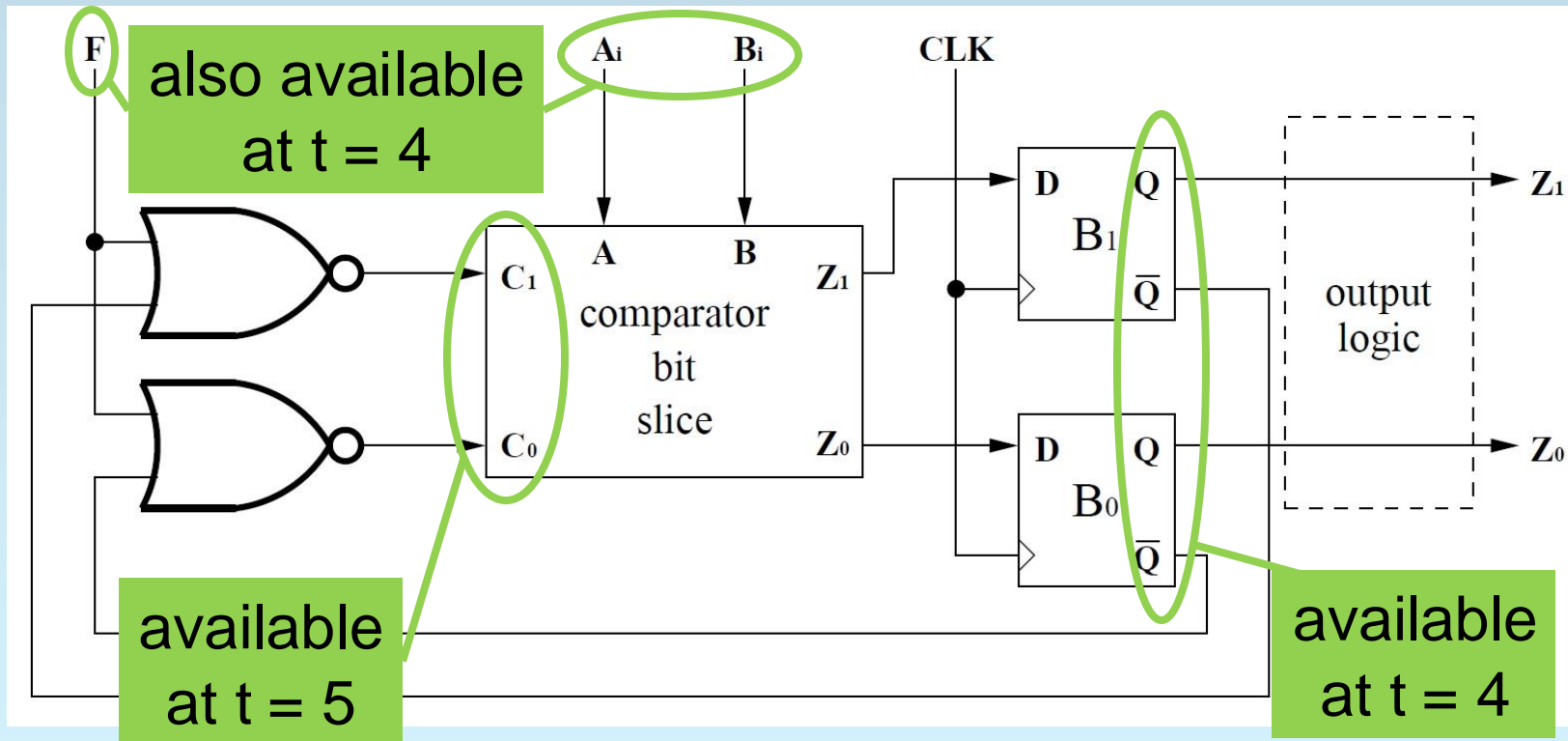
What about the flip-flops?

Let's assume

- **four gate delays** of stable D input needed **before the rising edge**, and
- **four gate delays after the rising edge.**

When Are Inputs Available?

A rising edge arrives at $t = 0$ (gate delays).



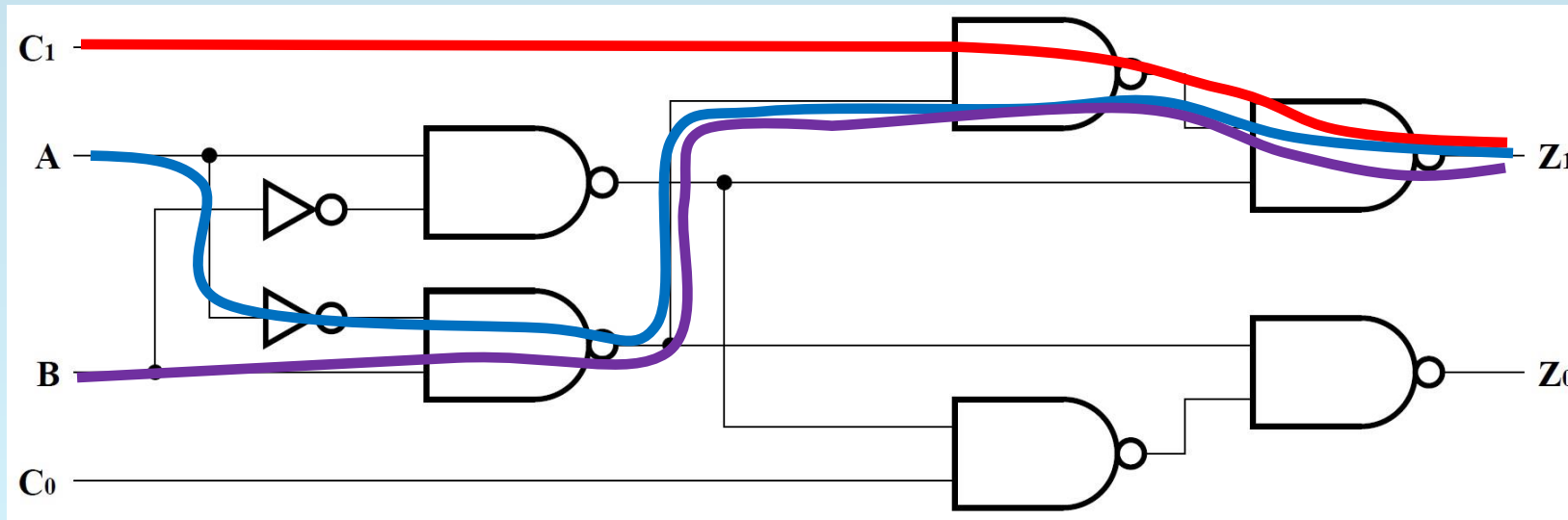
Delay Analysis for the Bit Slice

A to Z_1 : 3 gate delays (ignoring NOT)

C_1 to Z_1 : 2 gate delays

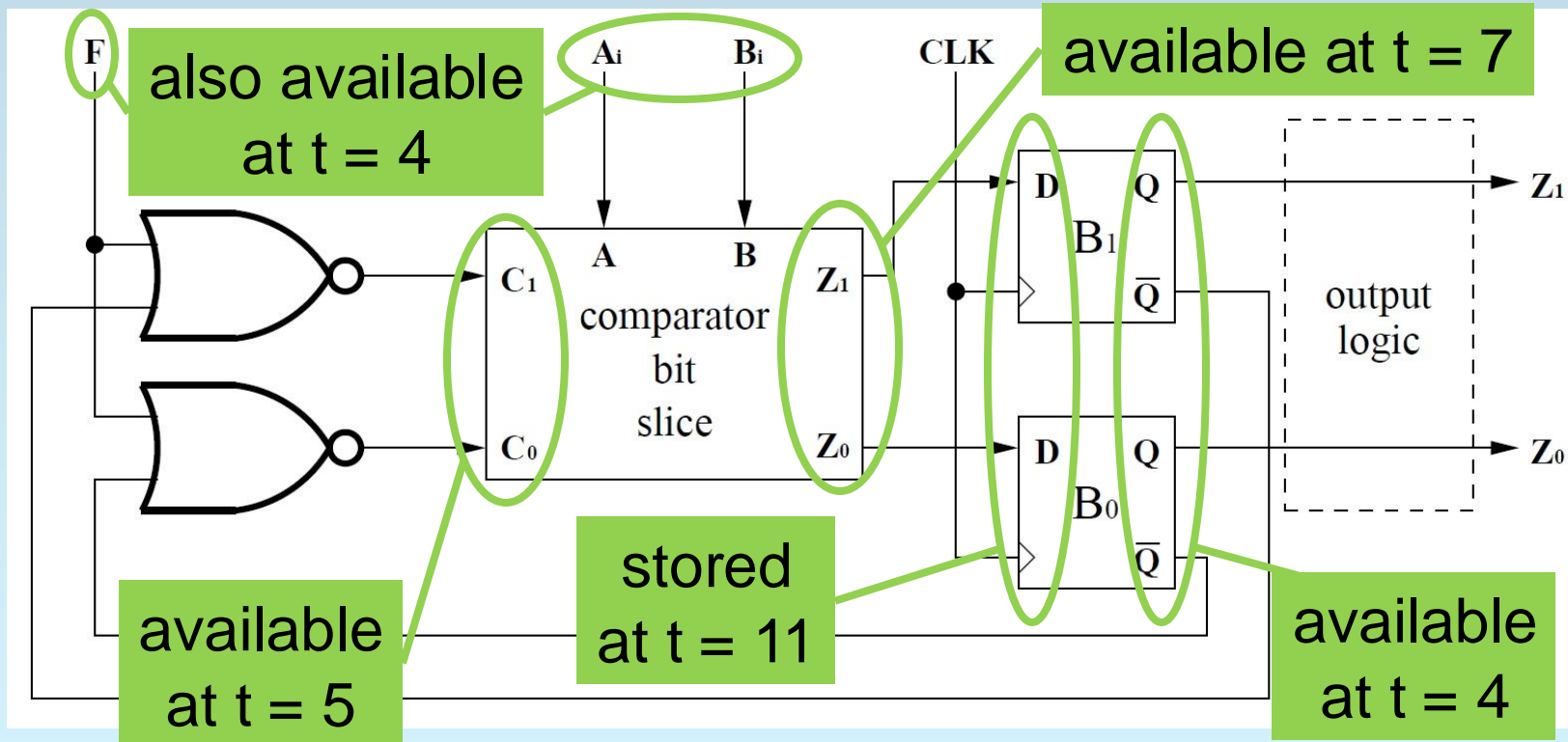
B to Z_1 : 3 gate delays

Paid 4 gate delays
for flip-flops.



When Are Results Stored?

A rising edge arrives at $t = 0$ (gate delays).



Serial Design is At Least 5.5x Slower

To handle **N-bit** operands, a bit-sliced design requires **$2N + 1$ gate delays**.

For a serial design,

- the clock cycle must be **at least 11 gate delays**, and
- we must execute for **N** cycles, so
- **N-bit** operands require **at least $11N$ gate delays**.

The serial design is at least 5.5x slower.

(And may be even slower!)

Bit-Sliced and Serial Designs are Extrema

Both designs are **simple**.

Serial designs are relatively **small, but slow**.

Bit-sliced designs are **fast, but large**.

But we **can build anything in between**:

- 2 bit slices per cycle,
- 3 bit slices per cycle,
- and so forth.

And/or **optimize more than one bit slice**
(increase complexity).

An Example of Partial Serialization in Practice

In one generation of Intel processors,

- the designers included **16-bit adders**
- clocked at twice the main clock speed (**6 GHz** instead of **3 GHz**).

These adders could be used to ...

- perform a **single 32-bit add** (two cycles at 6 GHz), **or**
- perform **two 16-bit adds** for multimedia codes.

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

ECE 120: Introduction to Computing

Another Example of Serialization:
Power-of-2 Checker

Review of General Bit-Slice Model

General model parameters

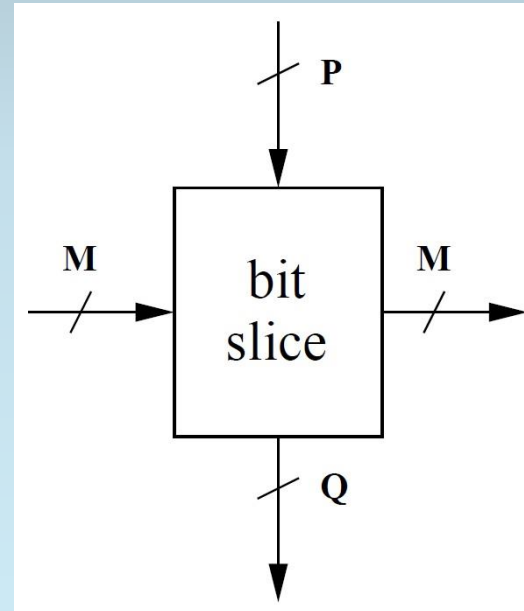
N-bit operands

P bits of input from operands

Q bits of output produced

M bits between bit slices

R bits of final output (not shown; produced by output logic operating on **M** bits from last bit slice).



Parameter Values for a Power-of-2 Checker

Power-of-2 checker parameters

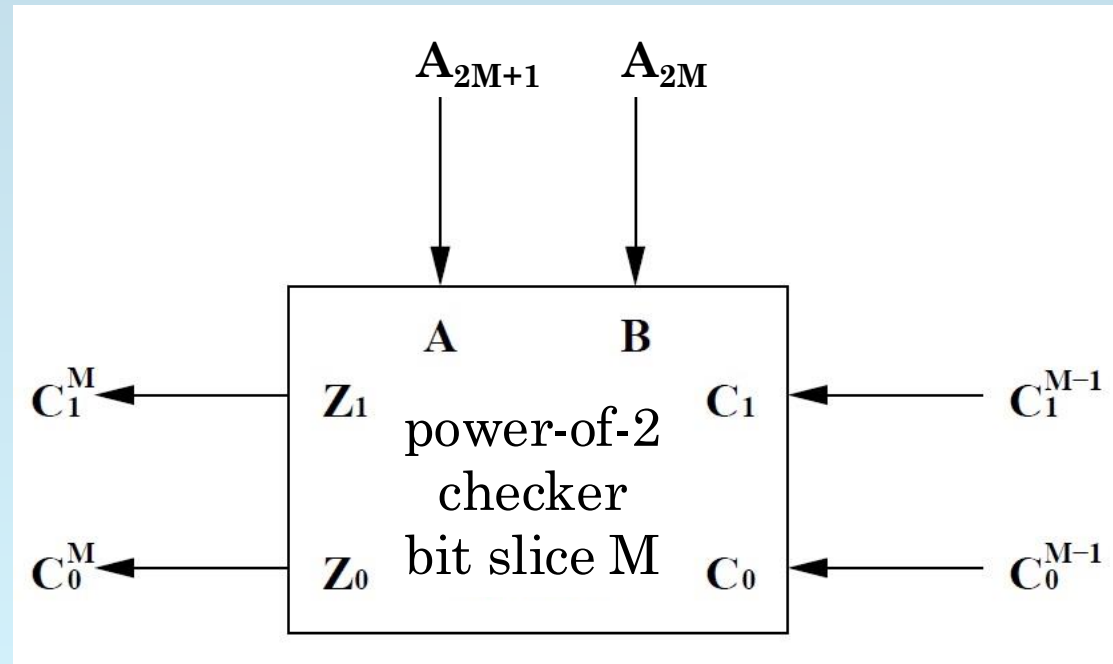
N-bit operands

P = 2

Q = 0

M = 2

R = 1



Initialization of a Power-of-2 Checker

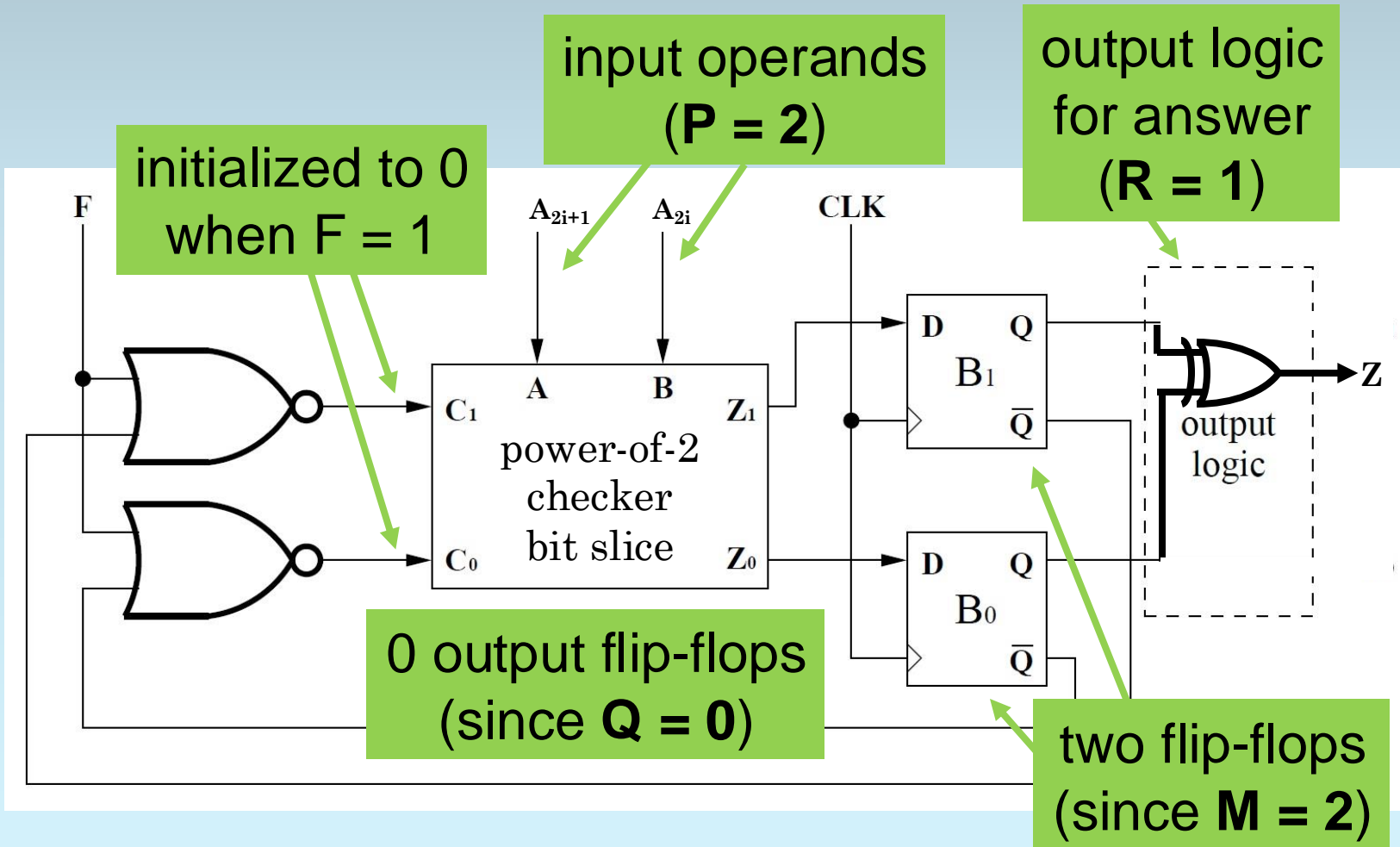
Our power-of-2 checker bit slice uses the representation shown here to pass information between slices.

**What values
should be passed
to the first bit
slice?**

**No 1 bits yet,
so $C_1C_0 = 00$**

C_1	C_0	meaning
0	0	no 1 bits
0	1	one 1 bit
1	0	not used
1	1	>one 1 bit

Example: A Power-of-2 Checker



Discrete Time Implies Delayed Results: $N = 8$

yellow = inputs

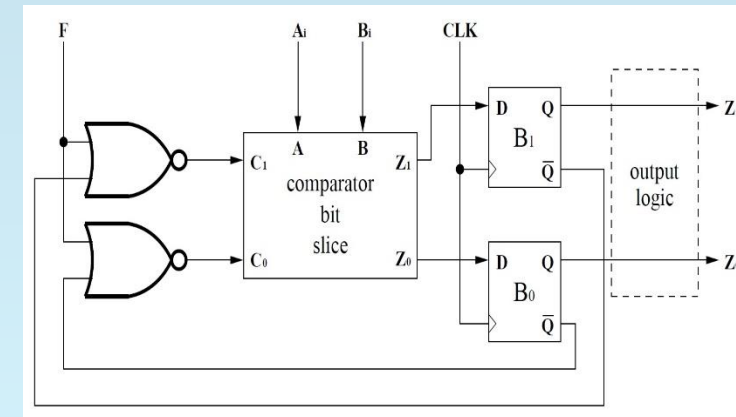
(green = bit slice labels)

cycle #	F	A	B	B ₁	B ₀	C ₁	C ₀	Z ₁	Z ₀
0	1	0	0	bits	bits	0	0	0	0
1	0	0	1	0	0	0	0	0	1
2	0	0	0	0	1	0	1	0	1
3	0	1	0	0	1	0	1	1	1
4	x	x	x	1	1	?	?	?	?

blue = outputs

$$Z = B_1 \oplus B_0 = 0$$

C ₁	C ₀	meaning
0	0	no 1 bits
0	1	one 1 bit
1	0	not used
1	1	>one 1 bit



A Power-of-2 Checker Consists of Four Parts

Let's analyze the area of a power-of-2 checker.

We have:

- one bit slice,
- two flip-flops,
- two 2-input NOR gates (selection logic),
- and a 2-input XOR.

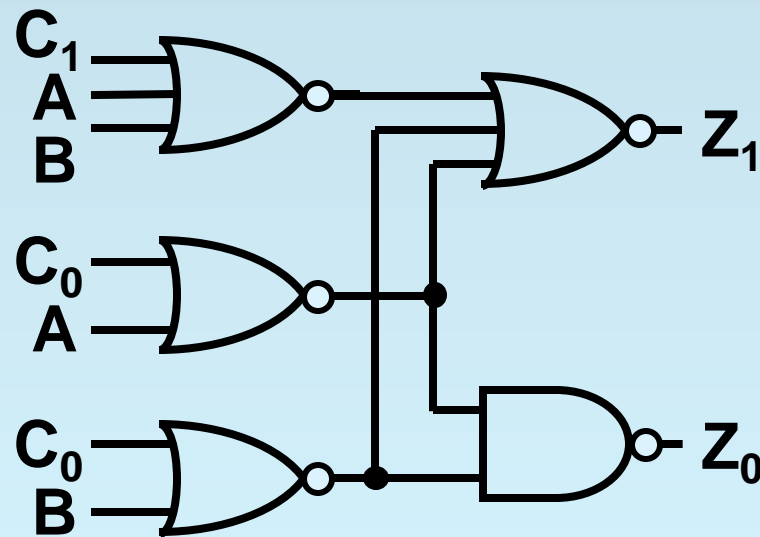


**16 2-input gates
and four inverters**

A Power-of-2 Checker Contains One Bit Slice

Here's our design for the power-of-2 checker bit slice.

So we need **three 2-input gates** and **two 3-input gates**.

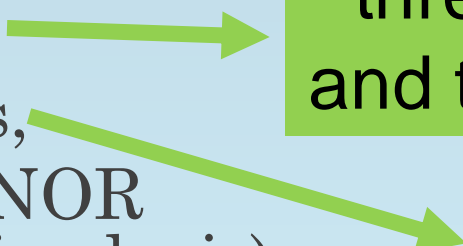


A Power-of-2 Checker Consists of Four Parts

Let's analyze the area of a power-of-2 checker.

We have:

- one bit slice,
- two flip-flops,
- two 2-input NOR gates (selection logic),
- and a 2-input XOR.



three 2-input gates
and two 3-input gates

16 2-input gates
and four inverters

Total: $3+16+2 = 21$ 2-input gates,
2 = 2 3-input gates,
4 = 4 inverters, and
1 2-input XOR.

Serial Design is Smaller for $N \geq 10$

To handle **N-bit** operands (2 bits per bit slice), a bit-sliced design requires:

- **$3N/2$ 2-input gates** ($6N$ transistors),
- **N 3-input gates** ($6N$ transistors), and
- **1 2-input XOR.**

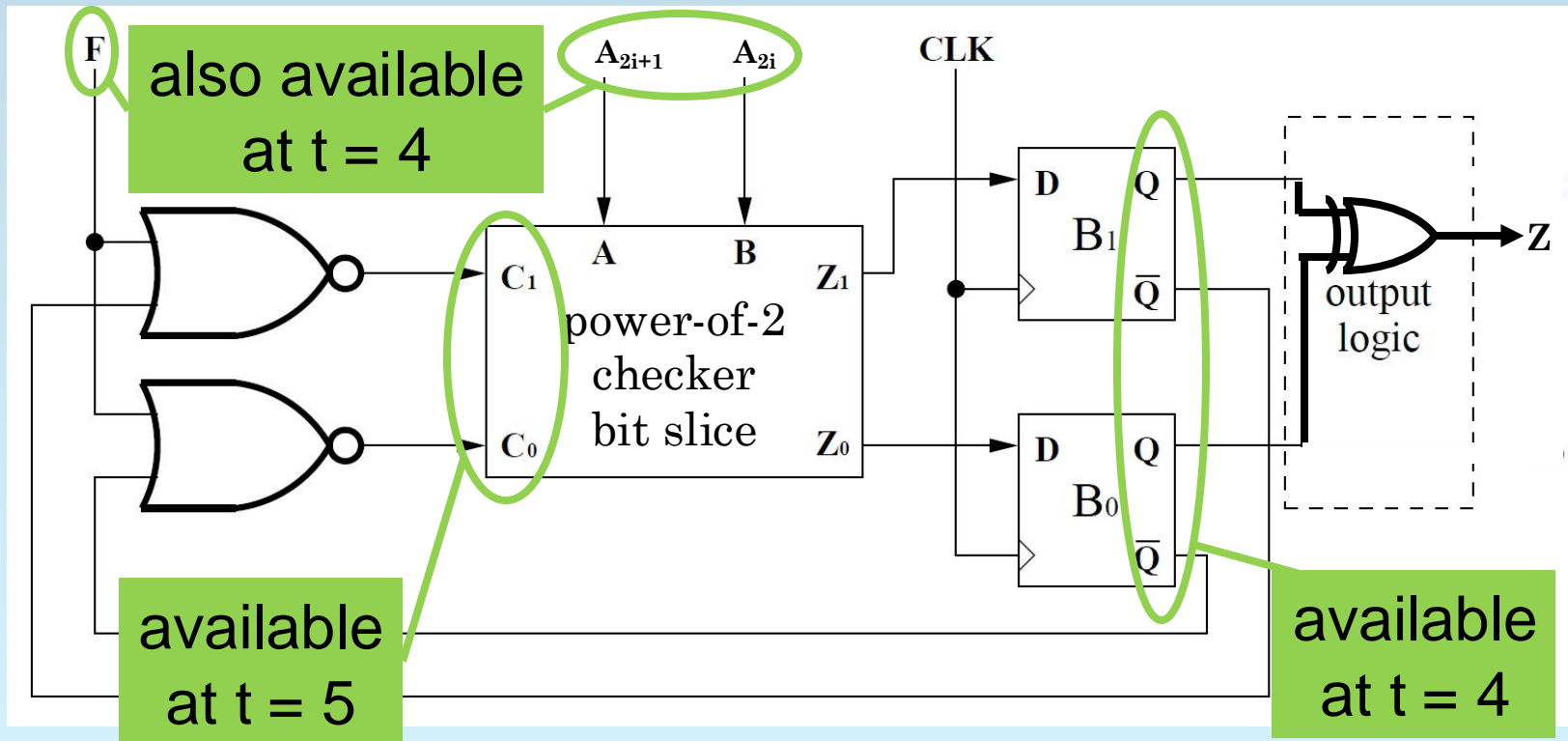
A serial design (independent of **N**) requires

- **21 2-input gates** (84 transistors),
- **2 3-input gates** (12 transistors),
- **4 inverters** (8 transistors), and
- **1 2-input XOR.**

The serial design is smaller for $N \geq 10$.

When Are Inputs Available?

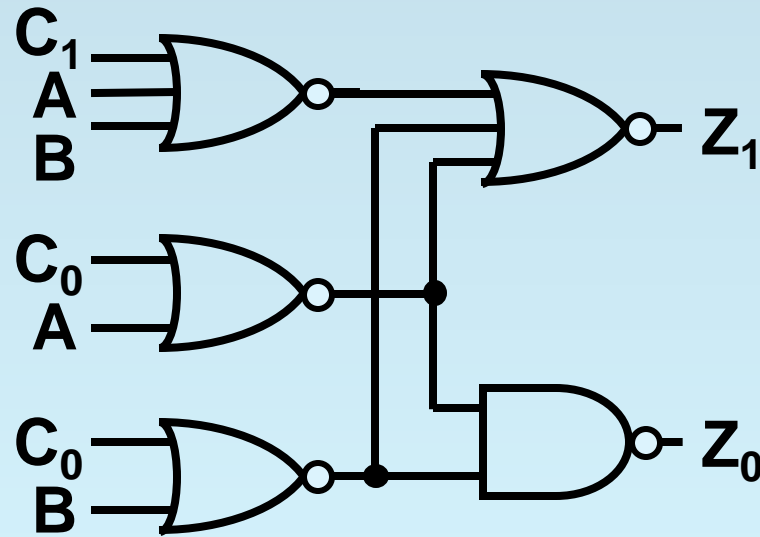
A rising edge arrives at $t = 0$ (gate delays).



Delay Analysis for the Bit Slice

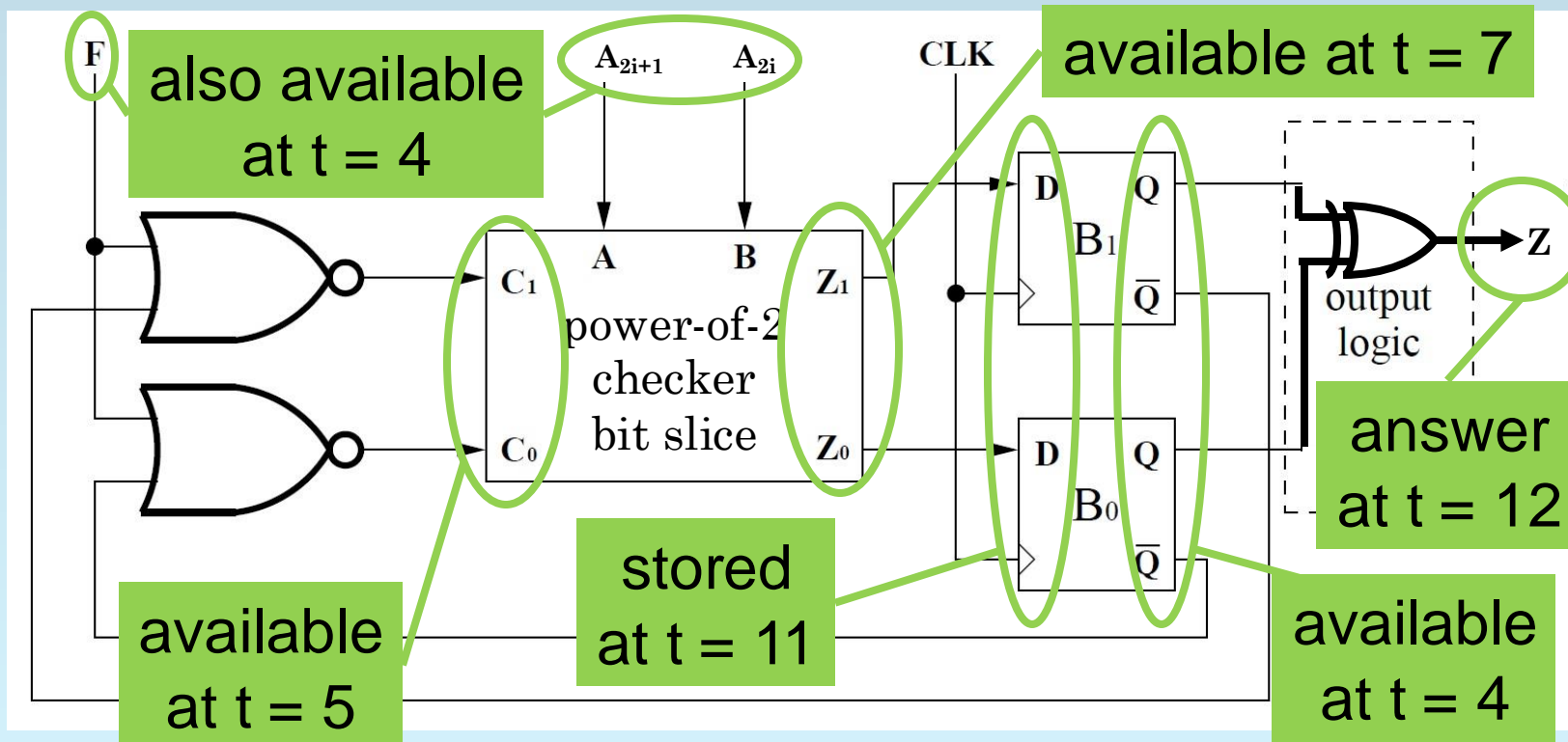
And delay?

2 on all paths.



When Are Results Stored?

A rising edge arrives at $t = 0$ (gate delays).



Serial Design is At Least 5.5x Slower

To handle **N-bit** operands, a bit-sliced design requires **$N + 1$ gate delays**.

For a serial design,

- the clock cycle must be **at least 11 gate delays**, and
- we must execute for **$N/2$** cycles, so
- **N-bit** operands require **at least $11N/2$ gate delays**.

The serial design is at least ~5.5x slower.

(And may be even slower!)