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

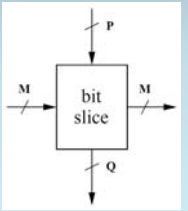
N-bit operands

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).



Parameter Values for a Serial Comparator

Comparator parameters

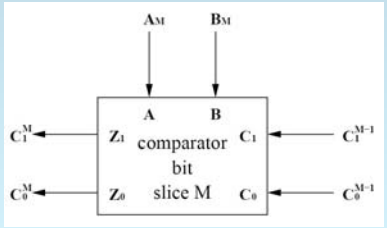
N-bit operands

P = 2

Q = 0

M = 2

R = 2



Initialization of a Serial Comparator

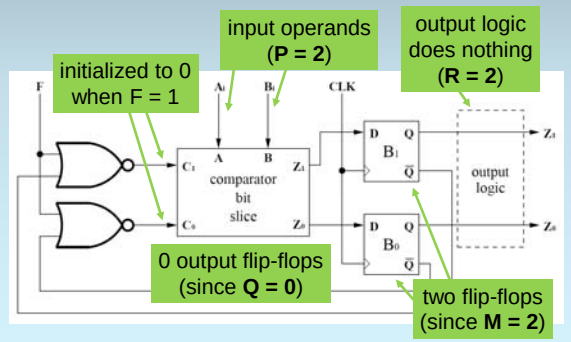
Our comparator bit slice uses the representation shown here to pass information between slices.

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

A Serial Comparator Consists of Three Parts

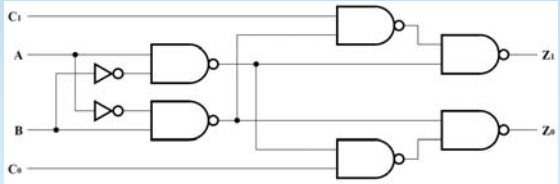
Let's analyze the area of a serial comparator.

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.

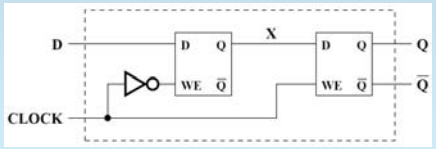
We have:
- one bit slice, → six 2-input NAND and two inverters
- two flip-flops, and
- two 2-input NOR gates (selection logic).

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, → six 2-input NAND and two inverters
- two flip-flops, and
- two 2-input NOR gates (selection logic). → 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 ≥ 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 ≥ 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.

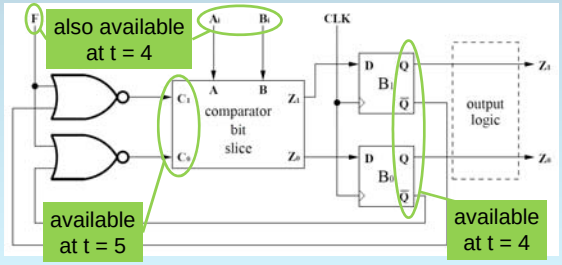
We can count gate delays
- in the bit slice, and
- for the selection logic.

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).



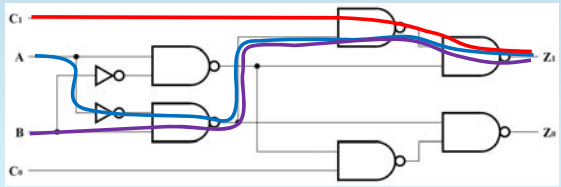
Delay Analysis for the Bit Slice

A to Z_1: 3 gate delays (ignoring NOT)

C_1 to Z_1: 2 gate delays

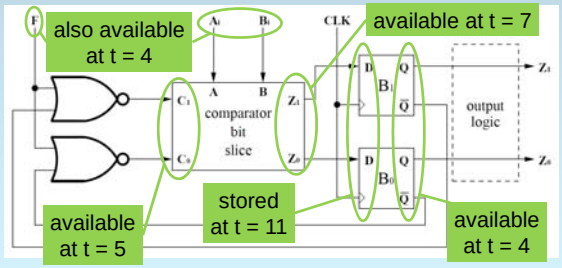
B to Z_1: 3 gate delays

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.

