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

ECE 120: Introduction to Computing

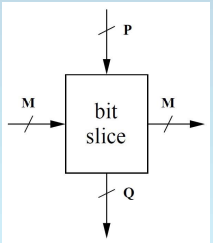
Serialization of Bit-Sliced Designs

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?

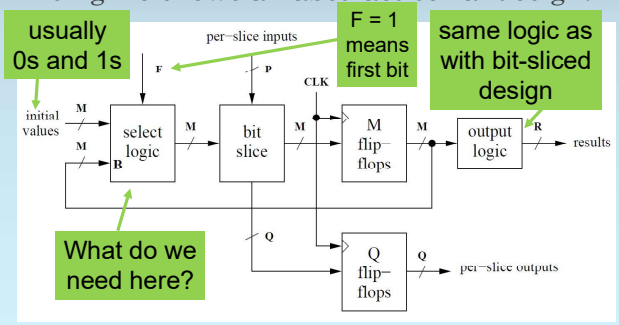
That bit slice has no previous bit slice, so instead the **M-bit input is 0s and 1s**.

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**).

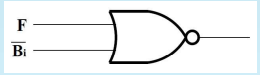
And write a truth table for C_i.

From the table, we can write

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

F	C_i
0	B_i
1	0

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



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

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

F	C_i
0	B_i
1	1

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