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

ECE 120: Introduction to Computing

The Ripple Carry Adder

Build an Addition Device Based on Human Addition

Weeks ago, we talked about a
"hardware device" to perform addition.

Now, you're ready to design it.

Let's start by reviewing the human approach.

Basing a design on the human approach

- is usually the **easiest way**, and
- **often** leads to **a good design**, too.
- (Humans are pretty smart.)

Example: Addition of Unsigned Bit Patterns

Let's do an example with **5-bit unsigned**

```
   11
   01110  (14)
+  00100  (4)
---------
   10010  (18)
```

Good, we got the right answer!

Name Signals (Bits) for Our Hardware Design

Let's do an example with **5-bit unsigned**

$$\begin{array}{rr} \text{carry C} & 11000 \\ \text{A} & 01110 \\ \text{B} & +\ 00100 \\ \hline \text{sum S} & 10010 \end{array}$$

There is no "blank" bit.

Each 1-bit sum needs a C input.

Good, we got the right answer!

For least significant bit, C←0.

For other bits, C comes from next bit to right.

Inputs and Outputs for a Full (One-Bit) Adder

Think about adding a single bit (a column).

A **full adder*** has **three inputs**

- **A** (one bit of the number **A**)
- **B** (one bit of the number **B**)
- C_{in} (a carry input from the next least significant bit, or 0 for bit 0)

And a full adder produces **two outputs**

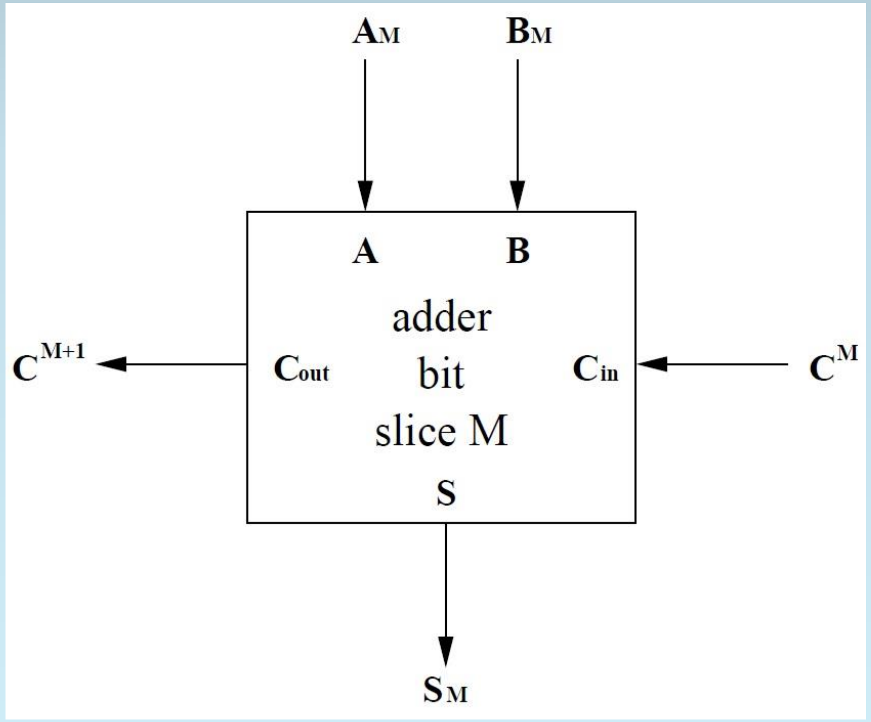
- C_{out} (a carry output for the next most significant bit)
- **S** (one bit of the sum **S**)

*A one-bit adder is called a "full adder" for historical reasons.
A "half adder" adds two bits instead of three.

Connecting the Full Adder to the N-Bit Problem

Consider bit **M** of the addition (bit 0 is on the right, bit 1 to the left of bit 0, and so forth).

We need to add A_M, B_M, and C^M to produce bit S_M, of the sum and bit C^{M+1}, the carry into bit **M+1** of the addition.



Write a Truth Table for Full Adder Outputs

Let's calculate the outputs for a full adder.

You may remember solving this truth table a few weeks ago.

But let's do it again…

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fill a K-map for C_{out} from the Truth Table

Now fill in the truth table for **C_{out}**.

C_{out}

$A \backslash BC_{in}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

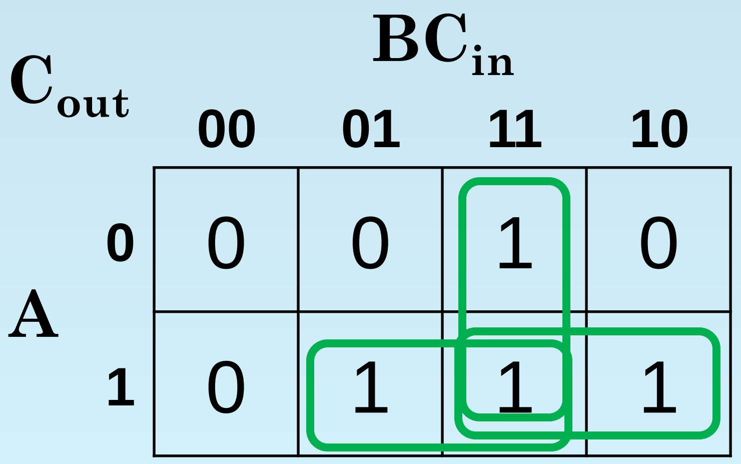
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Solve the K-map to Find C_{out}

And find the loops.

So we can write $\mathbf{C_{out} = AB + AC_{in} + BC_{in}}$

(called a majority function, by the way).



C_{out} — BC_{in} across, A down

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

The Sum is Best Written as an XOR

What about **S**? We can (of course) use another K-map.

But a K-map doesn't give us the best answer in this case (a rare case!).

S is 1 when an odd number of inputs are 1.

So $\mathbf{S = A \oplus B \oplus C}$.

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

* *

XOR Shows up as a Checkerboard Pattern

Here's the K-map for S. Notice the checkerboard pattern of the XOR.

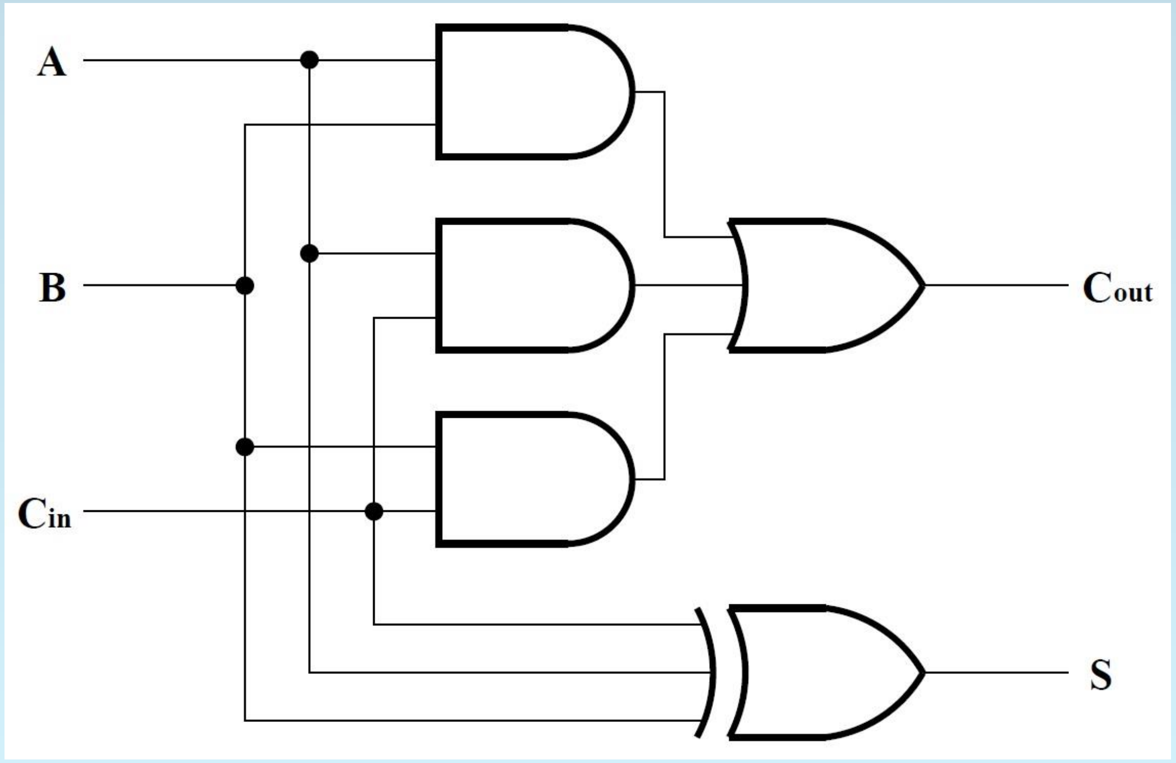
S

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Circuit for a Full Adder Using AND, OR, and XOR

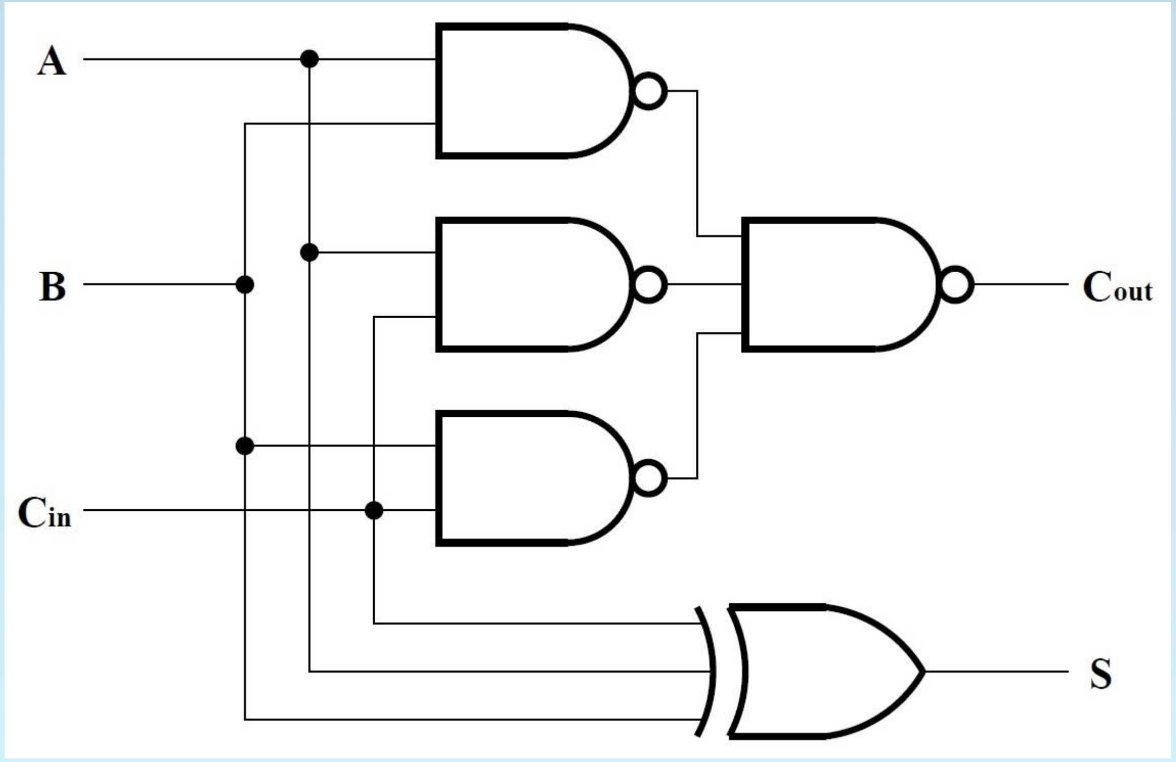
We can draw our full adder using AND, OR, and XOR.



CMOS Implementation Using NAND and XOR

In CMOS, we replace AND/OR with NAND/NAND.

The XOR remains as an XOR gate.



How Do We Add N Bits?

Use a full adder for each of the **N** columns.

Feed a **0** into C_{in} for the least significant bit.

C_{out} of the most significant bit
is the adder's carry out.

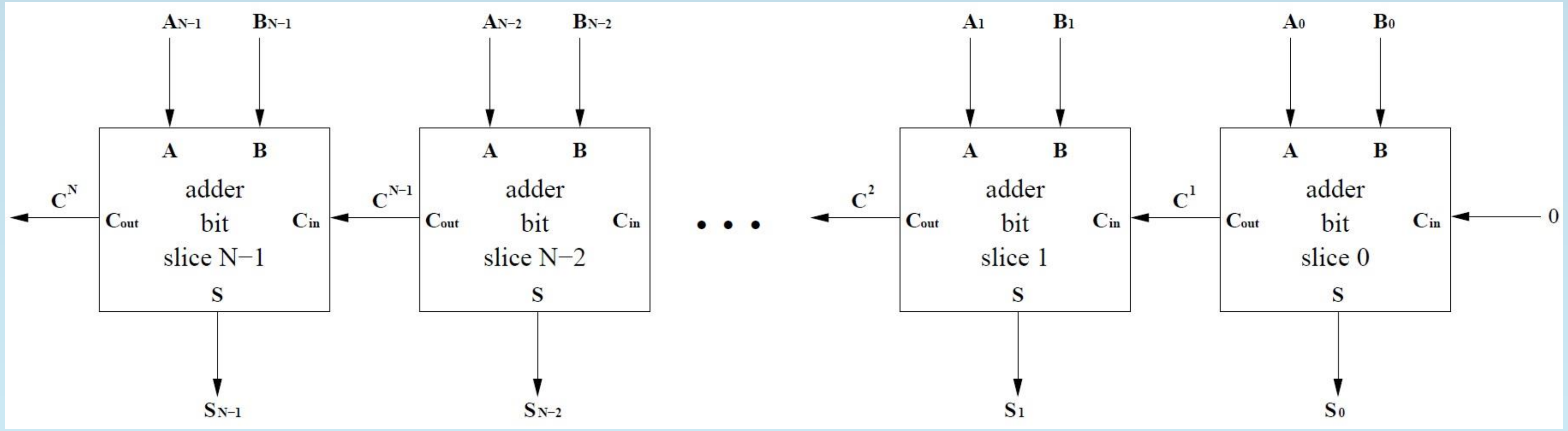
For the other carry signals, connect C_{out} of
each bit to C_{in} of the next most significant bit.

Divide the bits of **A** and **B**
amongst the full adders.

Collect the bits of **S** from the full adders.

Use N One-Bit Adders to Build an N-Bit Adder

The figure below illustrates construction of an **N**-bit adder from **N** full adders.



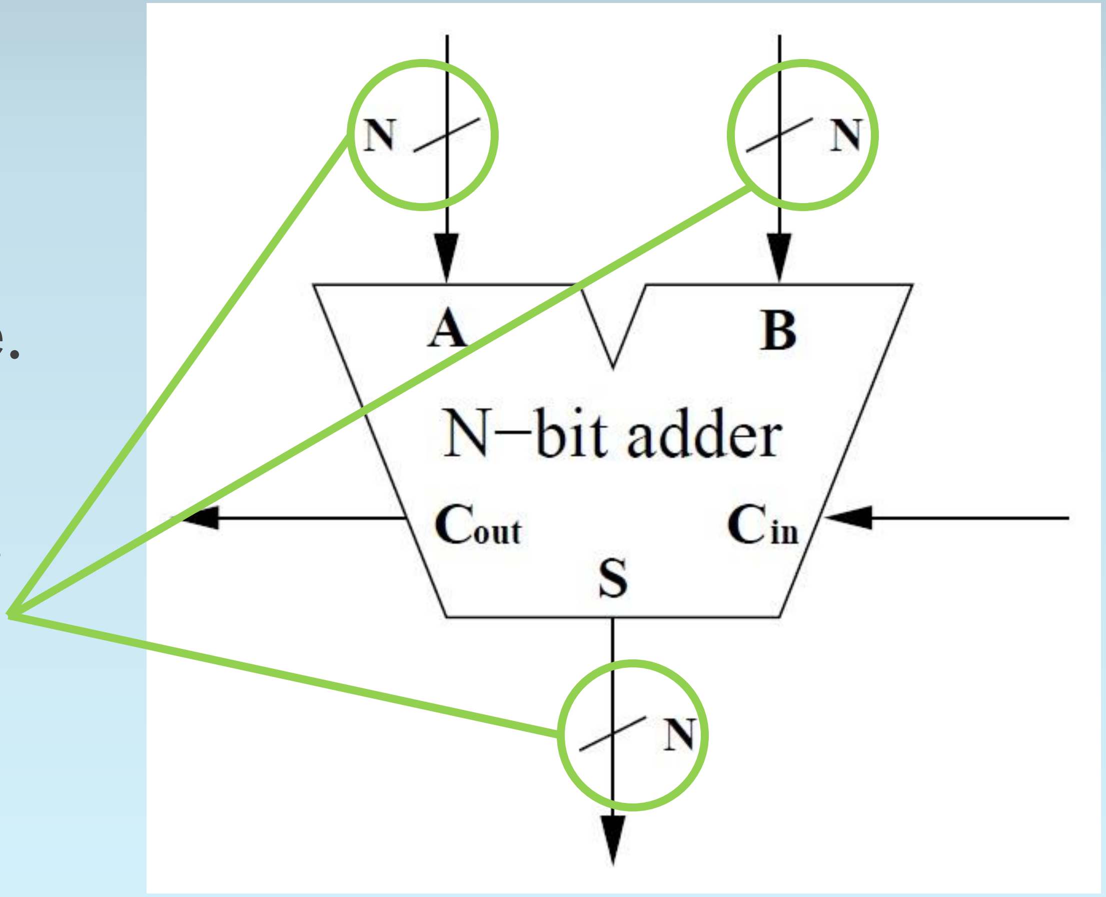
This approach is called a **ripple carry adder** because the carry ripples slowly from low to high. We also call it a bit-sliced adder.

Symbol for an N-Bit Adder

We draw an **N-bit adder** as shown here.

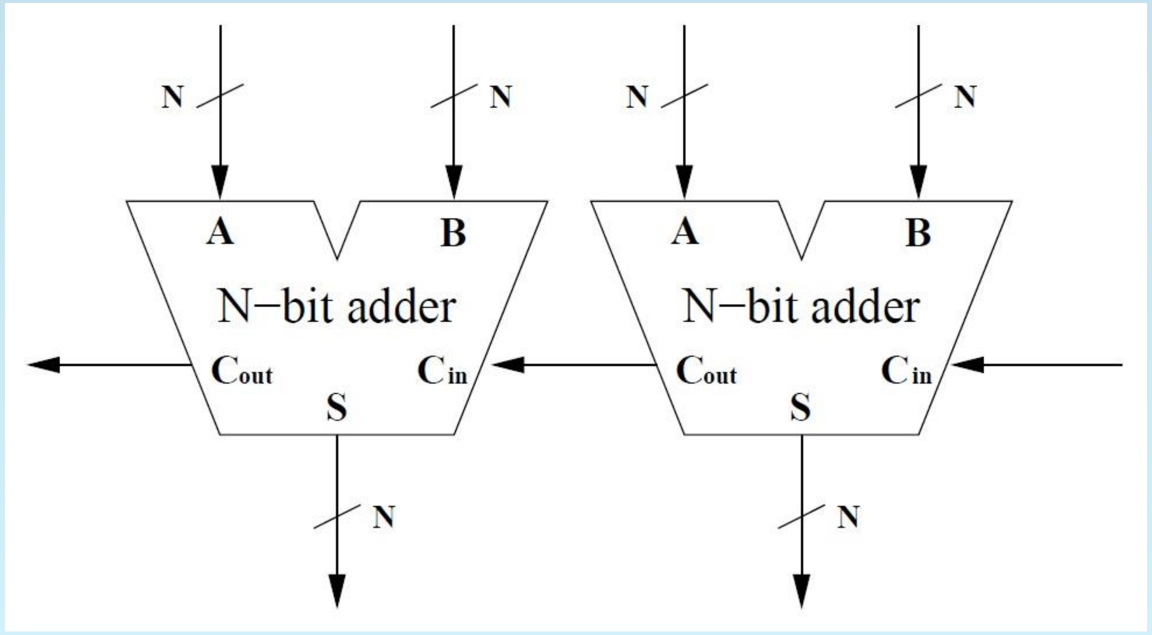
Note the shape.

Note also the crosshatch and bit width ("N") for multi-bit signals.



To Build a Bigger Adder, Just Connect C_{out} to C_{in}

We can build bigger adders by connecting adders together physically (as shown below) or virtually (by saving the carry out bit and using it as the carry in to the next adder).



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

ECE 120: Introduction to Computing

Bit-Sliced Designs

What’s the Theory Behind a Ripple Carry Adder?

Think for a moment about addition.

Can you add 2-digit numbers?

What about 5-digit numbers?

What about 5,000-digit numbers?

Does it matter if I add more digits?

Have you ever seen a proof that you’re correct?

What kind of proof would you need?

Multi-Digit Addition is Correct by Induction

Probably a **proof by induction**…

1. You know how to add 1-digit numbers. Verifying an addition table suffices.
2. **GIVEN that you can add N-digit numbers**, show (based, for example, on place value) that **you can add (N+1)-digit numbers**.

But you didn't know about proof by induction

- when you learned how to add,
- so you've probably never seen a proof.

The Ripple Carry Adder is Also Correct by Induction

When we designed a ripple carry adder, we also **assumed proof by induction**.

1. We know how to add one bit. We made a truth table (a binary addition table).
2. **GIVEN that we can build an N-bit adder**, show that we can build an **(N+1)-bit** adder by **attaching a full (1-bit) adder to an (N-bit) adder**.

Build an Addition Device Based on Human Addition

In ECE220, you will write **recursive functions**. These functions call themselves.

And you will use the same idea…

1. The answer for some base case (one or more **stopping conditions**) is known.
2. **GIVEN that we can write a function that works for input of size N**, show that we can write a function that works for size **(N+1)** by **handling the extra "1" and calling the function recursively for the "N"**.

The Three Contexts are the Same Mathematically

The approach is the same.

The part that sometimes confuses people (particularly for software/recursion, but sometimes also for hardware/bit slicing) is the ASSUMPTION in the inductive step.

You **must assume that the design works for N pieces** (bits, input size, or whatever).

All Three Approaches Require a “Leap of Faith”

You don’t need to design the system all at once for **N** (other than some base case).

In other words,

- you must make a “**leap of faith**” and
- **assume that your answer works**
- before you actually design it!

People sometimes have trouble making such an assumption, but it’s just a **standard part of an inductive proof**.

Bit Slicing Requires Problem Decomposition

Bit slicing works for problems that

- allow us to **break off a small part** of the problem,
- say **1 bit (or a few bits)**,
- and be able to **solve the full problem using the solution for the remaining part and the 1 bit.**

(That's the inductive step.)

Signals Between Bit Slices Must be Fixed (and Few)

For hardware, we also need

- to be able to **express the "answer"** for the remaining part
- **in a (small!) fixed number of bits**.

Otherwise, the number of inputs and outputs to the bit slice changes from slice to slice!

Examples of Problems that Allow Bit Slicing

- Addition / subtraction
- Comparison
- Check for power of 2
- Check for multiples (of 3, 7, and so forth)
- Division by constants
- Pattern matcher
- Bitwise logic operation

When Can't We Used Bit Slicing?

One example: **when the answer depends on ALL of the other bits** (can't summarize an answer for N bits).

For example, can you create a bit-sliced prime number identifier?

$\mathbf{A_{N-1}\ A_{N-2}\ \ldots\ A_5 \leftarrow (summary)\ 0\ 1\ 0\ 0\ 1}$

What information do you pass to bit 5?

All 5 bits? 01001? I have no idea!