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!

