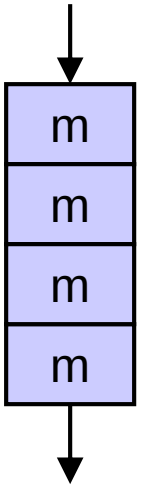
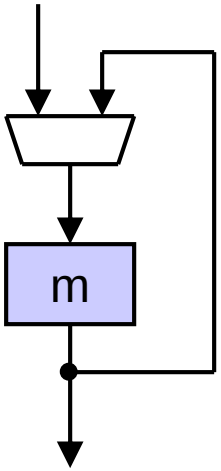
ADDERS & SUBTRACTORS

Arithmetic Blocks

- We'll first look at adder and multiplier
- Look at in "full" non-iterative view, but hardware may require smaller area and less performance so we can then "time-multiplex" a portion of the datapath

"full view"





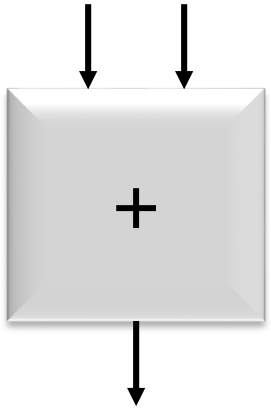
an iterative potentially functionally-identical implementation

Adders

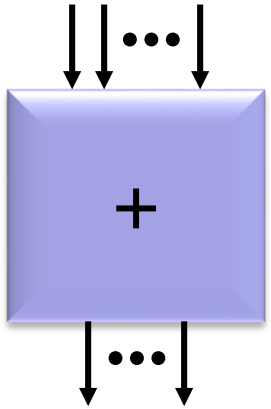
- One of the most fundamental arithmetic units
- Signal Growth Rule:
  width of *sum* = width of *input* + 1
  - for a 2-input adder
  - both inputs are of equal width
- Different styles produce faster or smaller circuits; many types are useful, not just the fastest ones
  - Ex: add two 3-bit numbers. Likely choose the simplest (slowest) architecture
  - Ex: 32-bit adder in critical path of a datapath. Likely choose the fastest architecture

Two Broad Classes of Adders

- Carry-propagate adder (CPA)
  - What we think of as normal addition
  - Output is a single word
  - Carry must effectively propagate across the entire word

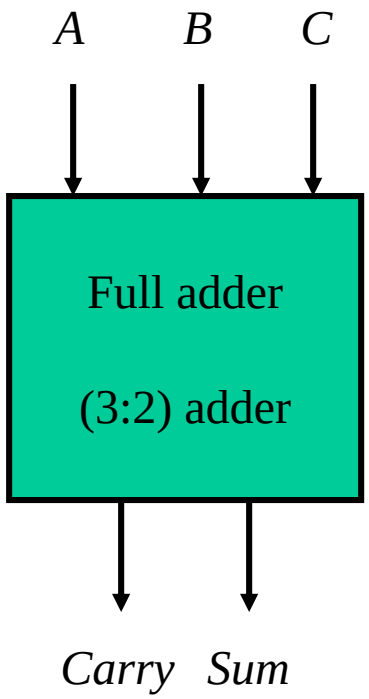


- Carry-save adder (sometimes CSA)
  - *Sum* is not in a "normal" single-word format
    - Output is in a redundant "carry-save" format
    - Output consists of 2+ words
    - Input consists of 3+ words



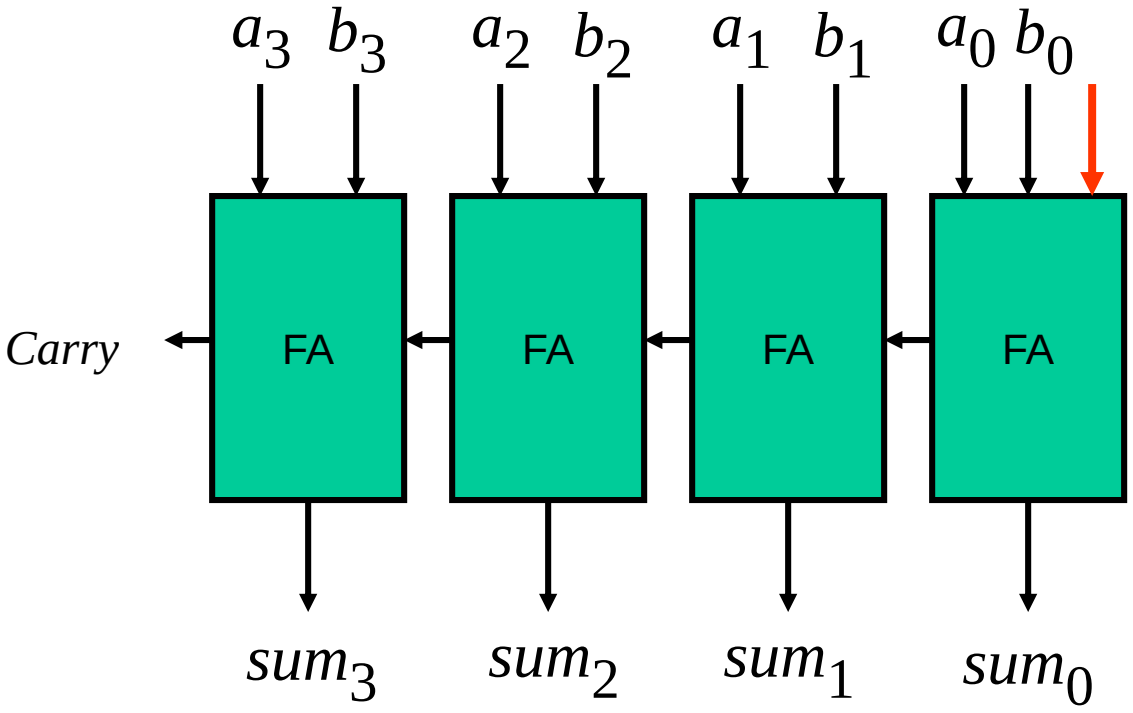
Full Adder

- A fundamental adder building block
- Adds three bits of equal weight
- *Carry* has a 2× higher-significance or positional weight than *Sum*



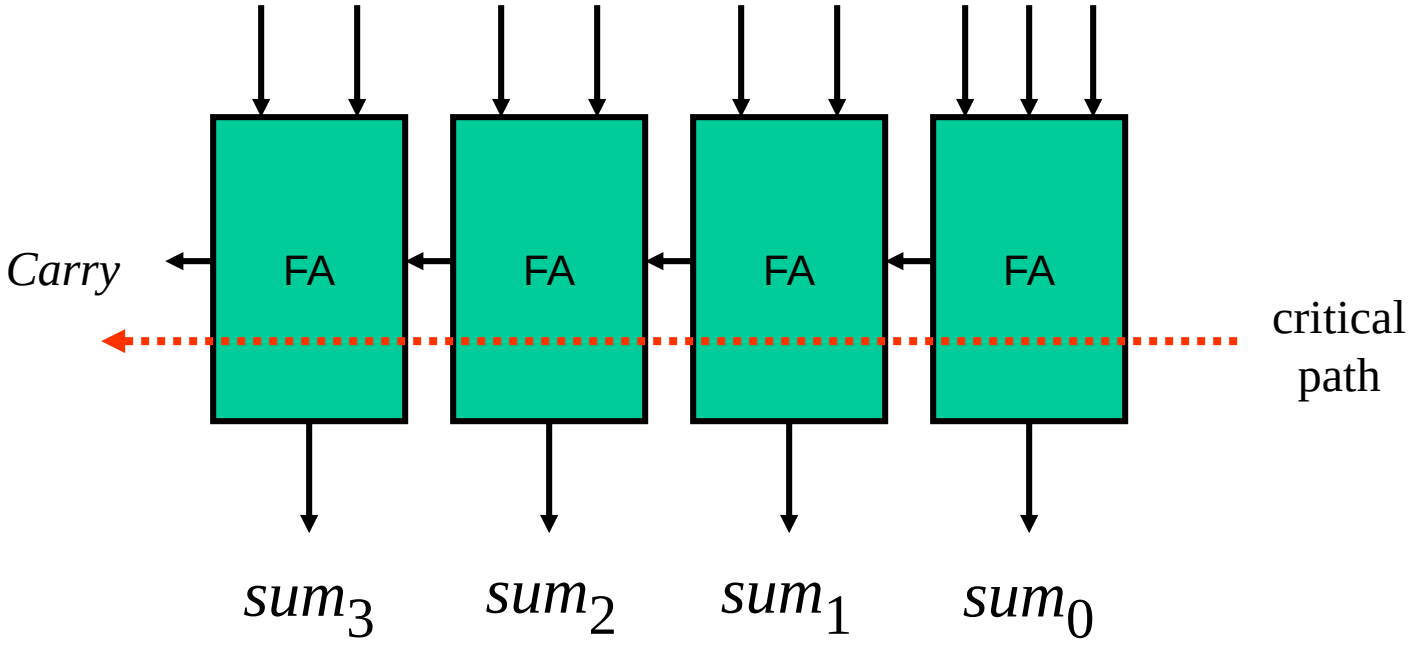
1) Ripple Carry Adder

- We can chain together or "bit slice" full adders to add two numbers
- Note the extra input in the LSB position



1) Ripple Carry Adder

- The *carry* **ripples** through the chain of full adders



1) Ripple Carry Adder

- Simplest adder
- Smallest adder
- Slowest adder (for wide words)
  - However, for narrow word-widths, it can be fast!
- We can view most (all?) other CPAs as improved ripple-carry adders

Faster Adders

- The entire goal to make faster adders is to resolve the carry across the entire adder structure more quickly
- A few common faster CPAs:
  1) Carry Select
     - Speculatively add and select later
  2) Carry Lookahead
     - Look at how a carry propagates through a group of bits
  3) Conditional-sum (recursive carry select)
  4) Carry skip
  5) Other parallel prefix adders
     - Kogge-Stone, 1973
     - Brent-Kung, 1982
     - etc.

Subtraction

- Subtraction *requires* a signed number format
  - Ex: 2 – 3 = –1
- 2's complement is the preferred format for fixed-point because subtraction with it is straightforward
- 2's complement subtraction is implemented with a slightly-modified normal adder
  - `A - B = A + (-B)`
  - Recall that for 2's complement numbers,
    `-B = (~B) + 1`
  - So now we have
    `A - B = A + (~B) + 1`
- Signal growth is the same as with addition

Subtraction

- It is typically easy to find a place to add a "1" in the LSB position