Plan for the rest of the term.

- Today: Faster Adders (CLA)
- Friday: Bus-based datapath
- Monday (June 12th): Mealy machines
- Wednesday: Errors, maybe some Verilog/embedded stuff.
- Friday: Catchup, FPGAs
- Monday (June 19th): Class review and practice problems.
- Thursday (June 22nd) Final exam 4-6pm

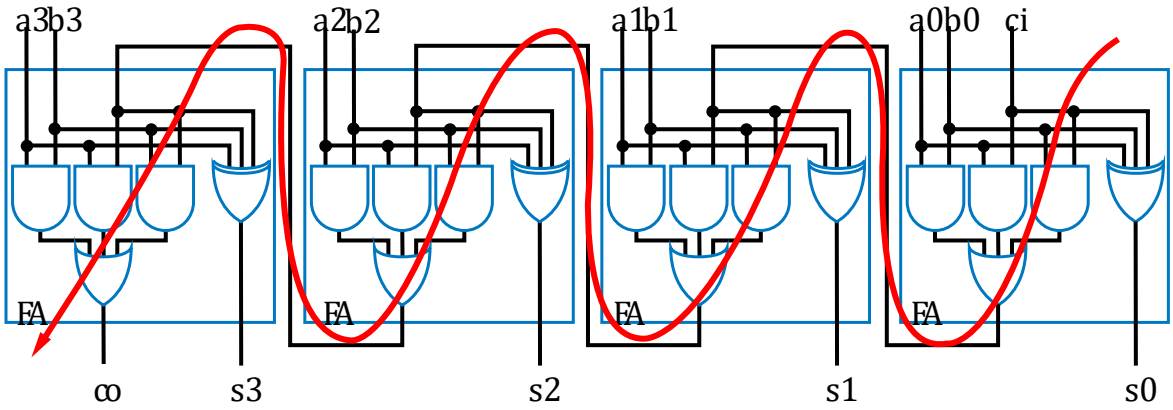
Faster adders (3.4)

Ripple-carry adders are slow.

- How many gate delays do we have for a 4-bit ripple-carry adder (in the worst case)?
- For a 32-bit RCA?

They are however pretty small.

- How many gates total for a 32-bit RCA?

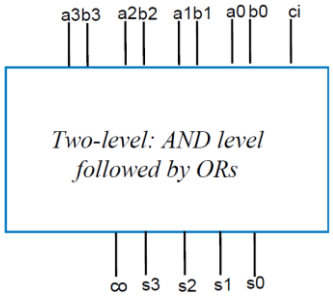


a

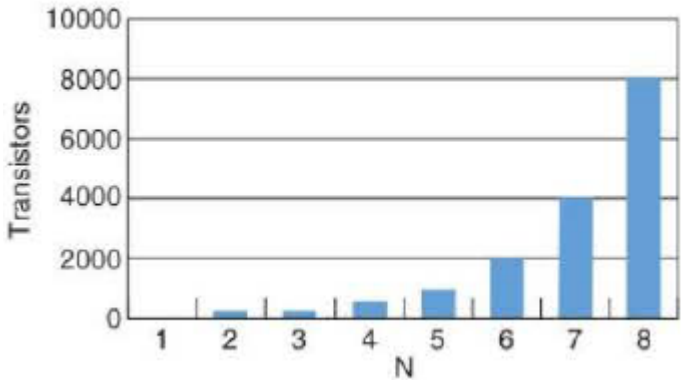
One other option is that we could "just" write out the truth table for the adder (9 inputs for a 4-bit adder) and write the sum-of-products for the 4-bit adder.

- What would be our gate delay?
- How many gates would there be (this one is hard and we can't really figure it out easily, but guess).

Pretty clearly 32-bit adders done as sum-of-products would be huge (we'll discuss how huge later). And if we were limited to 2-input gates, things get crazy quickly.

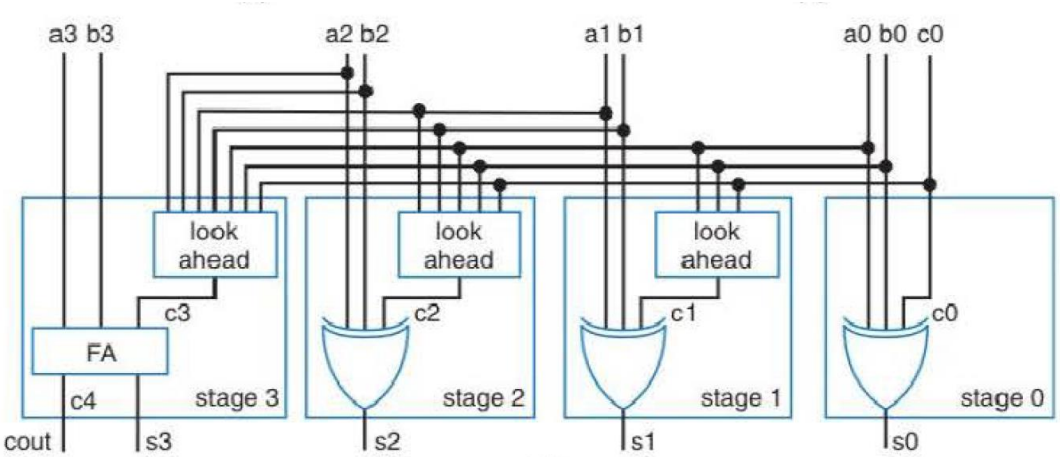


Graph of number of *transistors* needed for a 2-level (sum-of-products in this case) N-bit adder. From our textbook's author (Figure 4.24)



Start on lookahead

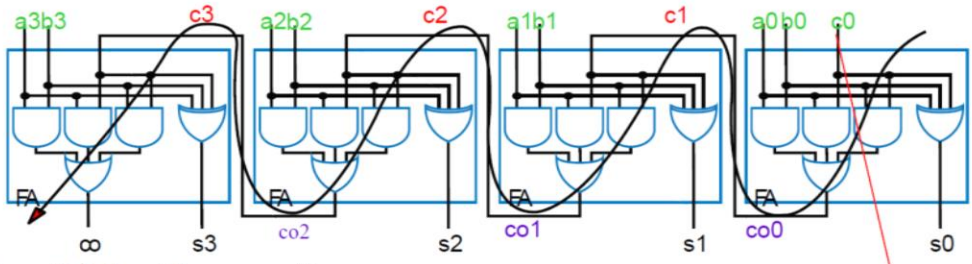
What we would like is a compromise. Ripple-carry is slow (linear in N). Sum-of-products is huge (probably exponential in N—think about the size of the truth table). We want something in between. Let's consider one option:



There is no rippling of the carry—we could "just" compute the "lookahead" without looking at previous stages. We just add some logic that figures out if there will be a carry in. *That* lookahead box, in theory could be 2-level logic. But as you can see, computing "c3" involves looking at c0, a0, b0, a1, b1, a2, and b2. Which sounds like our sum-of-products adder. And doing a 32-bit one seems crazy and about as big as our sum-of-products adder.

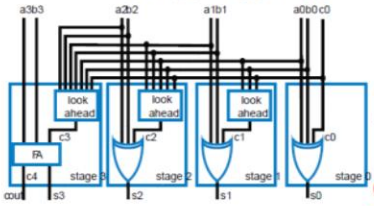
Faster Adder – (Bad) Attempt at "Lookahead"

- Want each stage's carry-in bit to be function of external inputs only (*a*'s, *b*'s, or *c0*)



- Recall full-adder equations:
  - s = a xor b
  - c = bc + ac + ab

Stage 0: Carry-in is already an external input: **c0**



Stage 1: c1=co0

co0= b0c0 + a0c0 + a0b0

c1 = b0c0 + a0c0 + a0b0

Stage 2: c2=co1

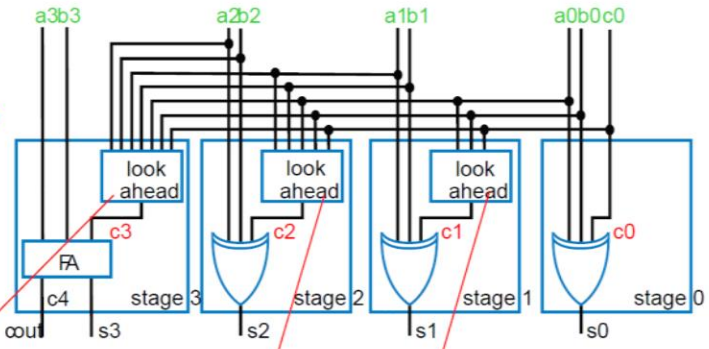
co1 = b1c1 + a1c1 + a1b1

c2 = b1c1 + a1c1 + a1b1

c2 = b1(b0c0 + a0c0 + a0b0) + a1(b0c0 + a0c0 + a0b0) +a1b1

c2 = b1b0c0 + b1a0c0 + b1a0b0 + a1b0c0 + a1a0c0 + a1a0b0 + a1b1

- **Carry** lookahead logic function of external inputs
  - No waiting for ripple
- Problem
  - Equations get too big
  - Not efficient
  - Need a better form of lookahead

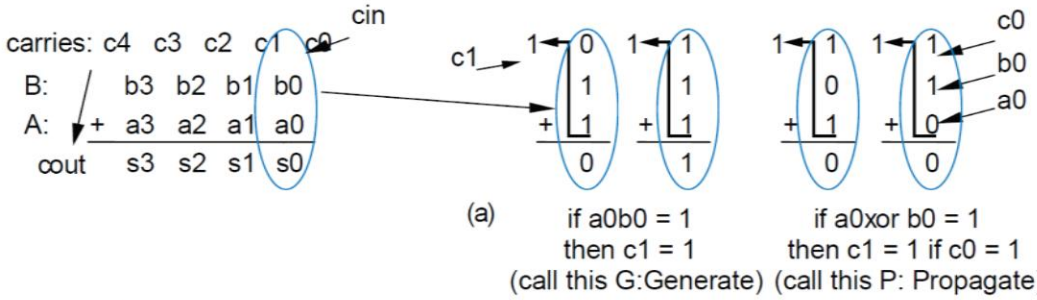


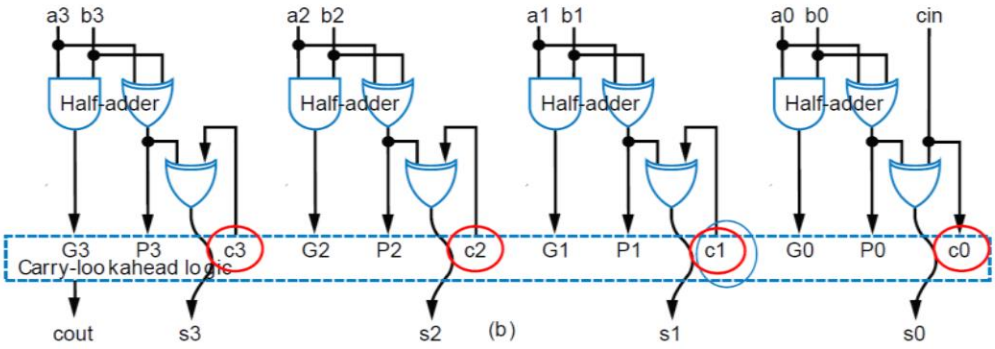
c1 = b0c0 + a0c0 + a0b0

c2 = b1b0c0 + b1a0c0 + b1a0b0 + a1b0c0 + a1a0c0 + a1a0b0 + a1b1

c3 = b2b1b0c0 + b2b1a0c0 + b2b1a0b0 + b2a1b0c0 + b2a1a0c0 + b2a1a0b0 + b2a1b1 + a2b1b0c0 + a2b1a0c0 + a2b1a0b0 + a2a1b0c0 + a2a1a0c0 + a2a1a0b0 + a2a1b1 + a2b2

- Have each stage compute two terms
  - ***Propagate***: P = a xor b
  - ***Generate***: G = ab
- Compute lookahead from *P* and *G* terms, *not from external inputs*
  - Why *P* & *G*? Because the logic comes out much simpler
    - Very clever finding; not particularly obvious though
    - Why those names?
      - *G*: If *a* and *b* are 1, carry-out will be 1 – "generate" a carry-out of 1 in this case
      - *P*: If only one of *a* or *b* is 1, then carry-out will equal the carry-in – propagate the carry-in to the carry-out in this case





- With *P* & *G*, the carry lookahead equations are much simpler
  - Equations before plugging in
    - c1 = G0 + P0c0
    - c2 = G1 + P1c1
    - c3 = G2 + P2c2
    - cout = G3 + P3c3

After plugging in:

c1 = G0 + P0c0

c2 = G1 + P1c1 = G1 + P1(G0 + P0c0)
c2 = G1 + P1G0 + P1P0c0

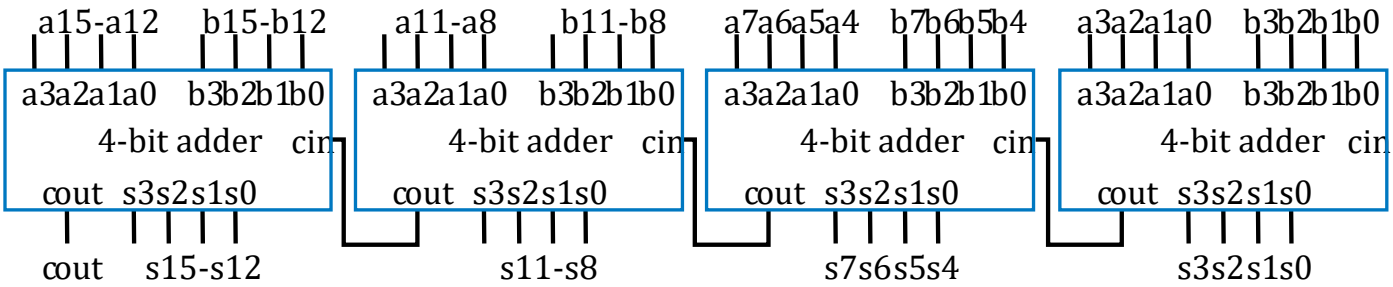
c3 = G2 + P2c2 = G2 + P2(G1 + P1G0 + P1P0c0)
c3 = G2 + P2G1 + P2P1G0 + P2P1P0c0

cout = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0c0

Much simpler than the "bad" lookahead

64

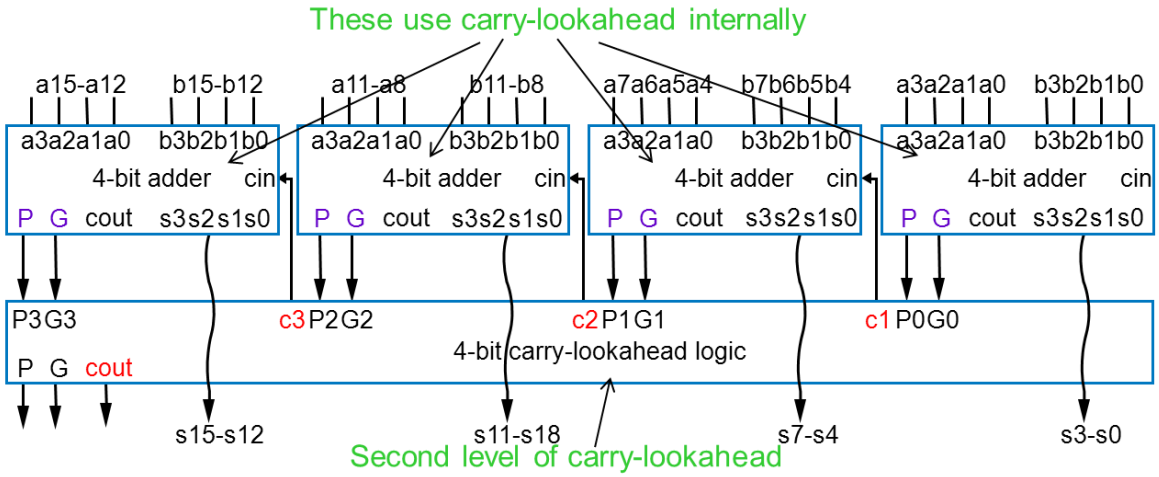
Making larger adders seems hard. The amount of work for each carry bit keeps growing. We could just limit ourselves to a 4-bit adder like this and then ripple the 4-bit adders (as shown below). That might be an interesting compromise between size and speed.



But we'd like to do better than that. This marginally might speed up things (at the cost of more logic) but it's only a marginal improvement.

Carry-lookahead (again)

Or we could try to get tricky. Obviously (?), we could use the lookahead logic again.



Adder type (16-bit)	CLA
Gate count	
Gate delays	

Adder type (16-bit)	CLA
Gate-input count	
Log_2(gate-input) delays	