

Plan for the rest of the term.

- Today: Finish Transistors, start (and maybe finish) fast adders.
- Friday: Mealy Machines, Quine-McClusky
- Monday (June 13th): Finish QM, Bus-based computers
- Wednesday: FPGAs
- Friday: SPI/Verilog
- Monday (June 20th): Class review and practice problems.

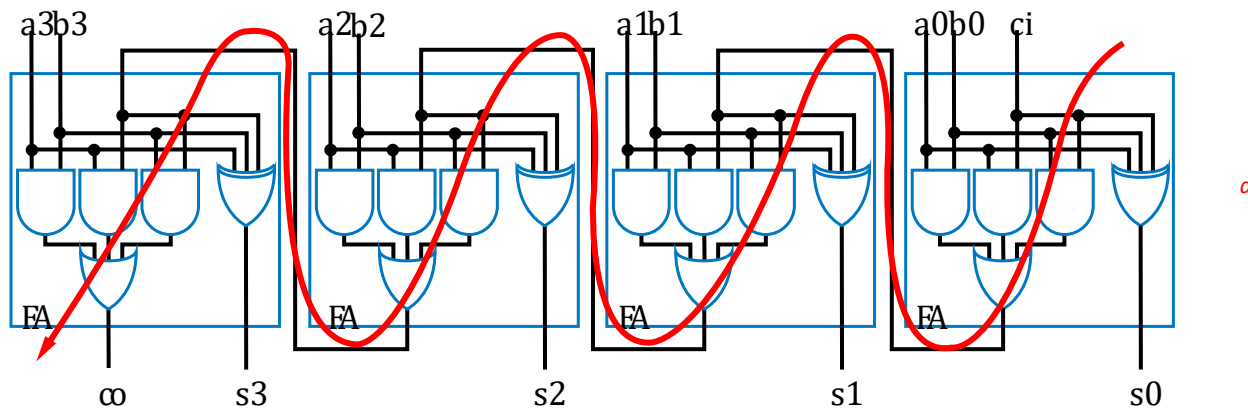
Faster adders (3.4)

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.

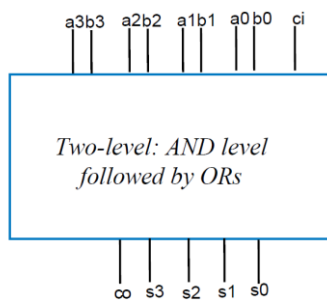
- How many gates total for a 32-bit RCA?



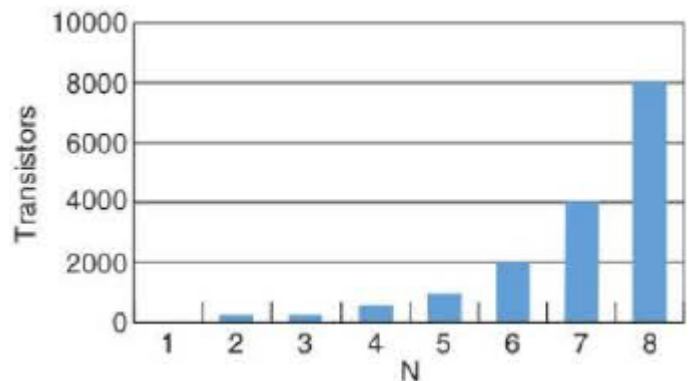
One other option is that we could “just” write out the truth table for the adder (9 inputs for a 4-bit adder) and write the sum-of-products for the 4-bit adder.

- 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.

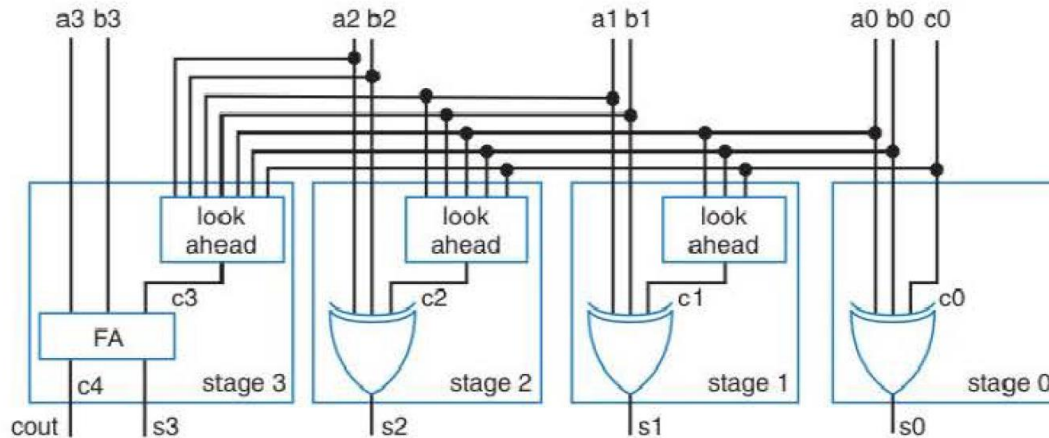


Graph of number of *transistors* needed for a 2-level (sum-of-products in this case) N-bit adder. From our textbook’s author (Figure 4.24)



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:



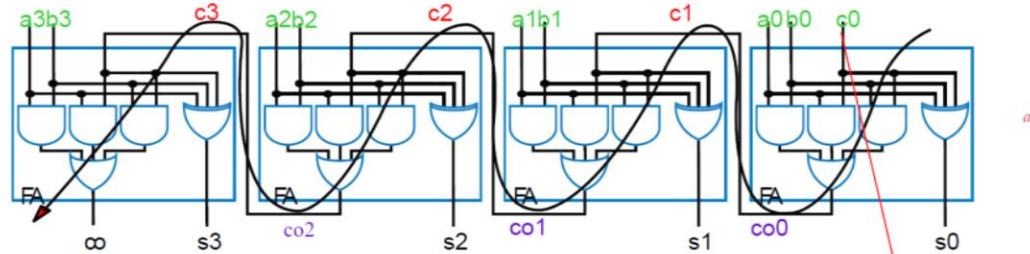
There is no rippling of the carry—we could “just” compute the “lookahead” without looking at previous stages. We just add some logic that figures out if there will be a carry in. *That* lookahead box, in theory could be 2-level logic. But as you can see, computing “ c_3 ” involves looking at c_0 , a_0 , b_0 , a_1 , b_1 , a_2 , and b_2 . 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.

Let's take a quick measure of delay and size in each of these. We will estimate delay by assigning a delay of a gate equal to $\log_2(\text{number of inputs})$. We will *estimate* area of a gate as being just the number of inputs. We'll have to make some guesses to do so.

- What is the expected delay of a 4-bit RCA? Its area?
 - How about 16 bit?
- What is the expected delay of the 4-bit “hybrid CLA” above? Its area?
 - How about 16 bit?

Faster Adder – (Bad) Attempt at “Lookahead”

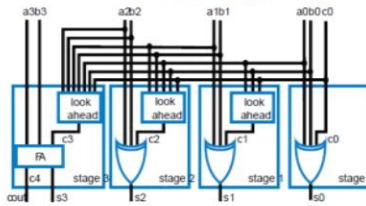
- Want each stage's carry-in bit to be function of external inputs only (a's, b's, or c0)



- Recall full-adder equations:

- $s = a \text{ xor } b$
- $c = bc + ac + ab$

Stage 0: Carry-in is already an external input: c_0



Stage 1: $c_1 = c_0$

$$c_0 = b_0c_0 + a_0c_0 + a_0b_0$$

$$c_1 = b_0c_0 + a_0c_0 + a_0b_0$$

Stage 2: $c_2 = c_1$

$$c_1 = b_1c_1 + a_1c_1 + a_1b_1$$

$$c_2 = b_1c_1 + a_1c_1 + a_1b_1$$

$$c_2 = b_1(b_0c_0 + a_0c_0 + a_0b_0) + a_1(b_0c_0 + a_0c_0 + a_0b_0) + a_1b_1$$

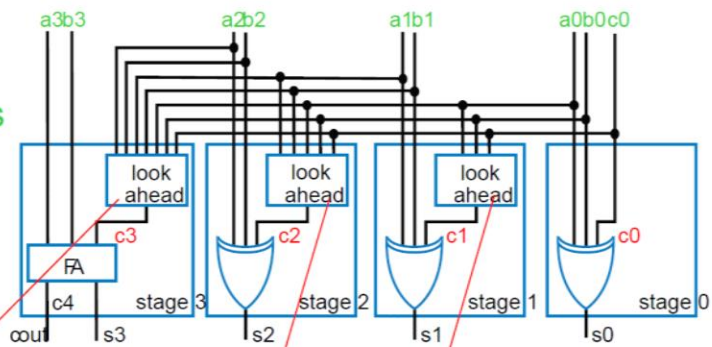
$$c_2 = b_1b_0c_0 + b_1a_0c_0 + b_1a_0b_0 + a_1b_0c_0 + a_1a_0c_0 + a_1a_0b_0 + a_1b_1$$

- Carry lookahead logic function of external inputs

- No waiting for ripple

- Problem

- Equations get too big
- Not efficient
- Need a better form of lookahead

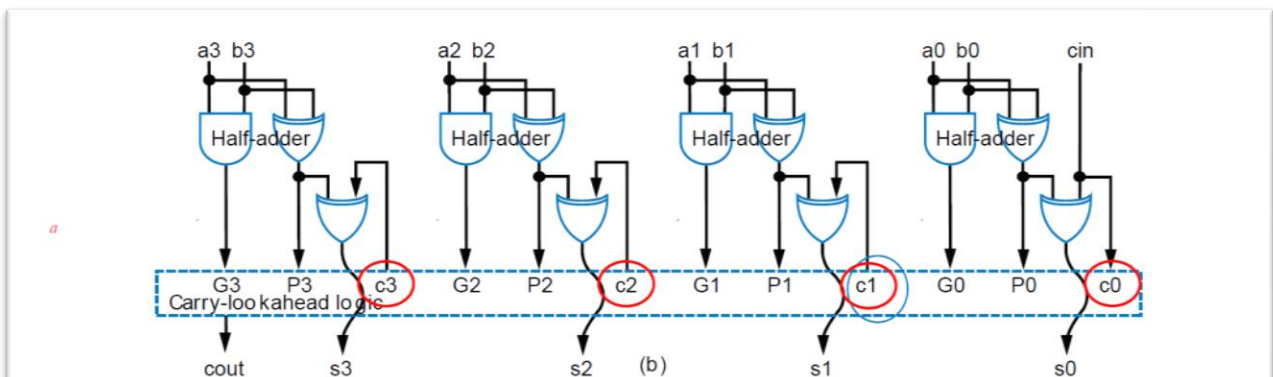
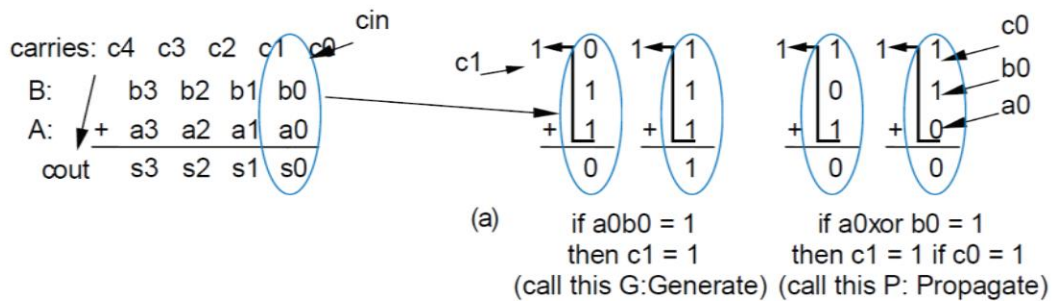


$$c_1 = b_0c_0 + a_0c_0 + a_0b_0$$

$$c_2 = b_1b_0c_0 + b_1a_0c_0 + b_1a_0b_0 + a_1b_0c_0 + a_1a_0c_0 + a_1a_0b_0 + a_1b_1$$

$$c_3 = b_2b_1b_0c_0 + b_2b_1a_0c_0 + b_2b_1a_0b_0 + b_2a_1b_0c_0 + b_2a_1a_0c_0 + b_2a_1a_0b_0 + b_2a_1b_1 + a_2b_1b_0c_0 + a_2b_1a_0c_0 + a_2b_1a_0b_0 + a_2a_1b_0c_0 + a_2a_1a_0c_0 + a_2a_1a_0b_0 + a_2a_1b_1 + a_2b_2$$

- Have each stage compute two terms
 - **Propagate:** $P = a \text{ xor } b$
 - **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
 - $c_1 = G_0 + P_0c_0$
 - $c_2 = G_1 + P_1c_1$
 - $c_3 = G_2 + P_2c_2$
 - $\text{cout} = G_3 + P_3c_3$

After plugging in:

$$c_1 = G_0 + P_0c_0$$

$$c_2 = G_1 + P_1c_1 = G_1 + P_1(G_0 + P_0c_0)$$

$$c_2 = G_1 + P_1G_0 + P_1P_0c_0$$

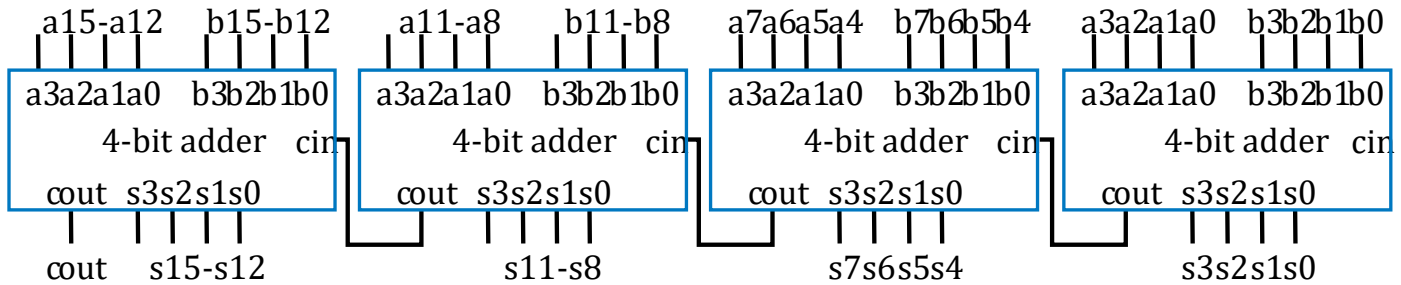
$$c_3 = G_2 + P_2c_2 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0c_0)$$

$$c_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0c_0$$

$$\text{cout} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$$

Much simpler than the “bad” lookahead

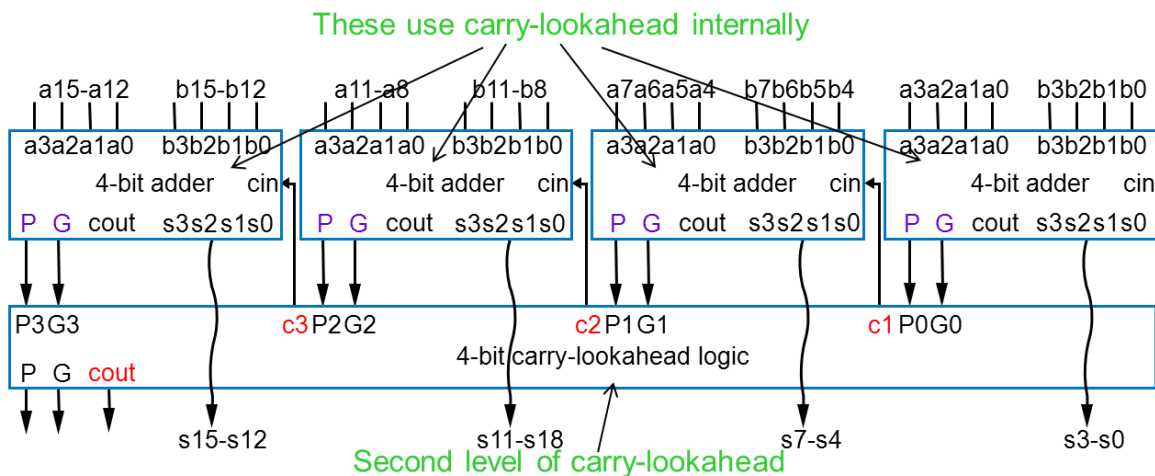
Making larger adders seems hard. The amount of work for each carry bit keeps growing. We could just limit ourselves to a 4-bit adder like this and then ripple the 4-bit adders (as shown below). That might be an interesting compromise between size and speed.



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	
Log₂(gate-input) delays	