ECE 274 Digital Logic – Fall 2008

Optimization and Tradeoffs

Carry-Lookahead Adders

Digital Design 6.4

THE UNIVERSITY OF ARIZONA
TUCSON ARIZONA

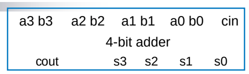
Digital Design

Chapter 6:
Optimization and Tradeoffs

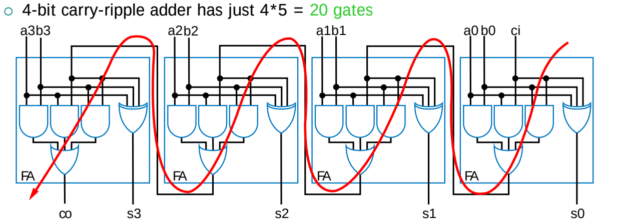
Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.
http://www.ddvahid.com

Carry-Lookahead Adder
Faster Adder

- Built carry-ripple adder in Ch 4
  - Similar to adding by hand, column by column
  - Con: Slow
    - Output is not correct until the carries have rippled to the left
    - 4-bit carry-ripple adder has 4*2 = 8 gate delays
  - Pro: Small
    - 4-bit carry-ripple adder has just 4*5 = 20 gates

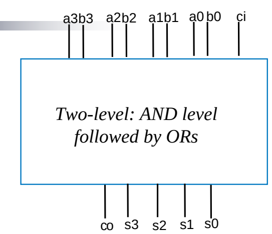


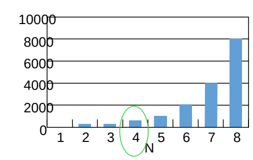
carries:		c3	c2	c1	cin
B:		b3	b2	b1	b0
A:	+	a3	a2	a1	a0
	cout	s3	s2	s1	s0



Carry-Lookahead Adder
Faster Adder

- Faster adder – Use two-level combinational logic design process
  - Recall that 4-bit two-level adder was big
  - Pro: Fast
    - 2 gate delays
  - Con: Large
    - Truth table would have $2^{(4+4)}$ =256 rows
    - Plot shows 4-bit adder would use about 500 gates
- Is there a compromise design?
  - Between 2 and 8 gate delays
  - Between 20 and 500 gates

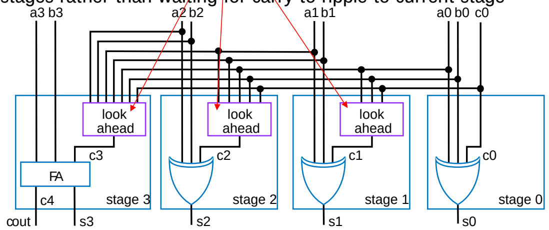




Carry-Lookahead Adder

Faster Adder – (Bad) Attempt at "Lookahead"

- Idea
  - Modify carry-ripple adder – For a stage's carry-in, don't wait for carry to ripple, but rather *directly compute* from inputs of earlier stages
    - Called "lookahead" because current stage "looks ahead" at previous stages rather than waiting for carry to ripple to current stage

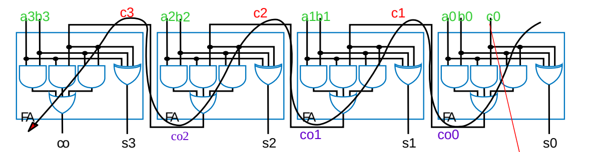


Notice – no rippling of carry

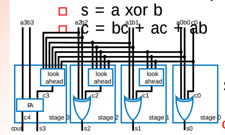
Carry-Lookahead Adder

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

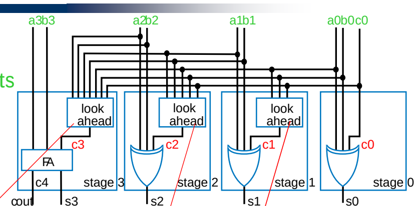
c2 = b1b0c0 + b1a0c0 + b1a0b0 + a1b0c0 + a1a0c0 + a1a0b0 + a1b1

Continue for c3

Carry-Lookahead Adder

Faster Adder – (Bad) Attempt at "Lookahead"

- 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

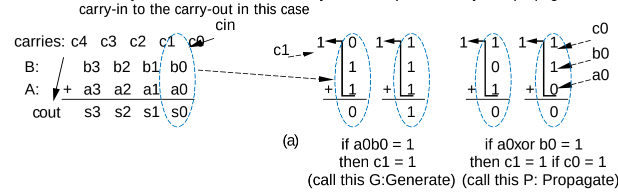
c2 = b1b0c0 + b1a0c0 + b1a0b0 + a1b0c0 + a1a0c0 + a1a0b0 + a1b1

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

Carry-Lookahead Adder

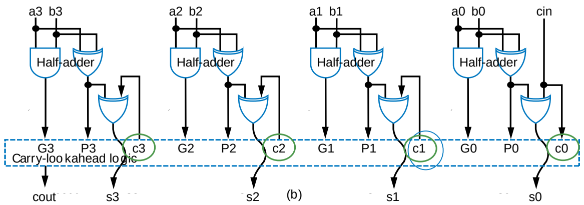
Better Form of Lookahead

- 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



Carry-Lookahead Adder

Better Form of Lookahead



- 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

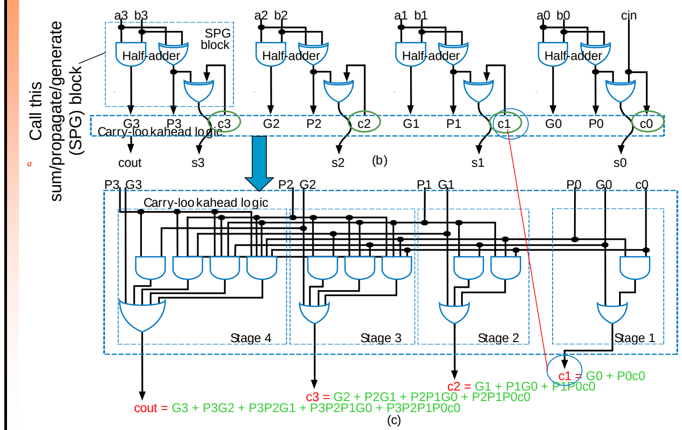
c3 = G2 + P2c2 = G2 + P2(G1 + P1G0 + P1P0c0)
c3 = G2 + P2G1 + P2P1G0 + P2P1P0c0

cout = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0c0

Much simpler than the “bad” lookahead

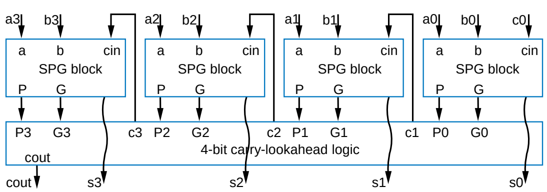
Carry-Lookahead Adder

Better Form of Lookahead



Carry-Lookahead Adder

Carry-Lookahead Adder -- High-Level View



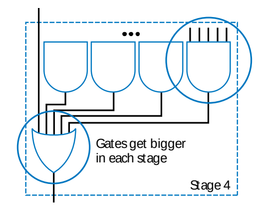
- Fast -- only 4 gate delays
  - Each stage has SPG block with 2 gate levels
  - Carry-lookahead logic quickly computes the carry from the propagate and generate bits using 2 gate levels inside
- Reasonable number of gates -- 4-bit adder has only 26 gates

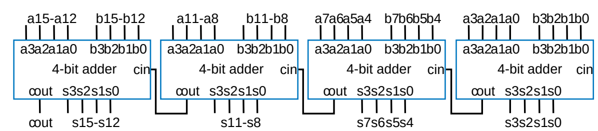
- 4-bit adder comparison (gate delays, gates)
  - Carry-ripple: (8, 20)
  - Two-level: (2, 500)
  - CLA: (4, 26)
    - Nice compromise

Carry-Lookahead Adder

Carry-Lookahead Adder – 32-bit?

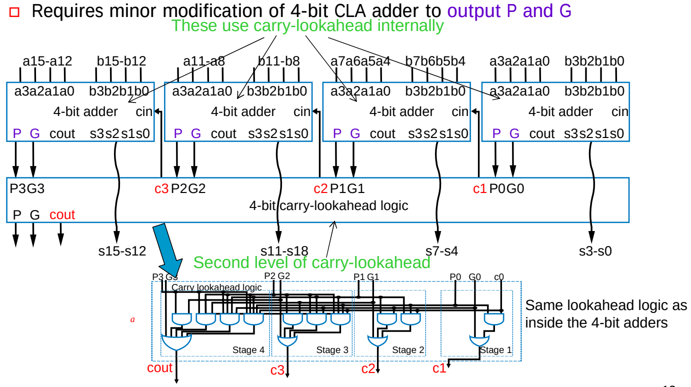
- Problem: Gates get bigger in each stage
  - 4th stage has 5-input gates
  - 32nd stage would have 33-input gates
    - Too many inputs for one gate
    - Would require building from smaller gates, meaning more levels (slower), more gates (bigger)
- One solution: Connect 4-bit CLA adders in ripple manner
  - But slow (4 + 4 + 4 + 4 gate delays)



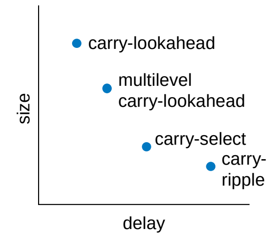


Carry-Lookahead Adder
Hierarchical Carry-Lookahead Adders

- Better solution -- Rather than rippling the carries, just *repeat* the carry-lookahead concept
  - Requires minor modification of 4-bit CLA adder to output P and G



Optimizations and Tradeoffs
Adder Tradeoffs



- Designer picks the adder that satisfies particular delay and size requirements
  - May use different adder types in different parts of same design
    - Faster adders on critical path, smaller adders on non-critical path