EECS 427
Lecture 9: Fast Adders

Reading: 11.3.2 – 11.3.3

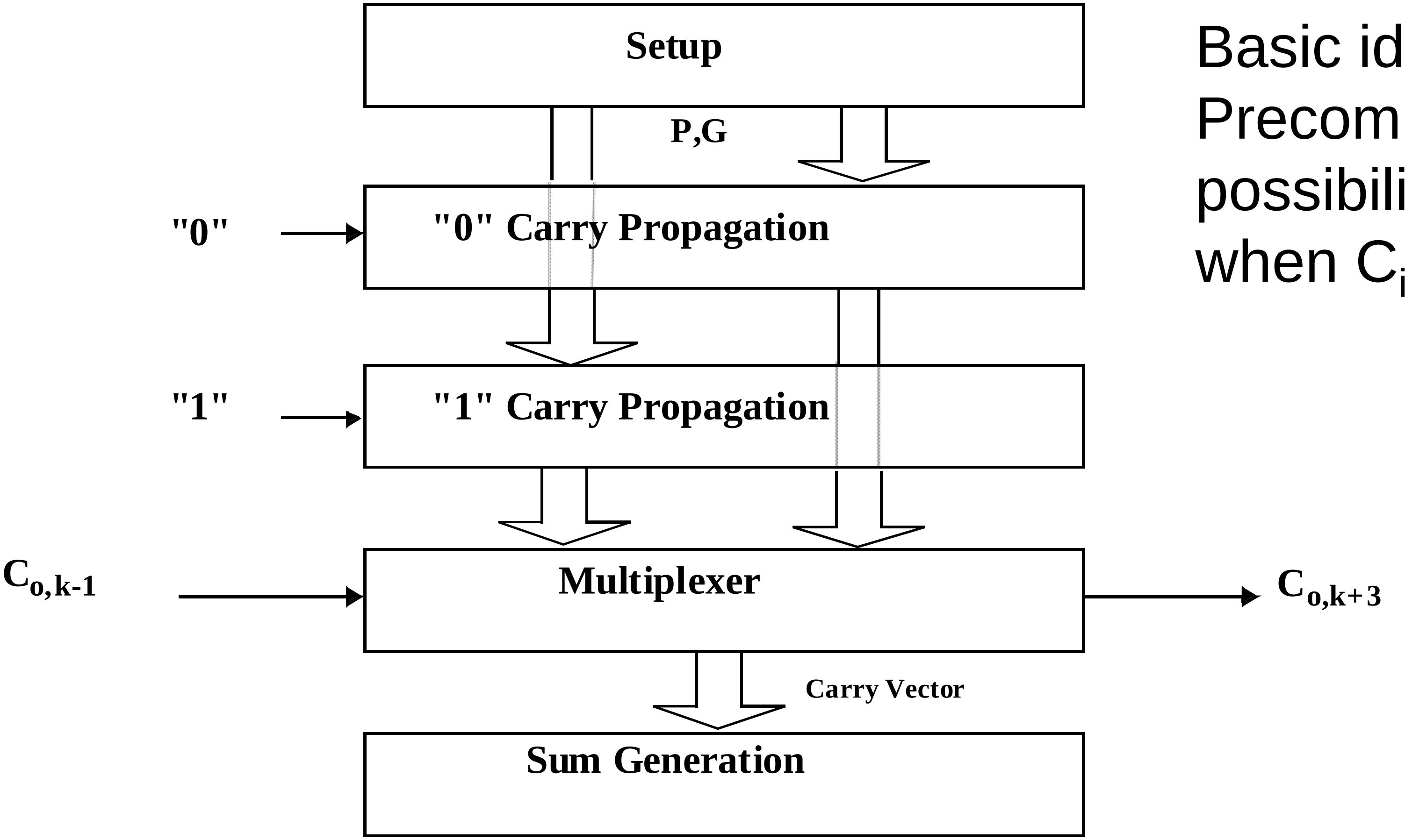
Last Time

- Adder intro
  - Ripple carry adder is simplest design but slow (delay grows linearly with # of bits)
  - Key point: Inputs (A and B) are available at time zero, carry in (deep in carry chain) is not
  - There are interesting circuit-level techniques to speed things up
  - Carry bypass (carry skip) adder

More Adder Topologies

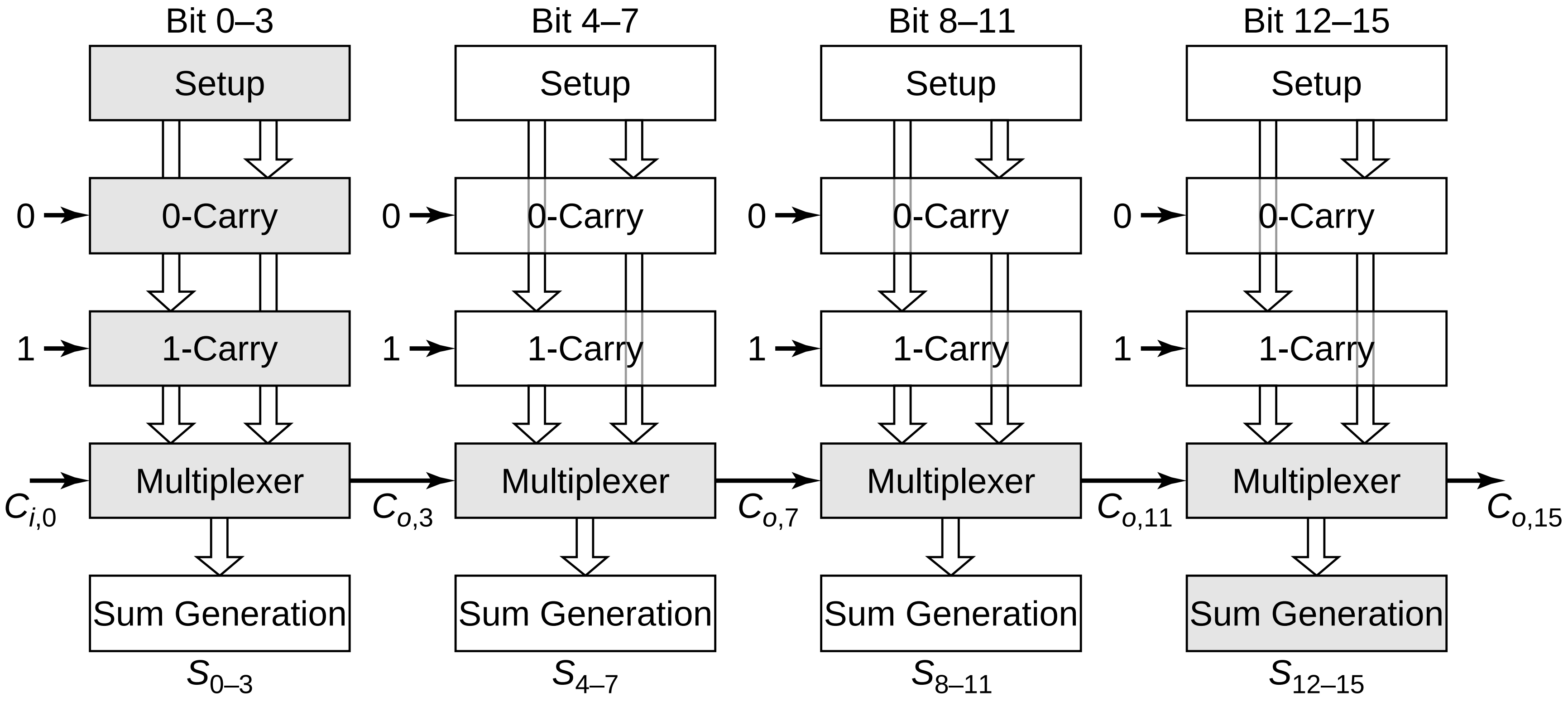
- Carry Select
- Carry lookahead
- Logarithmic lookahead
  - Tree
  - Brent-Kung
  - Kogge-Stone
- CAD3 Due Friday
- HW3 due next Tuesday in discussion
  - Initial project proposal, only a brief description required

Carry-Select Adder

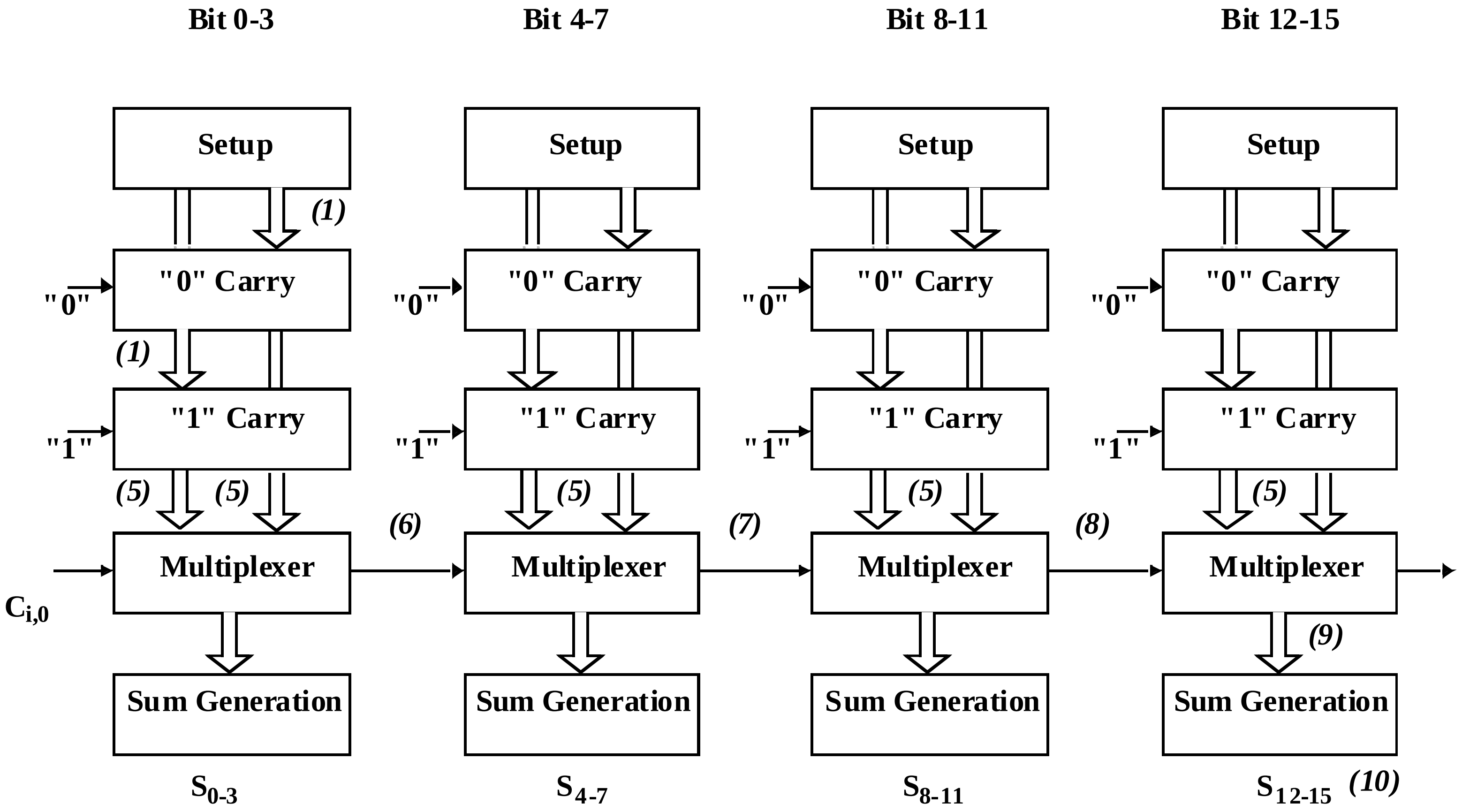


Basic idea: Precompute both possibilities, choose when C_{in} available

Carry Select Adder: Critical Path

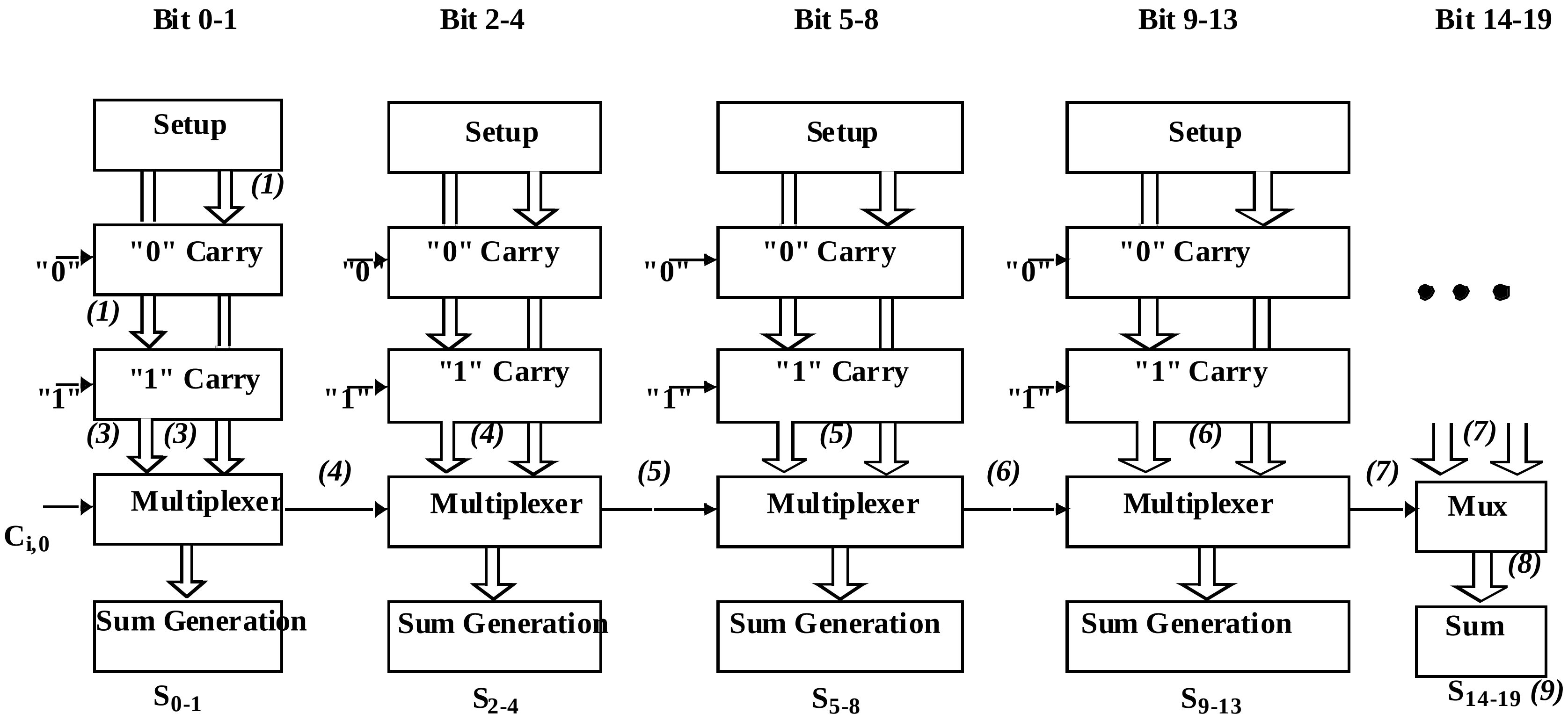


Linear Carry Select

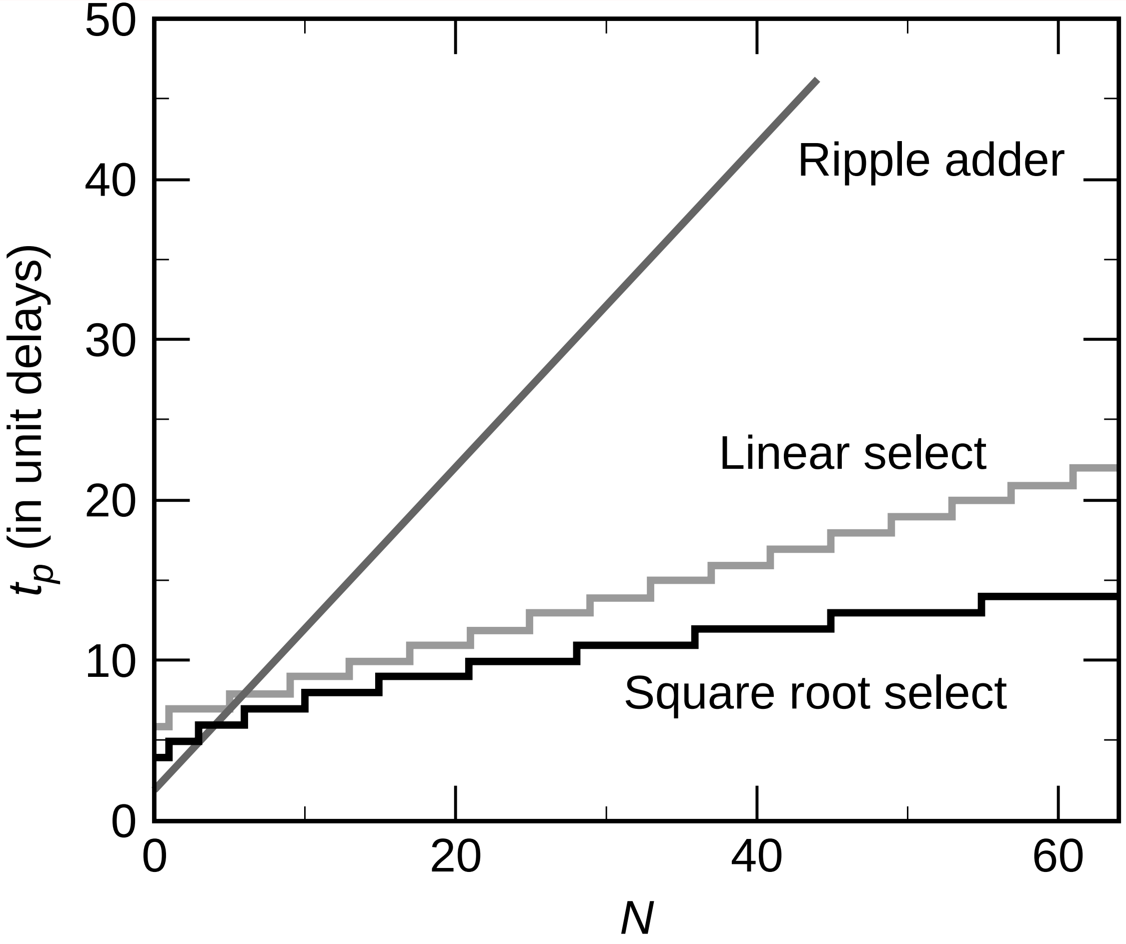


$$t_{add} = t_{setup} + M{*}t_{carry} + (N/M)t_{mux} + t_{sum}$$

Square Root Carry Select



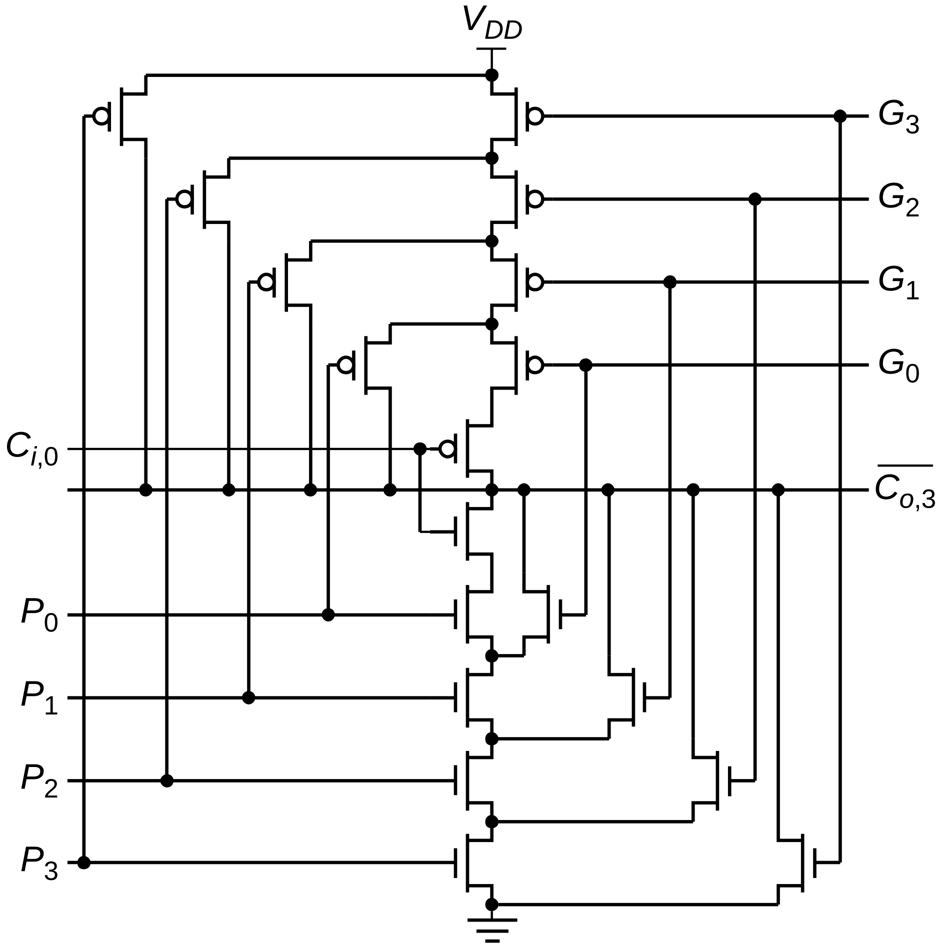
Adder Delays - Comparison



Carry lookahead

- It all comes down to computing carry out faster
- The obvious expensive solution:
  - Carry out of bit 4 depends on all inputs to bits 1-4 so, treat it as a single Boolean expression and make a circuit, possibly one large complex gate.
  - Simplifying down to P and G helps a lot here.
- Even so 4 bits of lookahead = 9 input function
  - $G_3 + P_3 (G_2 + P_2 (G_1 + P_1 (G_0 + P_0 C_{in})))$
- Called Carry lookahead
- Different amounts of lookahead possible
- Just makes blocks of bits act like single bit
- Can be combined with other designs

- Example of a single cmos gate implementation of carry lookahead of 4 bits.
- Note that this is a somewhat flawed example that would only work when P is defined as A or B rather than A xor B



Log Lookahead

- The idea with lookahead is to treat a block of bits like a single bit.
- What about the P,G of a block?

- Take 2 bits
  - $P_b=P_1P_0$
  - $G_b=G_1+P_1G_0$

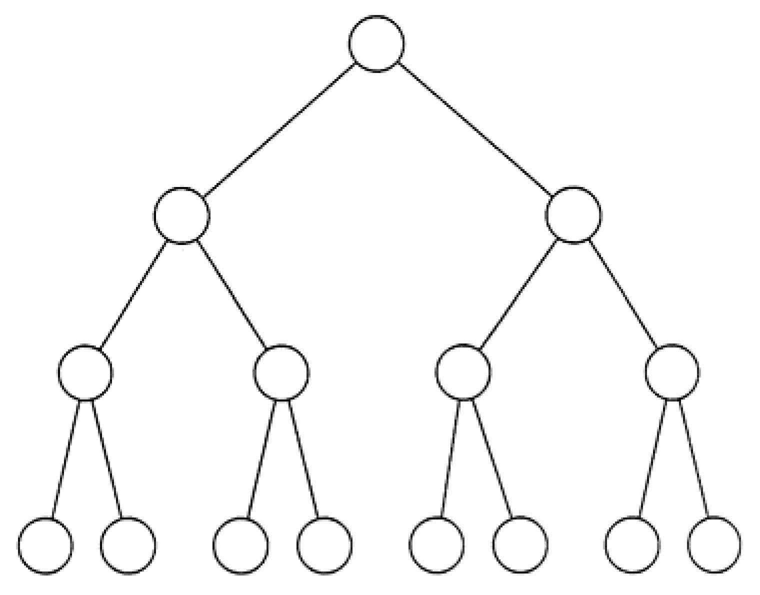
- With P, G we can compute C_{out} and go past the block more quickly.

- But now we could double our block size by building block of blocks and so on.

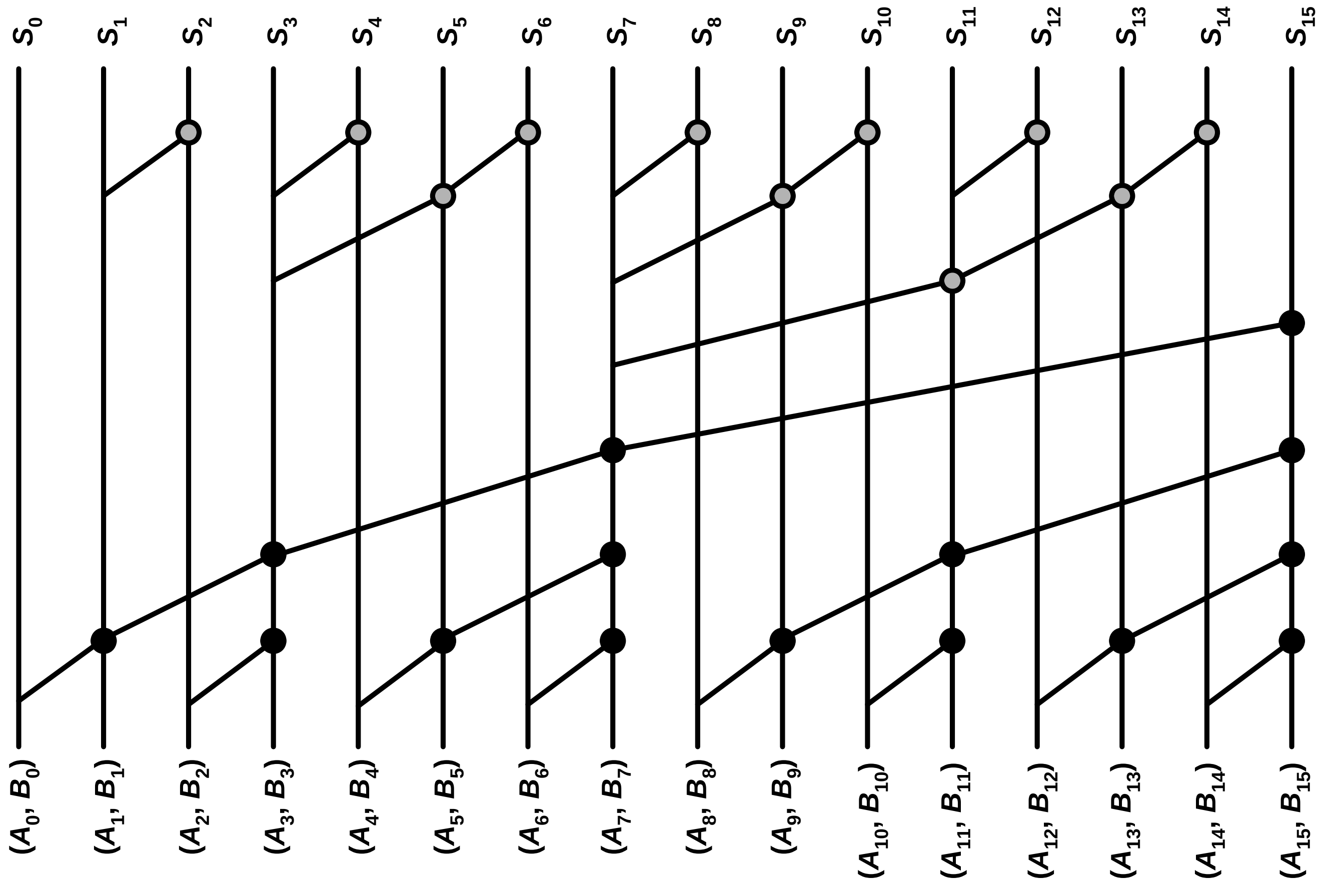
- Binary (or any radix really if you chunk more than 2 at a time) tree
- Lowest level computes local P,G
- Next level up computes P,G of 2 bit blocks
- Next level P,G of 4 bit blocks, etc.

Binary Tree

- P,G flow up, carry flows back down
- N-1 internal nodes for n bit adder
- $C_0 = C_{in}$, $C_1 = G_0 + P_0 C_{in}$



Brent-Kung

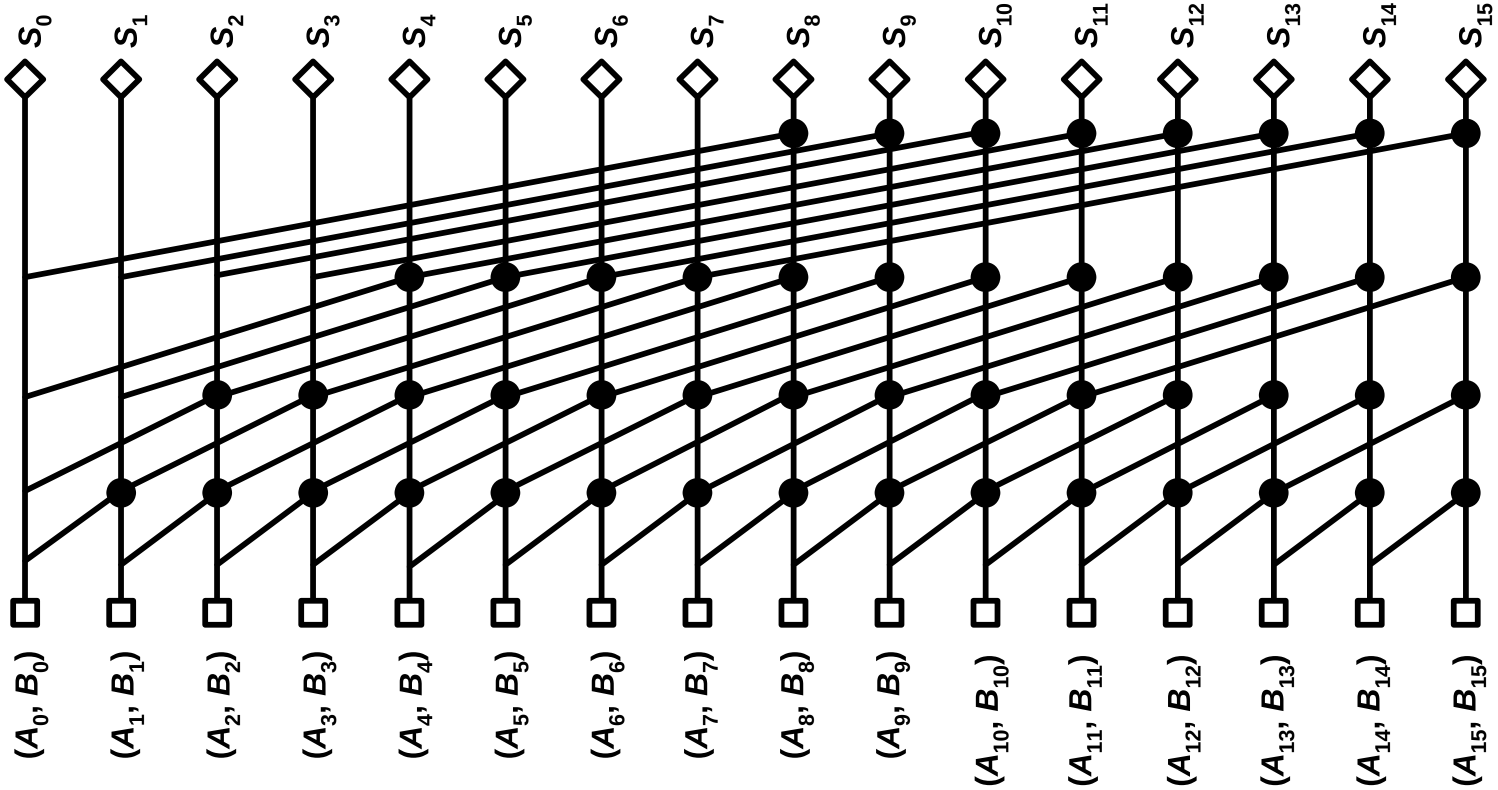


S_0
S_1
S_2
S_3
S_4
S_5
S_6
S_7
S_8
S_9
S_{10}
S_{11}
S_{12}
S_{13}
S_{14}
S_{15}
(A_0, B_0)
(A_1, B_1)
(A_2, B_2)
(A_3, B_3)
(A_4, B_4)
(A_5, B_5)
(A_6, B_6)
(A_7, B_7)
(A_8, B_8)
(A_9, B_9)
(A_{10}, B_{10})
(A_{11}, B_{11})
(A_{12}, B_{12})
(A_{13}, B_{13})
(A_{14}, B_{14})
(A_{15}, B_{15})

Brent-Kung

- Not really any different than the binary tree except that data flow has been visualized spatially flowing forward
- Dots represent computing P and G of block from children
- Note the irregular loading and wire lengths

Kogge-Stone


S_0
S_1
S_2
S_3
S_4
S_5
S_6
S_7
S_8
S_9
S_{10}
S_{11}
S_{12}
S_{13}
S_{14}
S_{15}
(A_0, B_0)
(A_1, B_1)
(A_2, B_2)
(A_3, B_3)
(A_4, B_4)
(A_5, B_5)
(A_6, B_6)
(A_7, B_7)
(A_8, B_8)
(A_9, B_9)
(A_{10}, B_{10})
(A_{11}, B_{11})
(A_{12}, B_{12})
(A_{13}, B_{13})
(A_{14}, B_{14})
(A_{15}, B_{15})

Kogge-Stone

- Eliminate “downward” flow by giving each bit its own “tree” (but share parts as possible)
- To even out the load and make it more regular, build trees top-down (0-3 built from 0, 1-2 instead of 0-1, 2
- Huge
- Regular structure
- Even fewer gates deep than Brent-Kung
- Gates have higher fan-out

Summary

- Many topologies for adders
  - Variants on carry lookahead/log lookahead are dominant today
- For 16-bit addition, complex techniques such as lookahead do not offer much benefit
  - Carry select and carry bypass yield good performance in this case

- Thought : revisit carry-select from lookahead perpective. Cin = 0 is G Cin = 1 is P. Could see carry select like lookahead with fewer long wires.