COMP 103

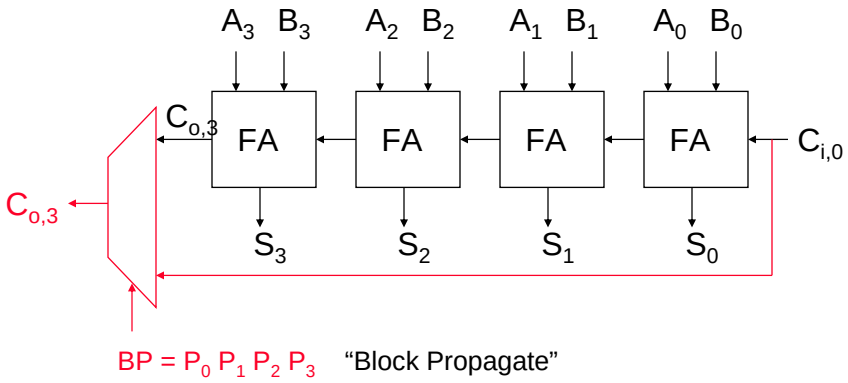
Lecture 14

Adder Design Continued

Reading:
Chapter 11, 577- 586
(skip dynamic implementations)

[All lecture notes are adapted from Mary Jane Irwin, Penn State, which were adapted from Rabaey's *Digital Integrated Circuits*, ©2002, J. Rabaey et al.]

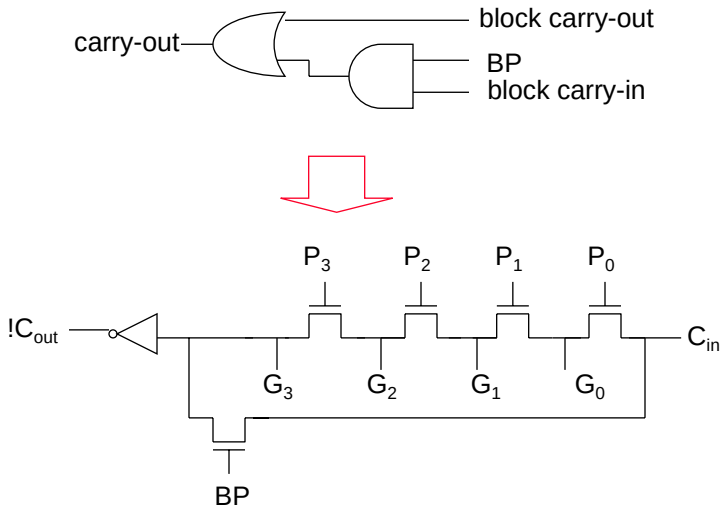
Carry-Skip (Carry-Bypass) Adder



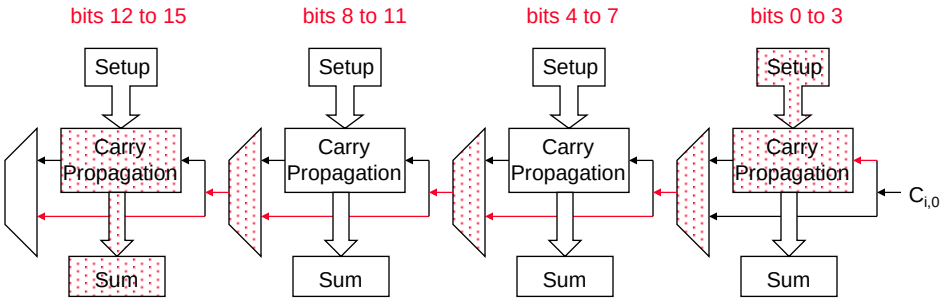
$BP = P_0 P_1 P_2 P_3$ "Block Propagate"

If (P_0 & P_1 & P_2 & P_3 = 1) then $C_{o,3} = C_{i,0}$ otherwise the block itself kills or generates the carry internally

Carry-Skip Chain Implementation



4-bit Block Carry-Skip Adder



Worst-case delay → carry from bit 0 to bit 15 = carry generated in bit 0, ripples through bits 1, 2, and 3, skips the middle two groups (B is the group size in bits), ripples in the last group from bit 12 to bit 15

$$T_{add} = t_{setup} + B\ t_{carry} + ((N/B) - 1)\ t_{skip} + B\ t_{carry} + t_{sum}$$

Optimal Block Size and Time

- Assuming one stage of ripple (t_{carry}) has the same delay as one skip logic stage (t_{skip}) and both are 1

$$T_{CSkA} = 1 + B + (N/B-1) + B + 1$$

t_{setup} — ripple in block 0 — skips — ripple in last block — t_{sum}

$$= 2B + N/B + 1$$

- So the optimal block size, B, is

$$dT_{CSkA}/dB = 0 \Rightarrow \sqrt{(N/2)} = B^{opt}$$

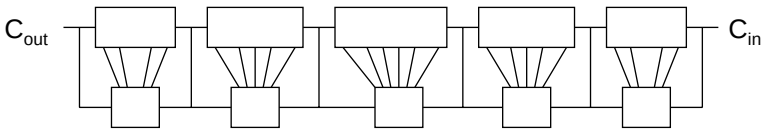
- And the optimal time is

$$\text{Optimal } T_{CSkA} = 2(\sqrt{(2N)}) + 1$$

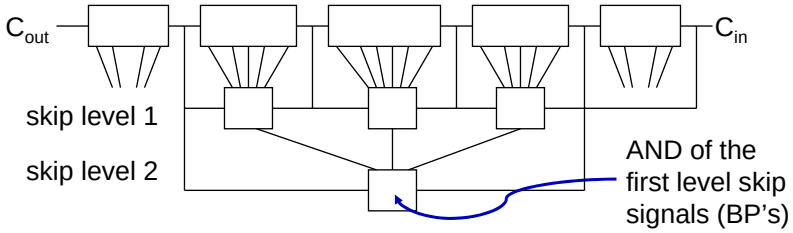
Example: N =32, B=? , T=?

Carry-Skip Adder Extensions

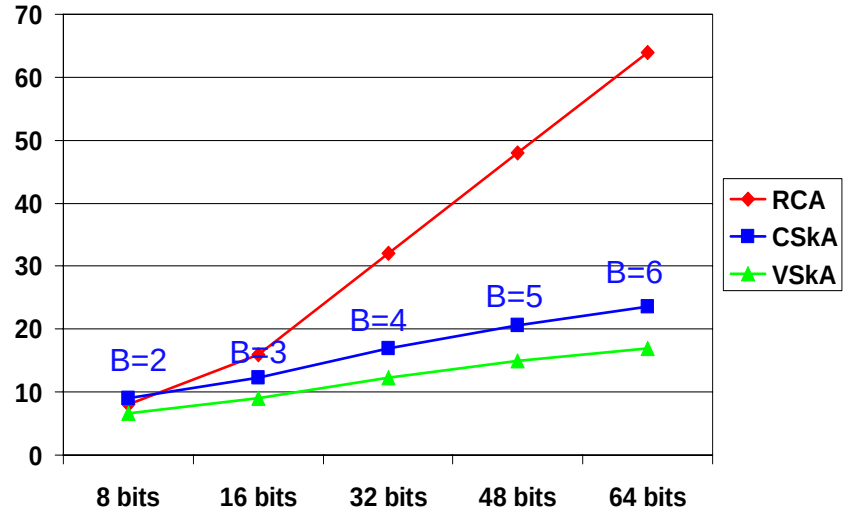
- Variable block sizes
  - A carry that is generated in, or absorbed by, one of the inner blocks travels a shorter distance through the skip blocks, so can have bigger blocks for the inner carries without increasing the overall delay



- Multiple levels of skip logic

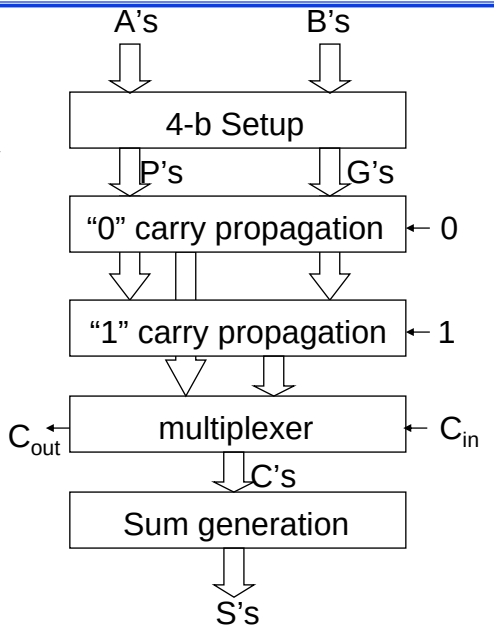


Carry-Skip Adder Comparisons

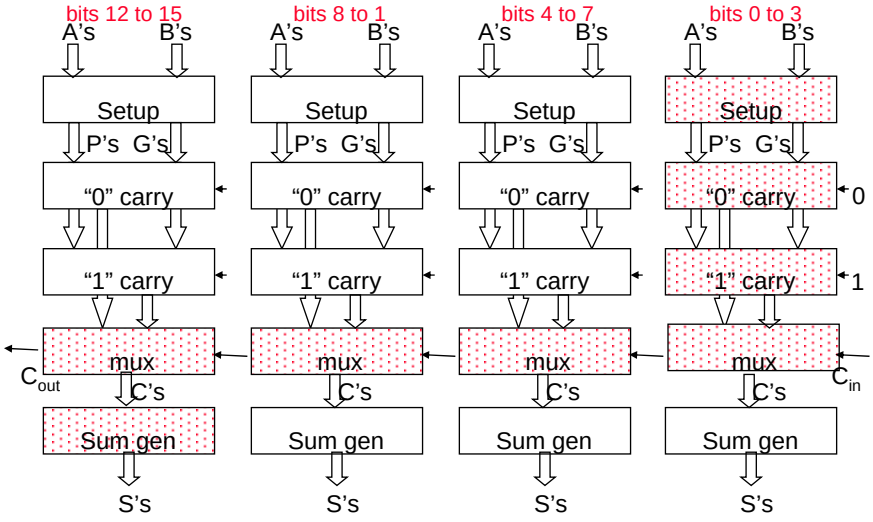


Carry Select Adder

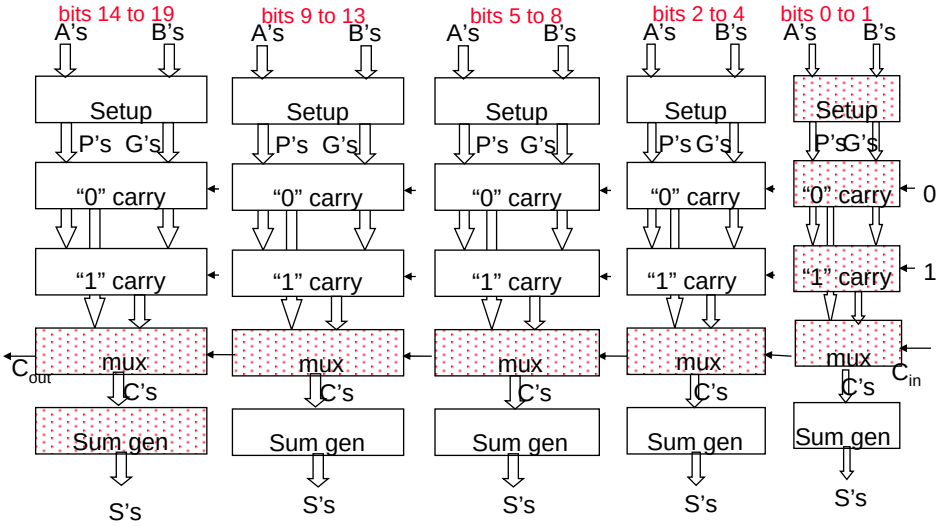
- Precompute the carry out of each block for both carry_in = 0 and carry_in = 1 (can be done for all blocks in parallel) and then select the correct one



Carry Select Adder: Critical Path

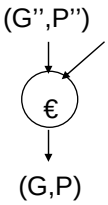


Square Root Carry Select Adder



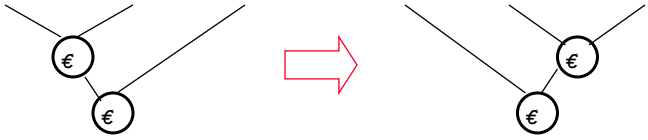
Parallel Prefix Adders (PPAs)

- Define carry operator € on (G,P) signal pairs



where
G = G'' ∨ P''G'
P = P''P'

  - € is associative, i.e.,
    [(g''',p''') € (g'',p'')] € (g',p') = (g''',p''') € [(g'',p'') € (g',p')]

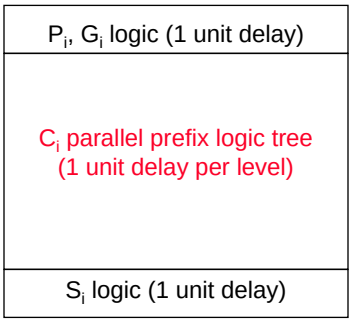


PPA General Structure

- Given P and G terms for each bit position, computing all the carries is equal to finding *all* the prefixes in parallel

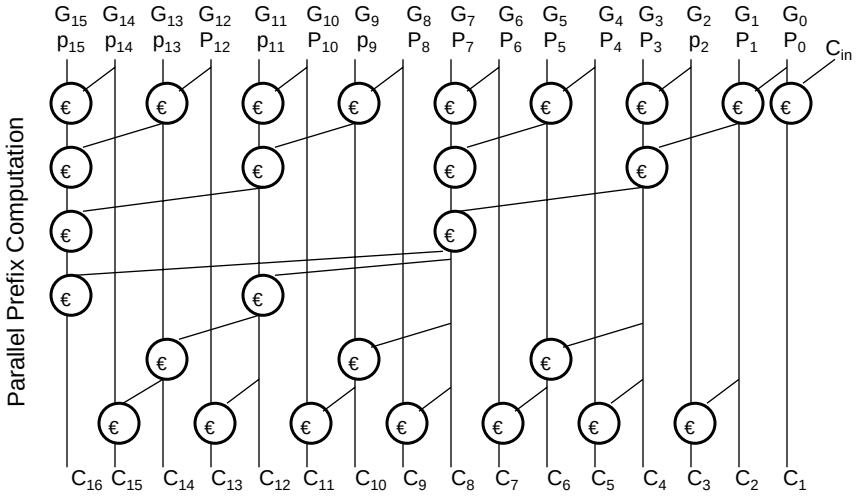
$$(G_0,P_0) \text{ € } (G_1,P_1) \text{ € } (G_2,P_2) \text{ € } \ldots \text{ € } (G_{N-2},P_{N-2}) \text{ € } (G_{N-1},P_{N-1})$$

- Since € is associative, we can group them in any order
  - but note that it is not commutative (can't swap the order!)

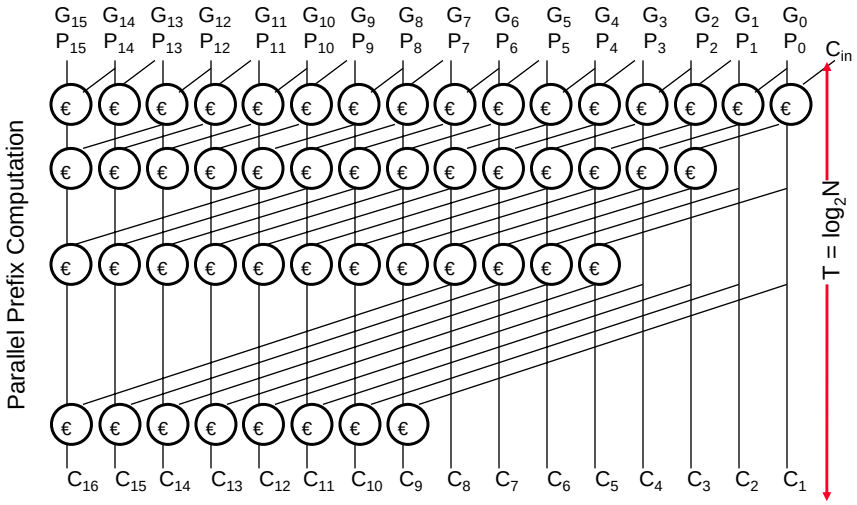


- Measures to consider
  - number of € cells
  - tree cell depth (time)
  - tree cell area
  - cell fan-in and fan-out
  - max wiring length
  - wiring congestion
  - delay path variation (glitching, power implications)

Brent-Kung PPA

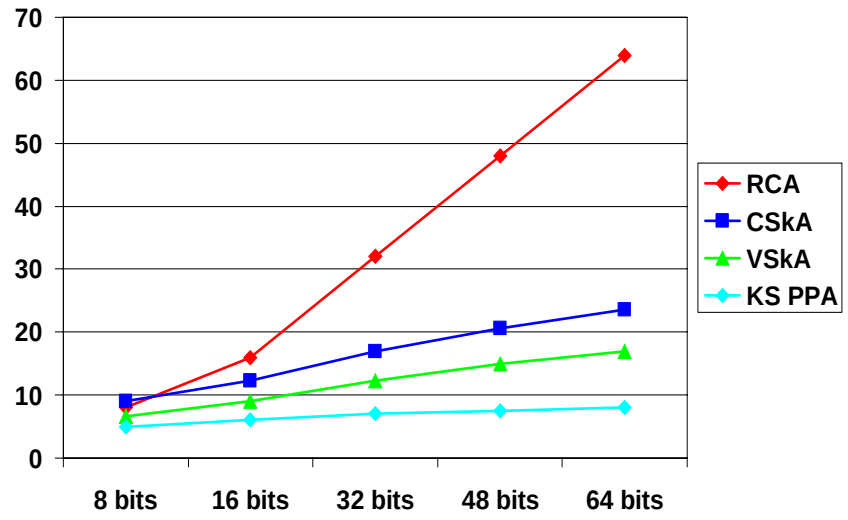


Kogge-Stone PPF Adder

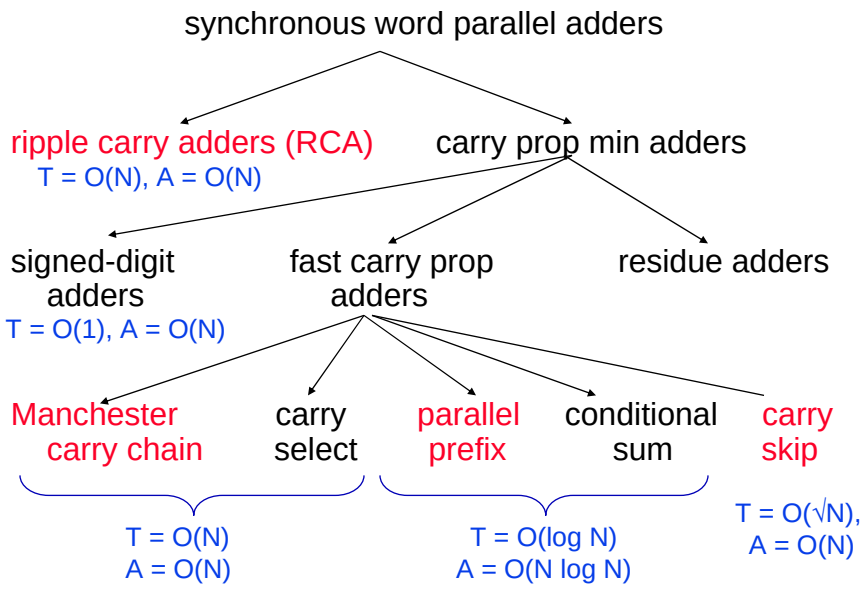


$$T_{add} = t_{setup} + \log_2 N\, t_{€} + t_{sum}$$

More Adder Comparisons



Binary Adder Landscape



Project: Teams of 2-3 people

- **Circuit design:** design and implement the switch level circuit and follow up with SPICE simulation. Possible circuits: adders, multipliers, VLIW instruction decoder, a DSP datapath, conversion from rectangular to polar coordinates, pipelining issues between synchronous and asynchronous domains, low-power comparators, on-chip level converters
- **Memories:** a register file design, content-addressable memories
- **Interconnect issues:**
  - Look into logical effort and see how it applies to interconnect. Deriving equations and checking the results against SPICE simulation
  - Bus encoding to minimize coupling
  - Near speed of light signaling (inductance effects)
- **Logical effort** related software effort: write a C/C++/Java program that evaluates the delay of a circuit based on logical effort, or that would allow you to size the gates.

More project ideas

IP-Blocks:

- Encryption chip for the Advanced Encryption Standard (Rijndael algorithm) with self-test upon power-up
- Public key encryption using Montgomery or Galois Field multiplier
- IP Packet Forwarding Engine (with possibly with Encryption / Decryption)
- USB (Universal Serial Bus) Function Controller for a Microcontroller (e.g. ARM, MIPS, etc.)
- PCI Controller for a Microcontroller
- Ethernet Controller for packet transmission/reception and encapsulation / de capsulation
- Viterbi decoder
- Game processors with rendering acceleration
- Floating point add / subtract / multiplier co-processor
- MPEG compression / decompression
- DSP core with multiple MAC units and VLIW instructions
- JPEG encoder / decoder
- High speed arithmetic coprocessor using advanced logic family such as SR-Domino or CPL logic for the data path design