Lecture 4

Adders

Computer Systems Laboratory
Stanford University
horowitz@stanford.edu

Copyright © 2006 Mark Horowitz
Some figures from High-Performance Microprocessor Design © IEEE

Overview

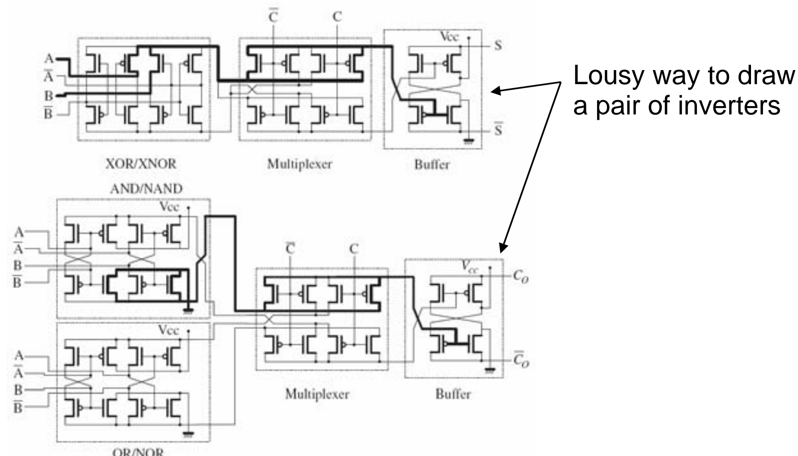
- Readings
- Today's topics
  - Fast adders generally use a tree structure for parallelism
  - We will cover basic tree terminology and structures
  - Look at a few example adder architectures
  - Examples will spill into next lecture as well

Adders

- Task of an adder is conceptually simple
  - Sum[n:0]=A[n:0]+B[n:0]+C_0
  - Subtractors also very simple: -B = ~B+1, so invert B and set C_0=1
- Per bit formulas
  - $Sum_i = A_i$ XOR B_i XOR C_i
  - $Cout_i = C_{i+1}$ = majority(A_i,B_i,C_i)
- Fundamental problem is calculating the carry to the nth bit
  - All carry terms are dependent on all previous terms
  - So LSB input has a fanout of n
    - And an absolute minimum of $\log_4 n$ FO4 delays without any logic

Single-Bit Adders

- Adders are chock-full of XORs, which make them interesting
  - One of the few circuits where pass-gate logic is attractive
  - A complicated differential passgate logic (DPL) block from the text

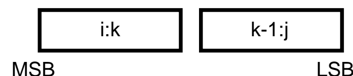


G and P and K, Oh My!

- Most fast adders "G"enerate, "P"ropagate, or "K"ill the carry
  - Usually only G and P are used; K only appears in some carry chains
- When does a bit Generate a carry out?
  - $G_i = A_i$ AND B_i
  - If G_i is true, then $Cout_i = C_{i+1}$ is forced to be true
- When does a bit Propagate a carry in to the carry out?
  - $P_i = A_i$ XOR B_i
  - If P_i is true, then $Cout_i$ $(=C_{i+1})$ follows C_i
  - Usually implemented as $P_i = A_i$ OR B_i
    - OR is cheaper/faster than an XOR
    - If you are doing logic, Cout is still equal to $G_i + P_i C_i$
    - Just beware that Sum_i != P_i XOR C_i

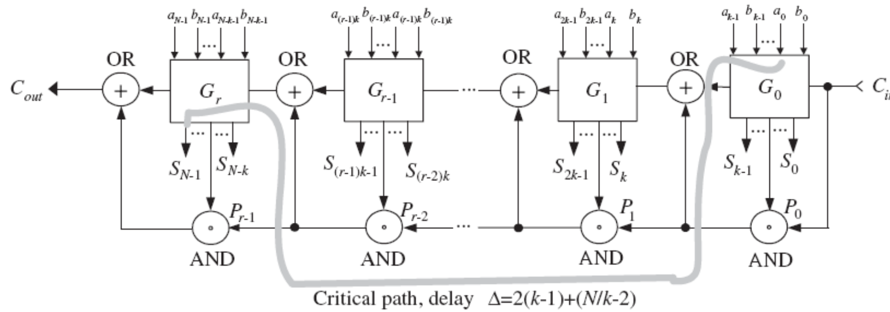
Using G and P

- We can combine G_i and P_i into larger blocks
  - Call these "group generate" and "group propagate" terms
- When does a group Generate a carry out? (*e.g.*, 4 bits)
  - $G_{3:0} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$
- When does a group Propagate a carry in to the carry out?
  - $P_{3:0} = P_3P_2P_1P_0$
- We can also combine groups of groups
  - $G_{i:j} = G_{i:k} + P_{i:k}G_{k-1:j}$
  - $P_{i:j} = P_{i:k}P_{k-1:j}$



Linear Adders Using P,G

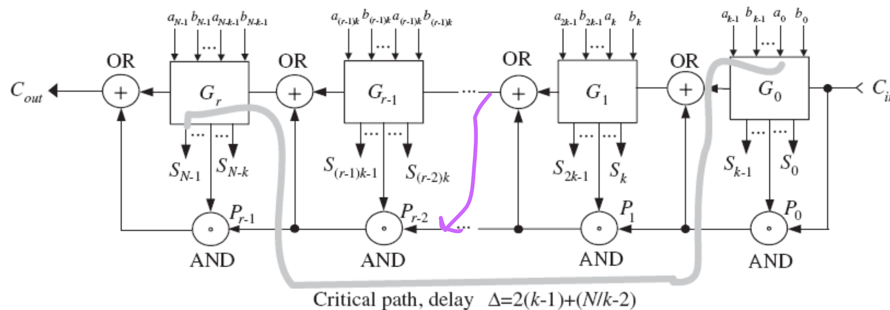
- Simple adders ripple the carry; faster ones bypass it
  - Better to try to work out the carry several bits at a time
- Best designs are around 11FO4 for 64b
  - Useful for small adders (16b) and moderate performance long adders
- Example of carry bypass from the text



Critical path, delay $\Delta=2(k-1)+(N/k-2)$

Figure Is Not Quite Right

- How does the drawn critical path go **down** from the OR gate?!
  - Fix: The OR gate output is the true input to the next group
- Subtle point: Each block spits out a G term for the OR
  - Not simply a $Cout_i$ term
  - Avoids a nasty critical path (11…1 + 00…0 + C_{in}; C_{in} goes 1→0)



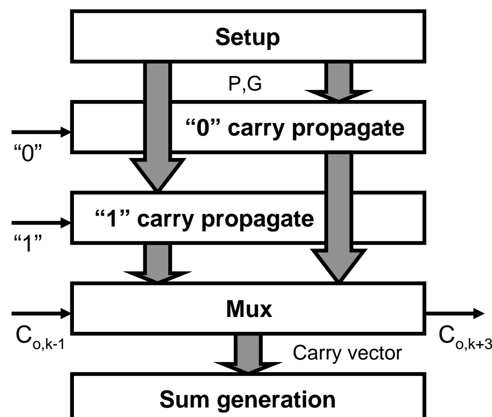
Critical path, delay $\Delta=2(k-1)+(N/k-2)$

Faster Carry Bypass (or Carry Skip) Adders

- We see the basic idea is to form multi-level carry chains
  - Break the bits into groups
  - Ripple the carry in each group, in parallel
  - Ripple the global carry across the groups
- How big should each group be? (N bits total, k bits per group)
  - If ripple time equals block skip time then delay = 2(N-1)+(N/k –2)
- Would groups of different sizes be faster? (yes)
  - Middle groups have longer to generate carry outs; should be larger
  - Early and late groups have ripples in critical path; should be shorter
  - Called "Variable Block Adders"

Carry Select Adders

- Why wait for the carry in? (If you can't find parallelism, invent it!)
  - Calculate answers for a group assuming $C_i = 1$ AND $C_i = 0$
  - Use two adders, and rely on the fact that transistors are cheap
  - Don't do this on the full adder (too expensive), just the MSBs



Many Papers on These Adders

- But no one builds them anymore
  - Or rather, nobody publishes papers on them (or gets PhDs on them)
- These are all clever improvements on adders
  - That tend to optimize transistors along with performance
  - Or are best for narrow-width operands (n=64 is slow)
- But scaling is pushing these adders to the wayside
  - We have very wide-word machines (media applications)
  - We have more transistors than we know what to do with
- Question: As power density questions increase…
  - … will these "simpler" adders make a comeback?

Logarithmic, or Tree, Adders

- Fundamental problem: to know C_i, we need C_{i-1}
  - So delay is linear with n, and this dominates for wide adders (n>16)
  - Can we lookahead across multiple levels to figure out carry? Yes.
  - Called "prefix computation" – turns delay into logarithmic with n
- Notation is always an issue; everybody does it differently
  - Here, $A_{i:j}$ means the signal "A" for group the ith to j^{th} position
  - P = propagate (A+B)
  - G = generate (AB)
  - C = CarryIn to this bit/Group position

Logic Stages For Logarithmic/Tree Adders

1. Compute single bit values

   $0<=i<n$ $G_i=A_iB_i$ $P_i=A_i+B_i$
2. Compute two-bit groups

   $0<=i<(n/2)$ $G_{2i+1:2i}=G_{2i+1}+G_{2i}P_{2i+1}$ $P_{2i+1:2i}=P_{2i+1}P_{2i}$
3. Compute four-bit groups

   $0<=i<(n/4)$ $G_{4i+3:4i}=G_{4i+3:4i+2}+G_{4i+1:4i}P_{4i+3:4i+2}$ $P_{4i+3:4i}=P_{4i+3:4i+2}P_{4i+1:4i}$
4. …Go down tree for G&P, then go back up for Cin…
5. Compute four-bit carries

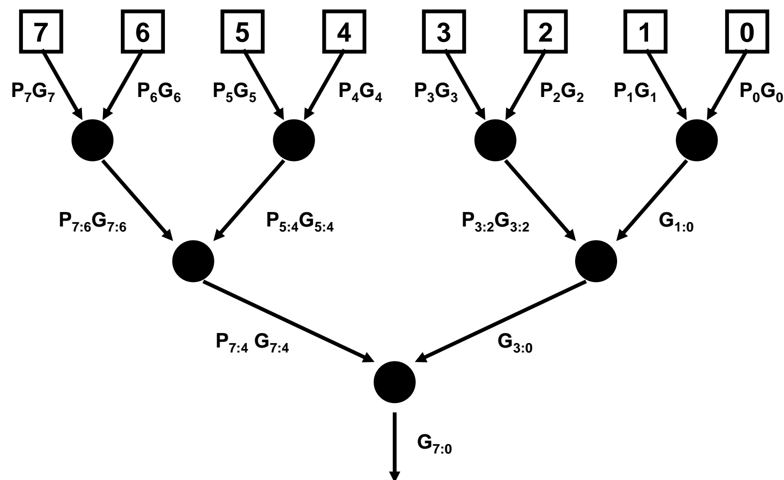
   $0<=i<(n/8)$ $C_{8i+7:8i+4} = G_{8i+3:8i} + C_{8i+7:8i}P_{8i+3:8i}$ $C_{8i+3:8i} = C_{8i+7:8i}$
6. Compute two-bit carries

   $0<=i<(n/4)$ $C_{4i+3:4i+2} = G_{4i+1:4i} + C_{4i+3:4i}P_{4i+1:4i}$ $C_{2i+1:2i} = C_{2i+3:2i}$
7. Compute single-bit carries

   $0<=i<(n/2)$ $C_{2i+1} = G_{2i} + C_{2i+1:2i}P_{2i}$ $C_{2i} = C_{2i+1:2i}$

An Eight-bit Example

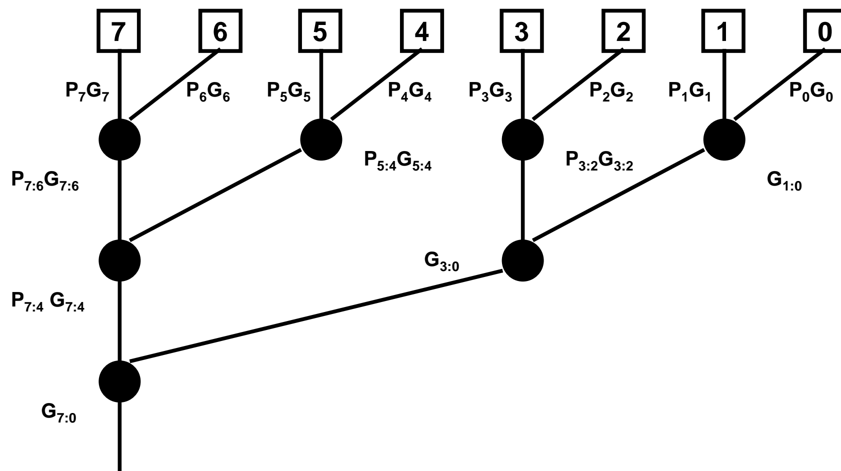
- “Lines and dots” notation shows the tree structure clearly



- Takes $\log_2 n$ time to get the final carry-out ($Cout_7=G_{7:0}$)

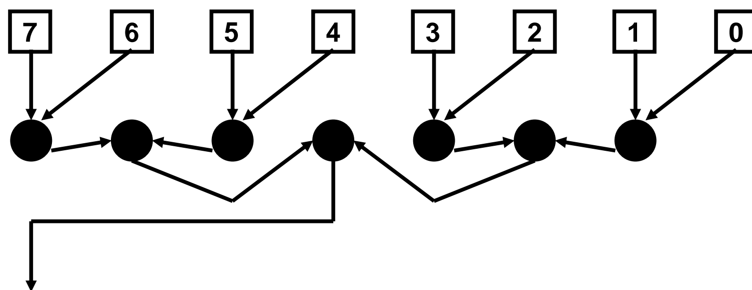
An Eight-Bit Example, Redrawn

- More common to line up the PG terms with their appropriate bits



Layout Of Our Example Tree Adder

- Logarithmic structures have somewhat ugly layout



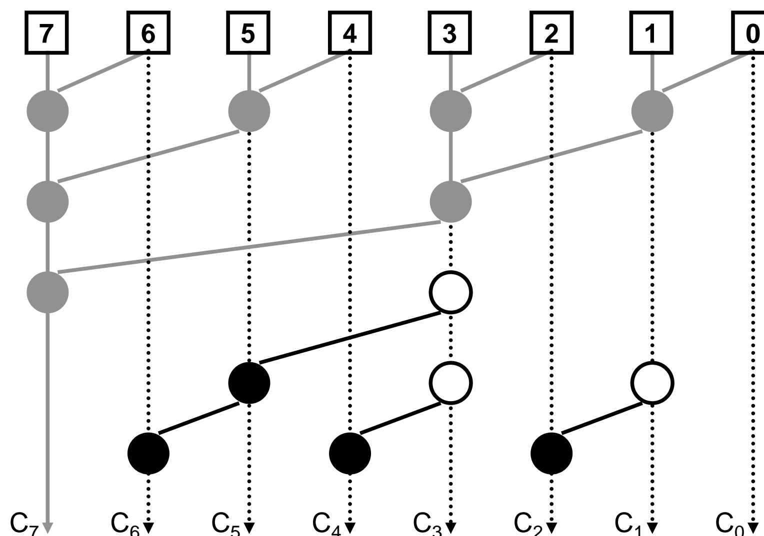
- Worst wire length grows as n increases (n=64? 128?)

That Was Half The Algorithm…

1. Compute single bit values
   0<=i<n $G_i=A_iB_i$ $P_i=A_i+B_i$
2. Compute two-bit groups
   0<=i<(n/2) $G_{2i+1:2i}=G_{2i+1}+G_{2i}P_{2i+1}$ $P_{2i+1:2i}=P_{2i+1}P_{2i}$
3. Compute four-bit groups
   0<=i<(n/4) $G_{4i+3:4i}=G_{4i+3:4i+2}+G_{4i+1:4i}P_{4i+3:4i+2}$ $P_{4i+3:4i}=P_{4i+3:4i+2}P_{4i+1:4i}$
4. …Go down tree for G&P, then go back up for Cin…
5. Compute four-bit carries
   0<=i<(n/8) $C_{8i+7:8i+4} = G_{8i+3:8i} + C_{8i+7:8i}P_{8i+3:8i}$ $C_{8i+3:8i} = C_{8i+7:8i}$
6. Compute two-bit carries
   0<=i<(n/4) $C_{4i+3:4i+2} = G_{4i+1:4i} + C_{4i+3:4i}P_{4i+1:4i}$ $C_{2i+1:2i} = C_{2i+3:2i}$
7. Compute single-bit carries
   0<=i<(n/2) $C_{2i+1} = G_{2i} + C_{2i+1:2i}P_{2i}$ $C_{2i} = C_{2i+1:2i}$

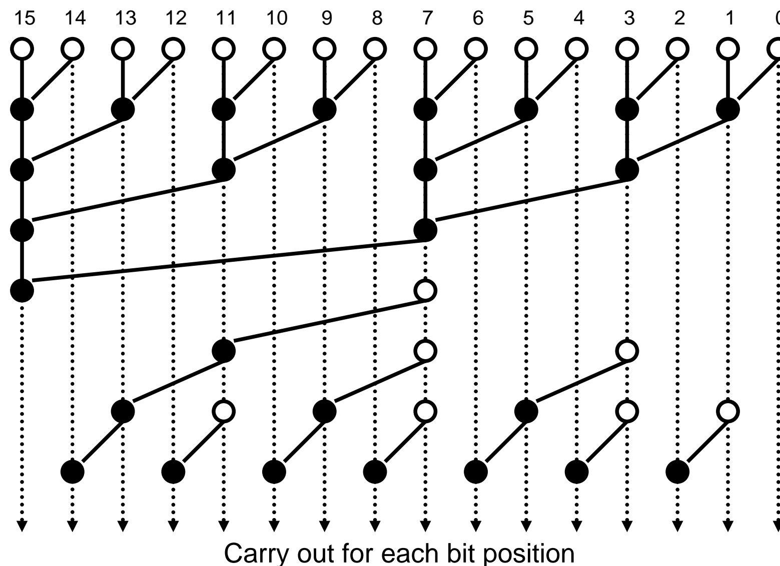
An Eight-Bit Example, Finished

- A Brent-Kung adder (1982): what's the critical path?



A 16b Brent-Kung Adder

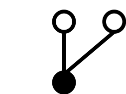
- Limit fanout to 2 (can collapse some nodes with higher FO)



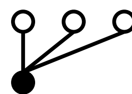
Carry out for each bit position

Many Kinds of Tree Adders

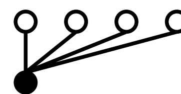
- We can vary some basic parameters
  - Radix, tree depth, wiring density, and fanout
- Radix: how many bits are combined in each Pgroup, Gg term?
  - Radix is generally < 4 (why not more?); prior example was 2
  - Radix-n can just compute the Pg and Gg terms directly



Radix-2 block

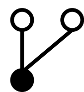


Radix-3 block

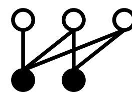


Radix-4 block

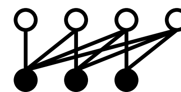
  - Or it can compute the intermediate P# and G# terms as well



Radix-2 block



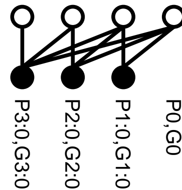
Radix-3 block



Radix-4 block

Building Multiple-Bit PG blocks

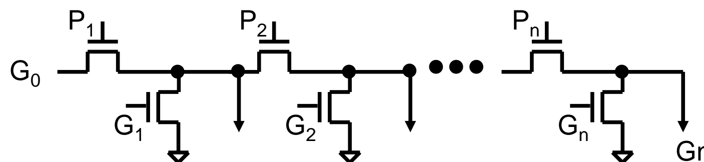
- Radix-4 Pg, Gg block that generates intermediate terms
  - Spits out "$P_{3:0}$" and "$G_{3:0}$" terminology
  - Also spits out P3, G3, P2, G2 and passes along P1, G1



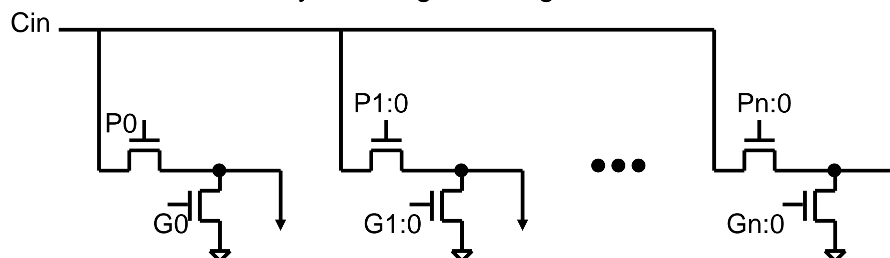
  - Allows for quick computation of the various carries, once we know C_{in}
  - How do we build this block?

Building Multiple-Bit PG blocks, con't

- Can we use dynamic logic to build fast blocks?
  - A Manchester carry chain can "gather" the multiple-bit G terms

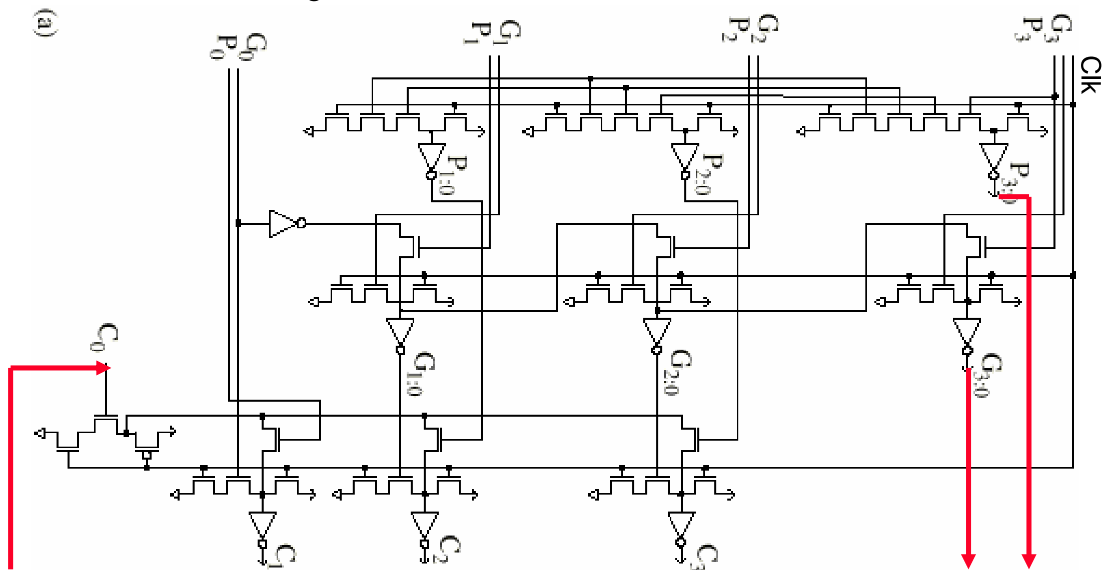


  - For C, since we already have Pgs and Ggs we can do better



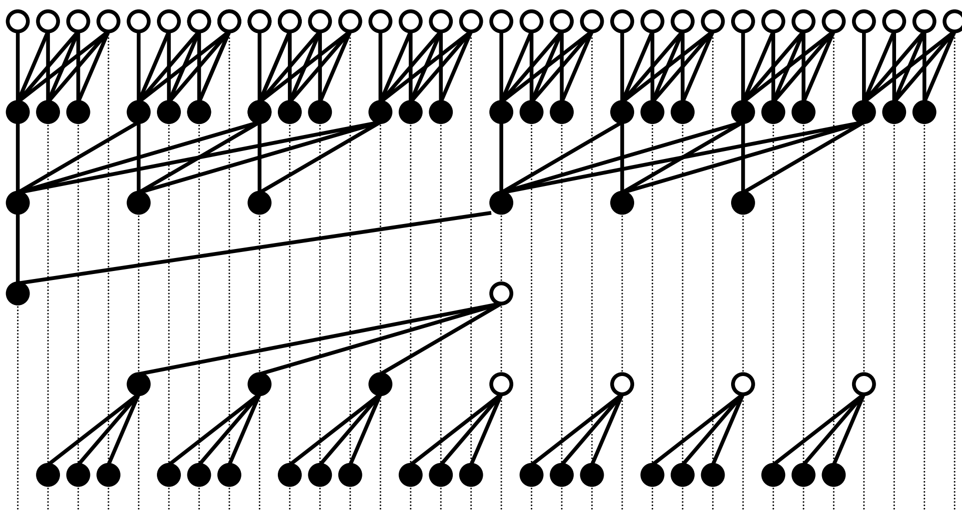
Dynamic Logic for 4-Bit PG Block

- Motorola design



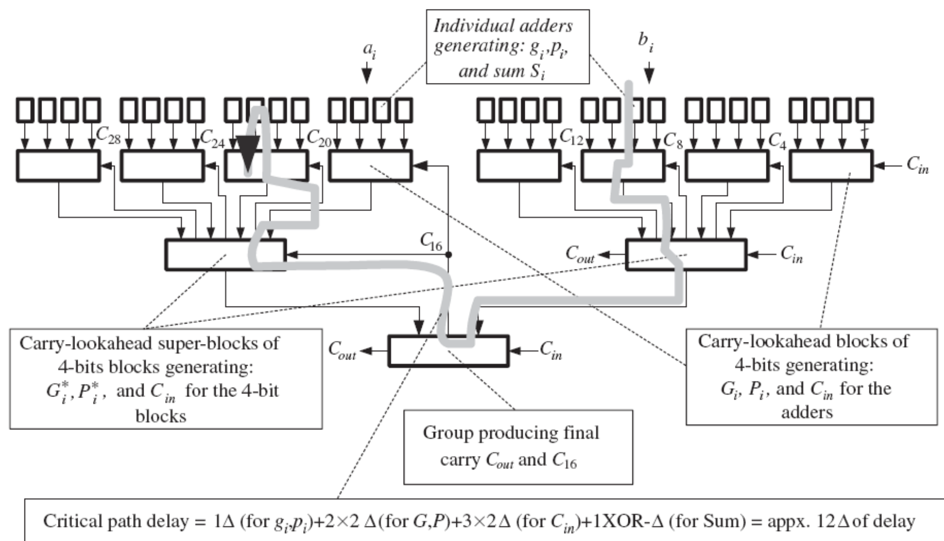
32b Mixed-Radix Brent-Kung Adder

- Vary radices at different tree levels because n may not be $(\text{radix})^k$



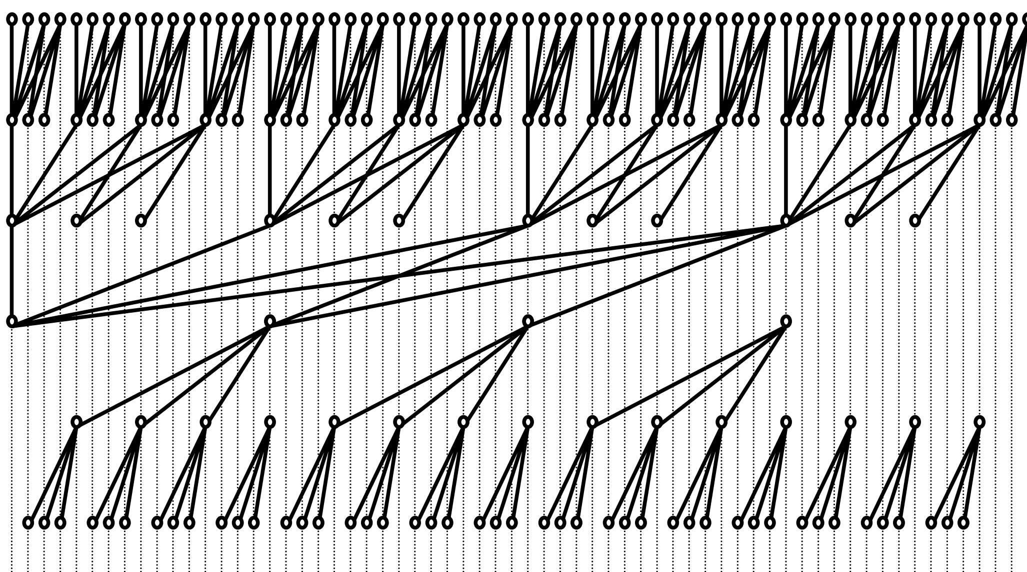
32b Mixed-Radix, Redrawn As Folded Tree

- PG goes "down" and Carry goes back "up"



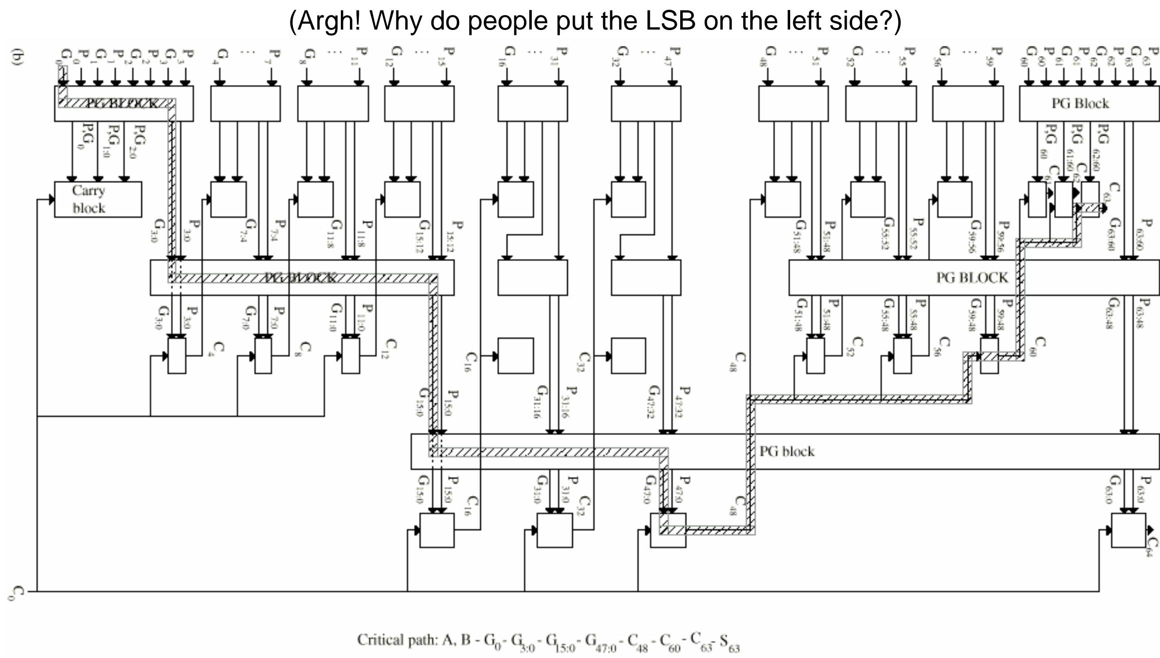
64b Radix-4 Brent-Kung Adder

- Takes longer to draw in Powerpoint than it does to design!



Radix 4 PG and Carry Trees

(Argh! Why do people put the LSB on the left side?)

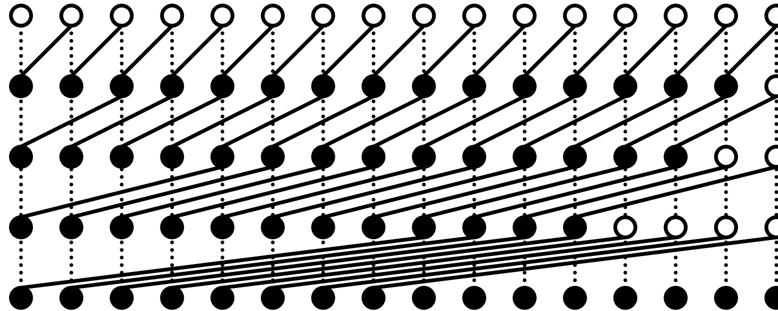


Tree Depth, Wiring Density, and Fanout

- Previous slides have all examined changing the adder radix
- We can also change the tree depth, wire density, and/or fanout
  - These usually get changed together; one affects the others
  - Density: How many wires criss-cross the tree?
  - Depth: How many stages of logic?
  - Fanout: How far is the reach of each stage?
- Reduce depth by chopping trees
  - Many adders use carry select at the final stage
    - Compute two results, and use a carry to select right result
  - Eliminate the carry tree altogether
    - Increasing wiring density or increasing fanout

A Very Dense 16b Tree

- Eliminate the carry out tree by computing a group for each bit
  - Kogge-Stone architecture (1973)



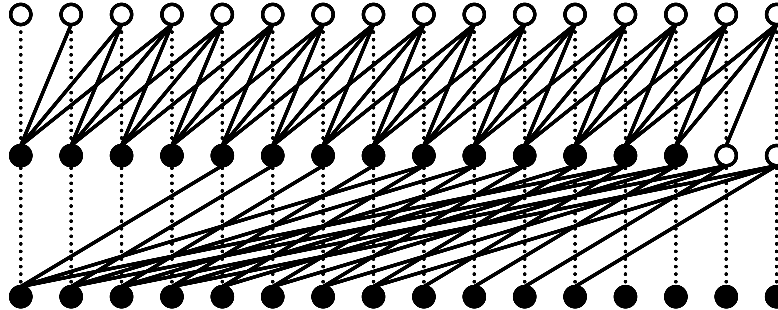
- Lots of wires, but minimizes the number of logic levels
- We can use this for a quick swag at the minimum delay

Minimum 64b Adder Delay

- Make a few assumptions
  - Output load is equal to the load on each input
  - Use static gates; very aggressive domino logic may change results
- Simple approximation
  - Need to compute $Sum_i = A_i$ XOR B_i XOR C_i
  - C_{in} (LSB) must fanout to all bits for a fanout of 64
  - Extra logic in chain raises effective fanout to about 128 → 3.5 FO4
- More complicated approximation
  - At each stage, P drives 3 gates, G drives 2; effective fanout ¼ 3.5
  - Total fanout = 1.5 (first NAND/NOR) * 3.5^6 * 1-ish (final mux)
  - 5.7 FO4, not really accounting for parasitic delay correctly

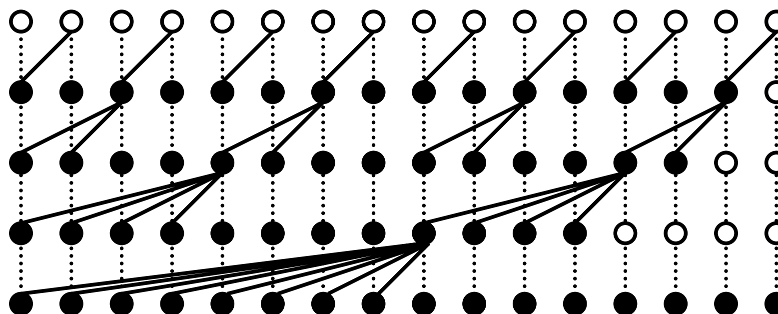
Higher Radix Still Possible

- Radix-4 Kogge-Stone tree
  - Trades off two layers of logic for lots and lots of wires
- Not a good idea in CMOS – it tends to increase stage efforts >> 4
  - Not bad, though, for domino – much lower logical effort



Reduce Depth With Fanout

- Can also reduce tree depth by increasing the stage fanouts
  - Sklansky (1960) called this a “divide-and-conquer” tree
  - Fanouts increase the further from the start you go



A Taxonomy

- Following Harris's 2003 paper
  - Assume 16b radix-2 adder families for this discussion
  - We can modify tree's depth, fanout, and wiring density
- What we've seen already
  - Brent-Kung: 7 logic levels, fanout of 2, one wiring track enough
  - Kogge-Stone: 4 logic levels, fanout of 2, eight wiring tracks
  - Sklansky: 4 logic levels, fanout of up to 9, one wiring track enough
- Formalism: Use a triplet (l,f,t) to represent the adders
  - Logic levels = $\log_2 N + l$ — Brent-Kung: (3,0,0)
  - Fanout = 2^f+1 — Kogge-Stone: (0,0,3)
  - Wiring tracks = 2^t — Sklansky: (0,3,0)

Points on a plane?

- All major adder architectures fall onto the same plane
  - Defined by $l+f+t=\log_2 N-1$

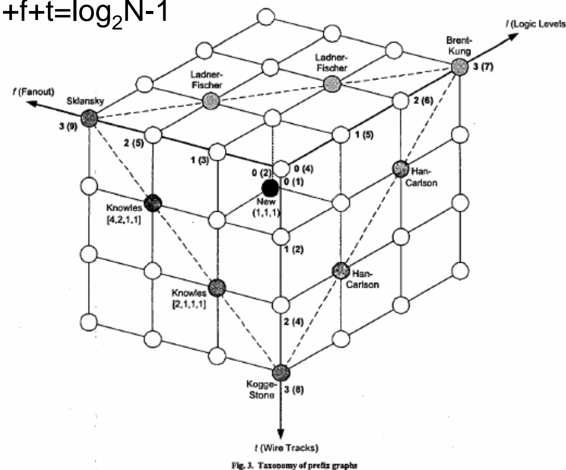
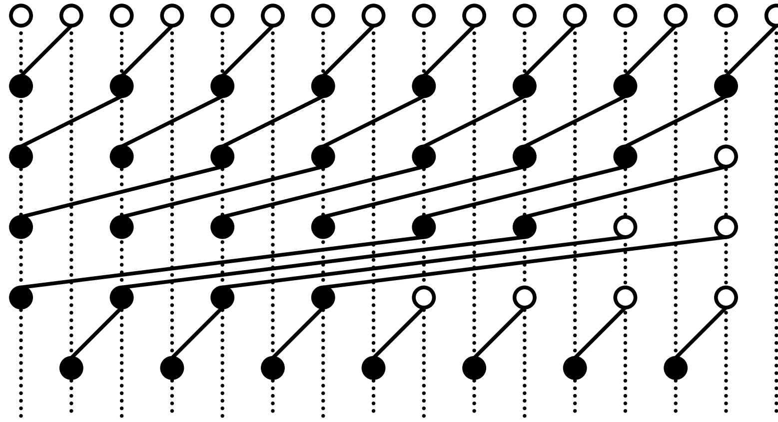


Fig. 3. Taxonomy of prefix graphs

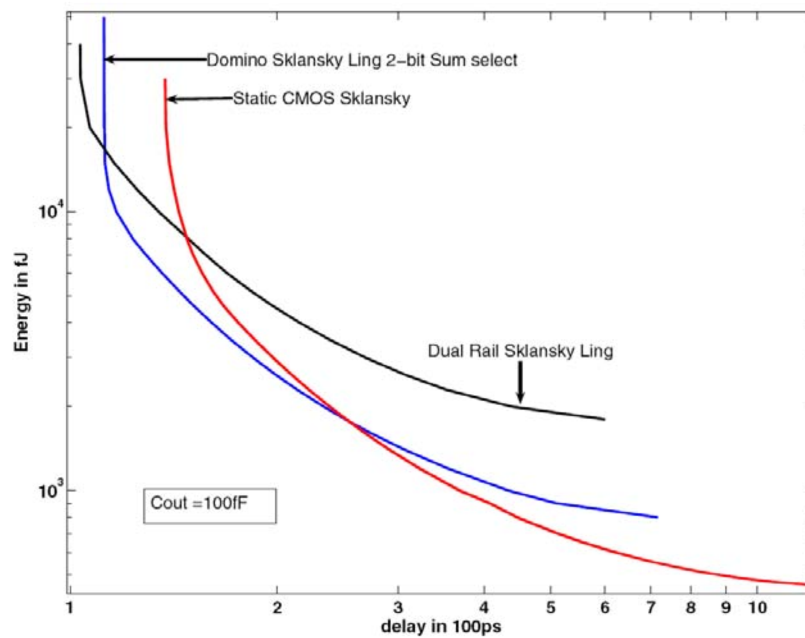
- Using this, we may expect a Han-Carlson adder to…
  - Trade off logic layers for some increased wiring

Han-Carlson Adder (1987)

- Think of this as a sparse Kogge-Stone
  - Called a sparse tree with sparseness of 2



Adder Tradeoffs



Other Sparse Trees

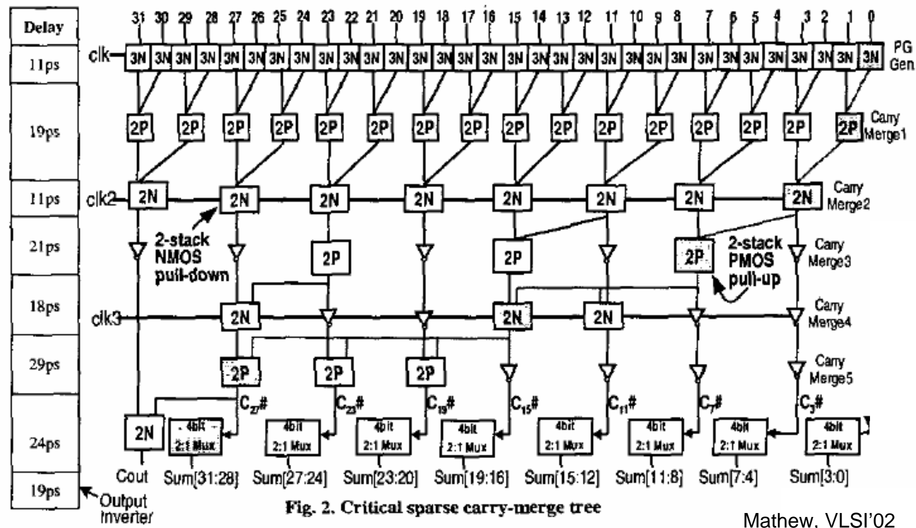


Fig. 2. Critical sparse carry-merge tree

Mathew, VLSI'02