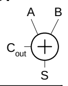
8. Design of Adders

- Last module:
  - Designing CMOS gate networks
  - Speeding up combinational gates
- This module
  - Adder circuits
  - Simple adders
  - Fast addition

Single-Bit Addition

Half Adder

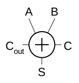
$$S = A \oplus B$$
$$C_{out} = A \cdot B$$



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

$$S = A \oplus B \oplus C$$
$$C_{out} = MAJ(A, B, C)$$

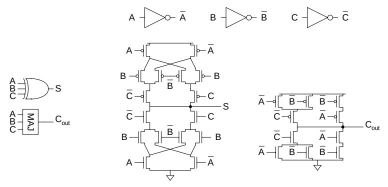


A	B	C	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full Adder Design I

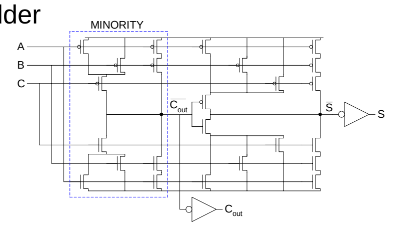
- Brute force implementation from equations

$$S = A \oplus B \oplus C$$
$$C_{out} = MAJ(A, B, C)$$



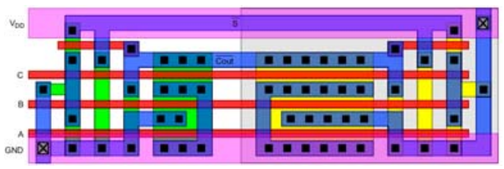
Full Adder Design II

- Factor S in terms of C_{out}
  S = ABC + (A + B + C)(~C_{out})
- Critical path is usually C to C_{out} in ripple adder



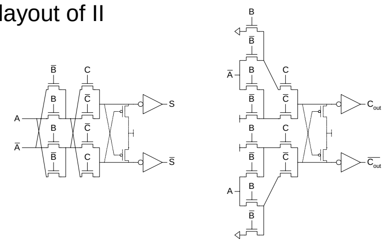
Layout

- Clever layout circumvents usual line of diffusion
  - Use wide transistors on critical path
  - Eliminate output inverters



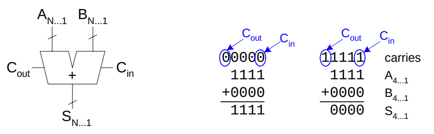
Full Adder Design III

- Complementary Pass Transistor Logic (CPL)
  - Slightly faster, but more area cf. the clever layout of II



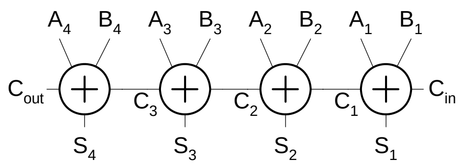
Carry Propagate Adders

- N-bit adder called CPA
  - Each sum bit depends on all previous carries
  - How do we compute all these carries quickly?



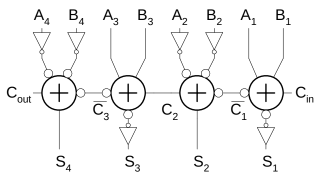
Ripple Carry Adder

- Simplest design: cascade full adders
  - Critical path goes from Cin to Cout
  - Design full adder to have fast carry delay



Inversions

- Critical path passes through majority gate
  - Built from minority + inverter
  - Eliminate inverter and use inverting full adder



PGK

- For a full adder, define what happens to carries
  - Generate: C_{out} = 1 independent of C
    - G = A • B
  - Propagate: C_{out} = C
    - P = A ⊕ B
  - Kill: C_{out} = 0 independent of C
    - K = ~A • ~B (i.e., ~K = A + B)

Generate / Propagate

- Equations often factored into G and P
- Generate and propagate for groups spanning i:j

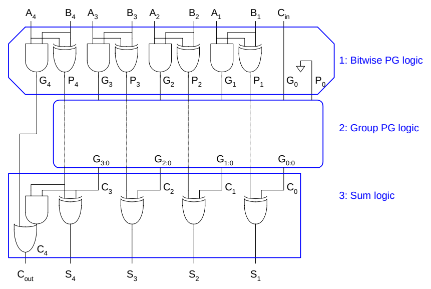
$$G_{i:j} = G_{i:k} + P_{i:k} \square G_{k-1:j}$$
$$P_{i:j} = P_{i:k} \square P_{k-1:j}$$

- Base case

$$G_{i:i} \equiv G_i = A_i \square B_i \qquad G_{0:0} \equiv G_0 = C_{in}$$
$$P_{i:i} \equiv P_i = A_i \oplus B_i \qquad P_{0:0} \equiv P_0 = 0$$

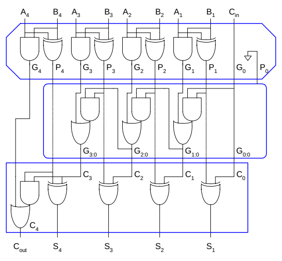
- Sum: $S_i = P_i \oplus G_{i-1:0}$

PG Logic


A_4 B_4 A_3 B_3 A_2 B_2 A_1 B_1 C_{in}
G_4 P_4 G_3 P_3 G_2 P_2 G_1 P_1 G_0 P_0
1: Bitwise PG logic
2: Group PG logic
$G_{3:0}$ $G_{2:0}$ $G_{1:0}$ $G_{0:0}$
C_3 C_2 C_1 C_0
3: Sum logic
C_4
C_{out} S_4 S_3 S_2 S_1

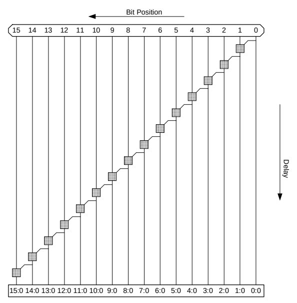
Ripple Carry Revisited

$G_{i:0} = G_i + P_i \cdot G_{i-1:0}$

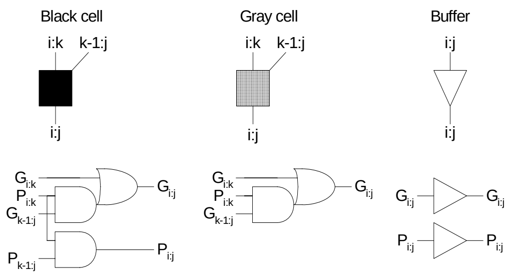


Ripple Carry PG Diagram

$t_{\text{ripple}} = t_{pg} + (N-1)t_{AO} + t_{\text{xor}}$

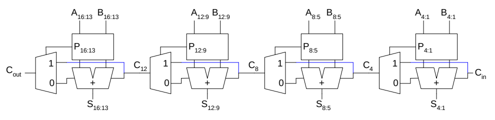


PG Diagram Notation

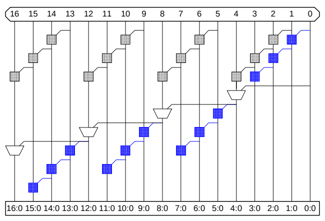


Carry-Skip Adder

- Ripple carry is slow through all N stages
- Carry-skip allows carry to skip over groups of *n* bits
  - Decision based on *n*-bit propagate signal



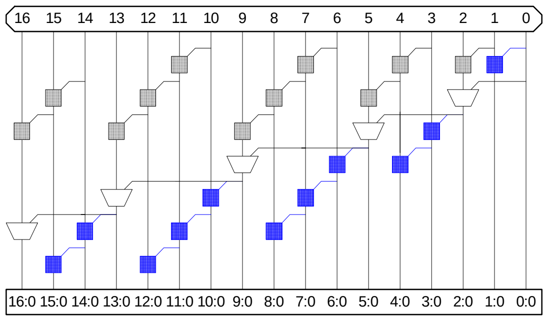
Carry-Skip PG Diagram



For k n-bit groups (N = nk)

$$t_{\text{skip}} = t_{pg} + \left[2(n-1) + (k-1)\right]t_{AO} + t_{\text{xor}}$$

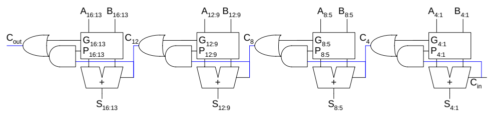
Variable Group Size



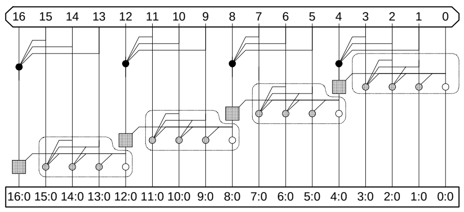
Delay grows as O(sqrt(N))

Carry-Lookahead Adder

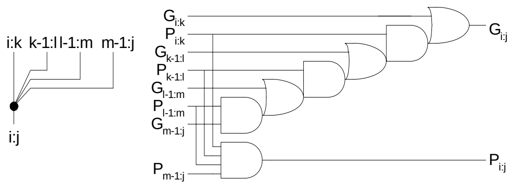
- Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel
- Uses higher-valency cells with more than two inputs



CLA PG Diagram

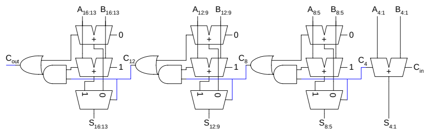


Higher-Valency Cells



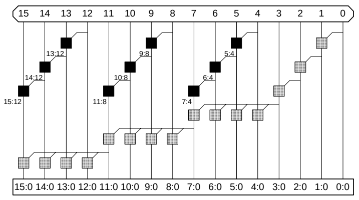
Carry-Select Adder

- Trick for critical paths dependent on late input X
  - Precompute two possible outputs for X = 0, 1
  - Select proper output when X arrives
- Carry-select adder precomputes n-bit sums
  - For both possible carries into n-bit group



Carry-Increment Adder

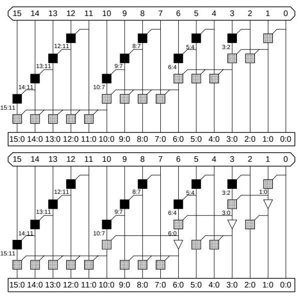
- Factor initial PG and final XOR out of carry-select



$$t_{\text{increment}} = t_{pg} + \left[(n-1) + (k-1)\right] t_{AO} + t_{\text{xor}}$$

Variable Group Size

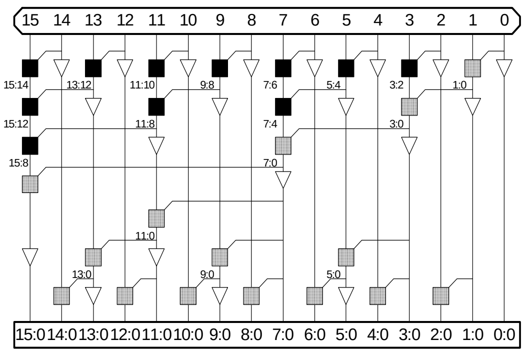
- Also buffer noncritical signals



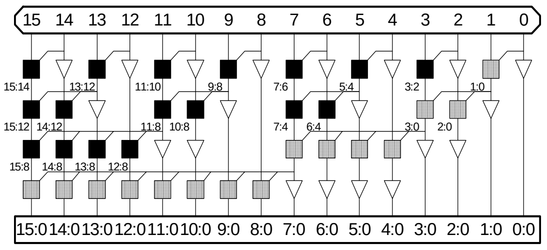
Tree Adder

- If lookahead is good, lookahead across lookahead!
  - Recursive lookahead gives O(log N) delay
- Many variations on tree adders

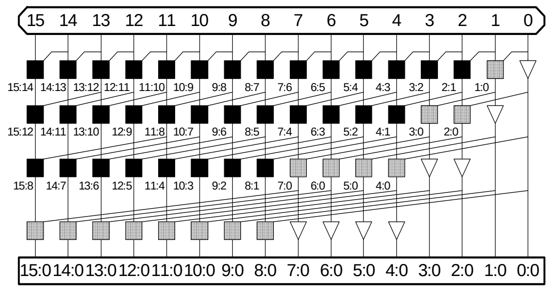
Brent-Kung



Sklansky



Kogge-Stone

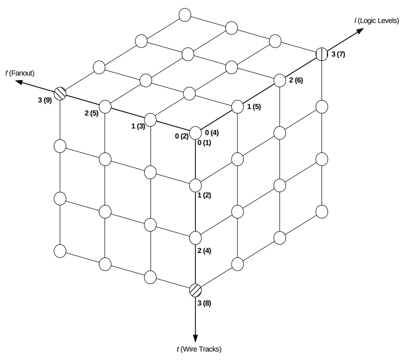


Tree Adder Taxonomy

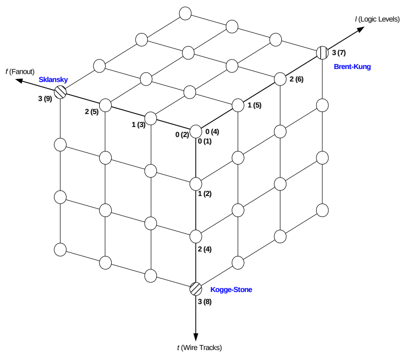
- Ideal N-bit tree adder would have
  - *L* = log N logic levels
  - Fanout never exceeding 2
  - No more than one wiring track between levels
- Describe adder with 3-D taxonomy (*l*, *f*, *t*)
  - Logic levels: $L + l$
  - Fanout: $2^f + 1$
  - Wiring tracks: 2^t
- Known tree adders sit on plane defined by

  $l + f + t = L\text{-}1$

Tree Adder Taxonomy

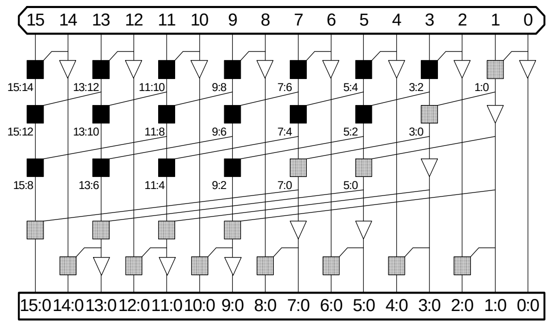


Tree Adder Taxonomy



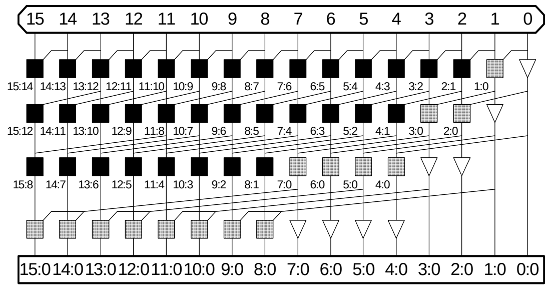
D. Z. Pan 8. Design of Adders 31

Han-Carlson



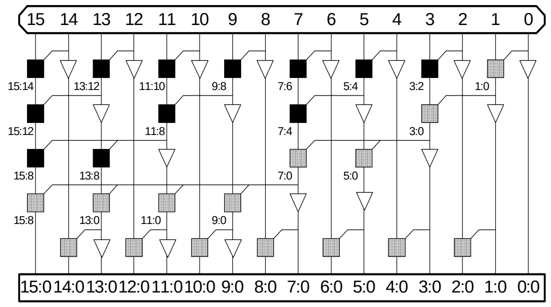
D. Z. Pan 8. Design of Adders 32

Knowles [2, 1, 1, 1]



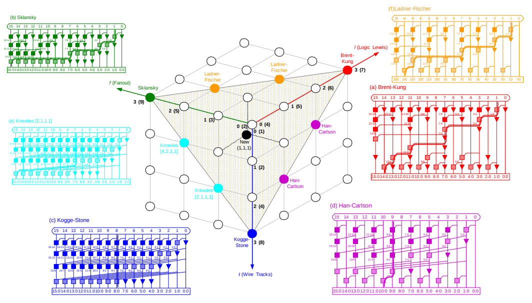
D. Z. Pan 8. Design of Adders 33

Ladner-Fischer



D. Z. Pan 8. Design of Adders 34

Taxonomy Revisited



D. Z. Pan 8. Design of Adders 35

Summary

Adder architectures offer area / power / delay tradeoffs.

Choose the best one for your application.

Architecture	Classification	Logic Levels	Max Fanout	Tracks	Cells
Ripple Carry		N-1	1	1	N
Carry-Skip n=4		N/4 + 5	2	1	1.25N
Carry-Inc. n=4		N/4 + 2	4	1	2N
Brent-Kung	(L-1, 0, 0)	$2\log_2 N - 1$	2	1	2N
Sklansky	(0, L-1, 0)	$\log_2 N$	N/2 + 1	1	$0.5 N\log_2 N$
Kogge-Stone	(0, 0, L-1)	$\log_2 N$	2	N/2	$N\log_2 N$

D. Z. Pan 8. Design of Adders 36