

8. Design of Adders

J. A. Abraham

Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 382M.7 – VLSI I
Fall 2011

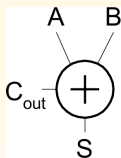
September 21, 2011

Single-Bit Addition

Half Adder

$$S = A \oplus B$$

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

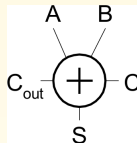


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

$$S = A \oplus B \oplus C$$

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



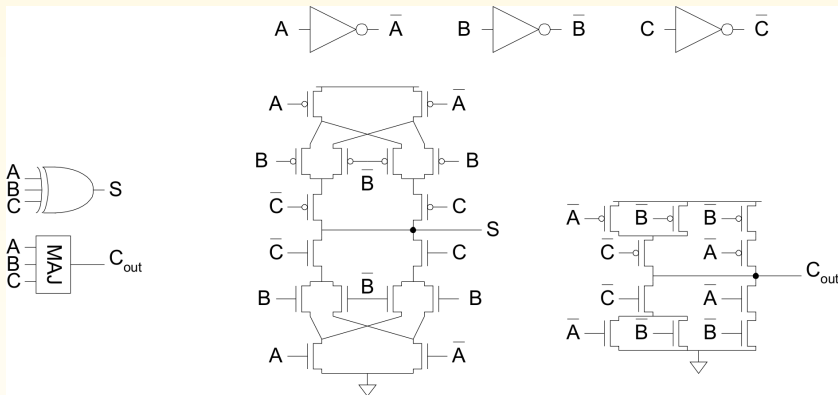
A	B	C	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full Adder Design I

Brute force implementation from equations

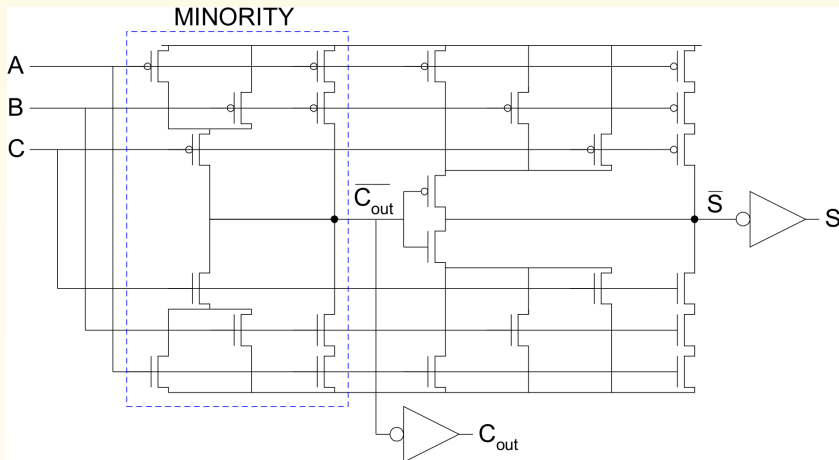
$$S = A \oplus B \oplus C$$

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



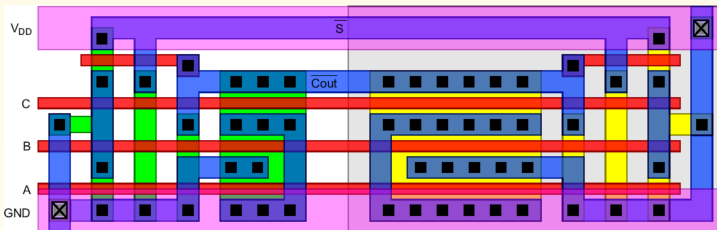
Full Adder Design II

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



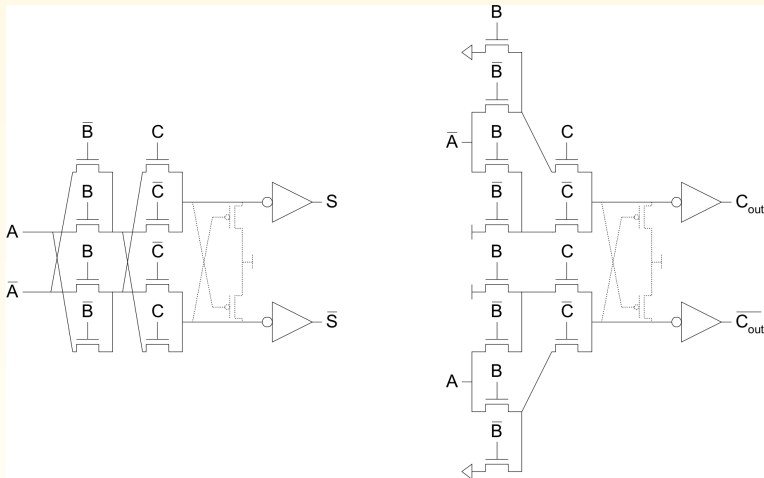
Layout of Full Adder

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



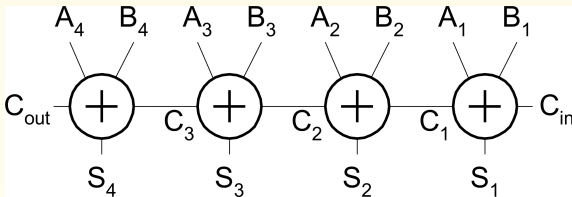
Full Adder Design III

- Complementary Pass Transistor Logic (CPL)
 - Slightly faster, but more area



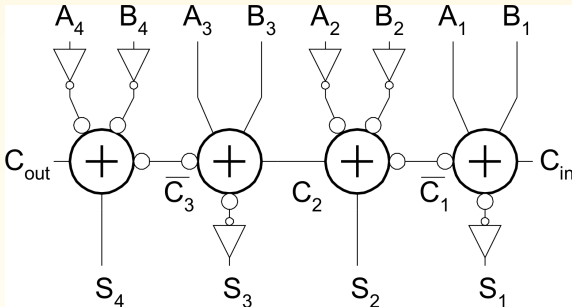
Ripple Carry Adder

- Simplest design: cascade full adders
 - Critical path goes from C_{in} to C_{out}
 - Design full adder to have fast carry (small delay for carry signal)



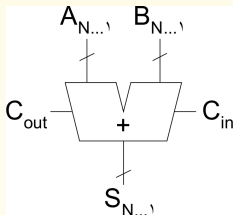
Deal with Inversions to Speed Up Carry Path

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



Carry Propagate Adders

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



$$\begin{array}{r} \text{C}_{out} \swarrow \quad \nwarrow \text{C}_{in} \\ \textcircled{0}0000\textcircled{0} \\ 1111 \\ +0000 \\ \hline 1111 \end{array}$$

$$\begin{array}{r} \text{C}_{out} \swarrow \quad \nwarrow \text{C}_{in} \\ \textcircled{1}1111\textcircled{1} \\ 1111 \\ +0000 \\ \hline 0000 \end{array} \quad \begin{array}{l} \text{carries} \\ A_{4...1} \\ B_{4...1} \\ S_{4...1} \end{array}$$

Carry Propagate, Generate, Kill (P, G, K)

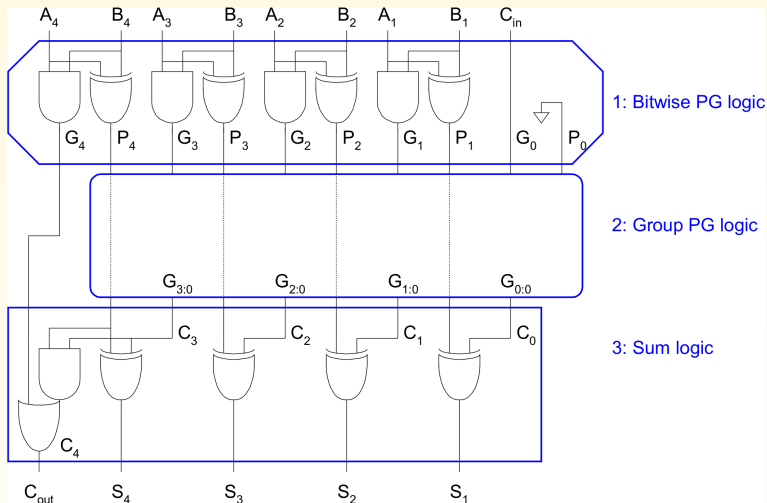
For a full adder, define what happens to carries

- **Generate:** $C_{out} = 1$, independent of C
 - $G = A \cdot B$
- **Propagate:** $C_{out} = C$
 - $P = A \oplus B$
- **Kill:** $C_{out} = 0$, independent of C
 - $K = \overline{A} \cdot \overline{B}$

Generate and Propagate for groups spanning i:j

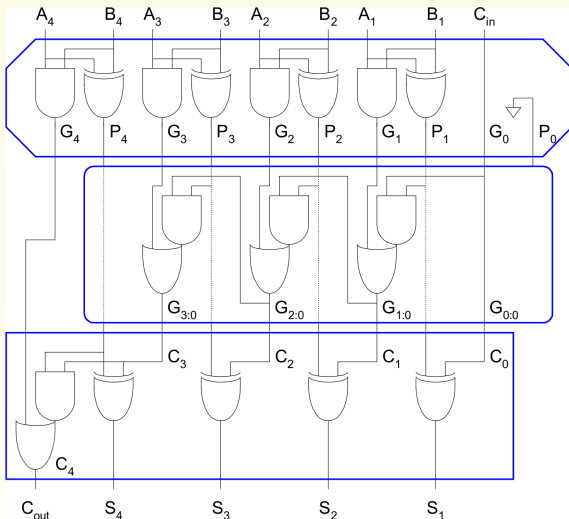
- $G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$
- $P_{i:j} = P_{i:k} \cdot P_{k-1:j}$
- Base Case
 - $G_{i:i} \equiv G_i = A_i \cdot B_i, \quad G_{0:0} = G_0 = C_{in}$
 - $P_{i:i} \equiv P_i = A_i \oplus B_i, \quad P_{0:0} = P_0 = 0$
- Sum: $S_i = P_i \oplus G_{i-1:0}$

PG Logic



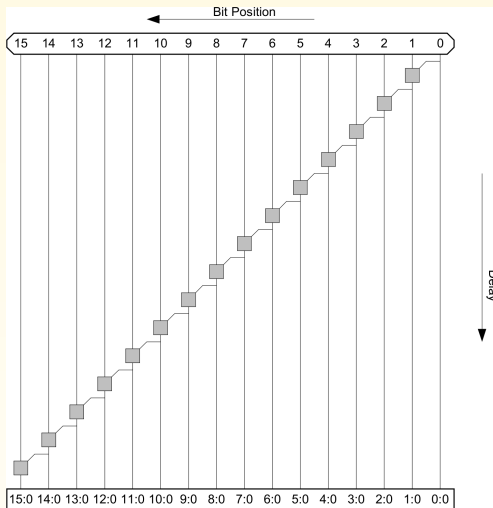
Ripple Carry Adder Revisited in the PG Framework

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



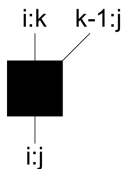
Ripple Carry PG Diagram

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

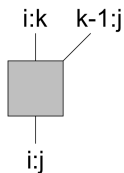


PG Diagram Notation

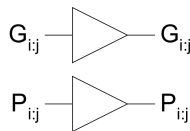
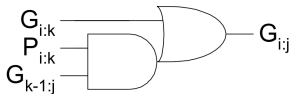
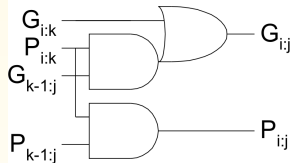
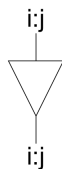
Black cell



Gray cell

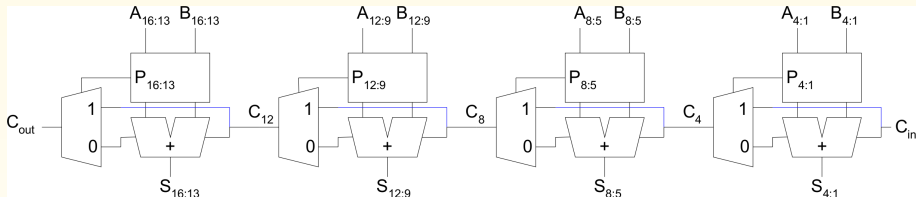


Buffer

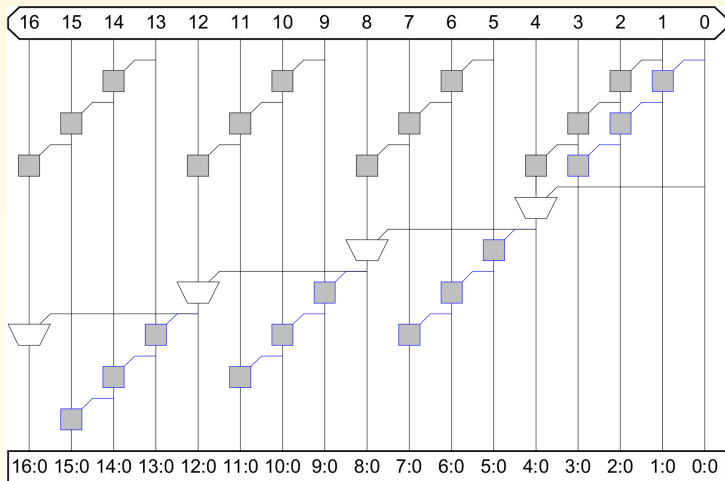


Carry-Skip Adder

- Carry-ripple 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

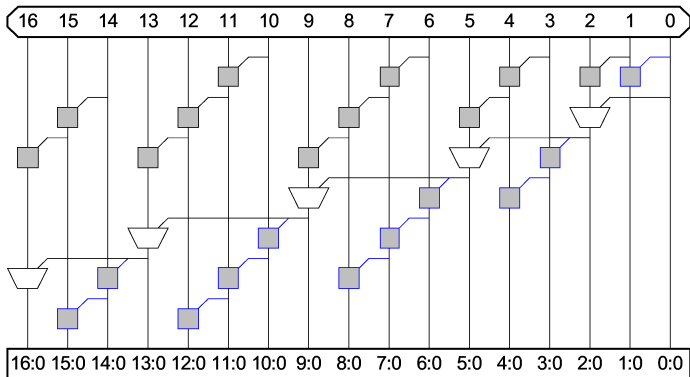


For k n -bit groups ($N = nk$)

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

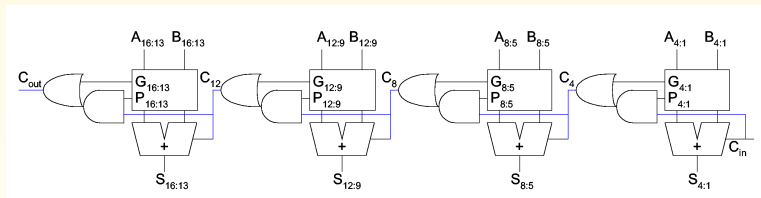
Variable Group Size

Delay grows as $O(\sqrt{N})$

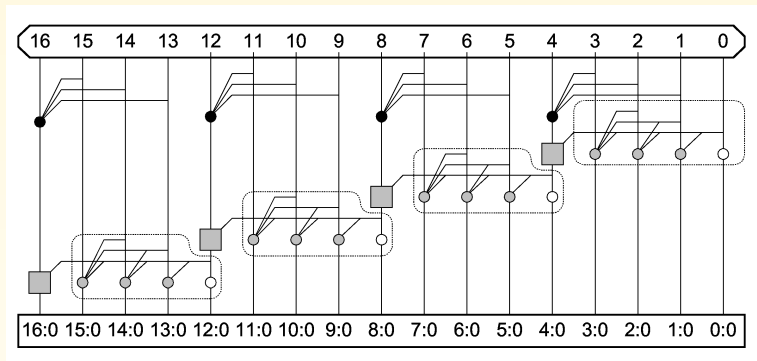


Carry-Lookahead Adder (CLA)

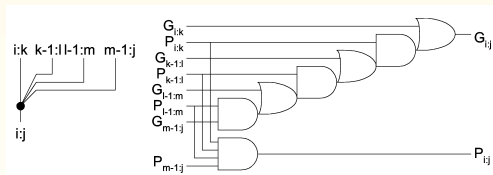
- Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel
- Uses higher-valency cells with more than two inputs



CLA PG Diagram

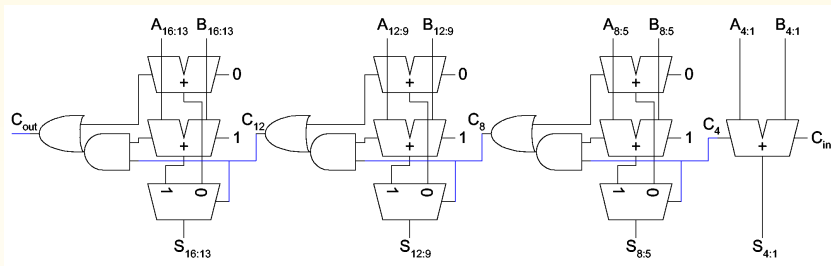


Higher Valency Cells



Carry-Select Adder

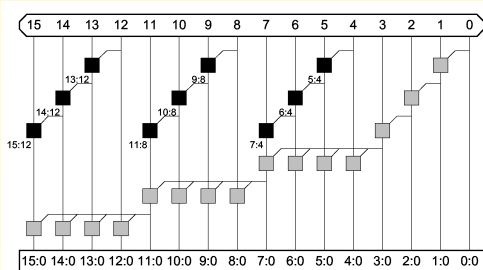
- Trick for critical paths dependent on late input X
 - Precompute two possible outputs for $X = 0, 1$
 - Select proper output when X arrives
- Carry-select adder precomputes n -bit sums for both possible carries into n -bit group



Carry-Increment Adder

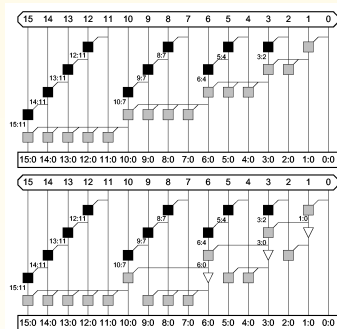
Factor initial PG and
final XOR out of
carry-select

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



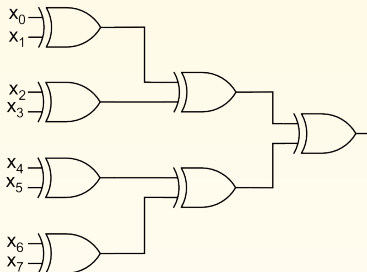
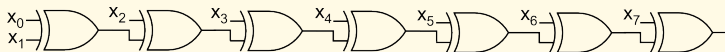
Variable Group Size

Buffer non-critical
signals to reduce
branching effort



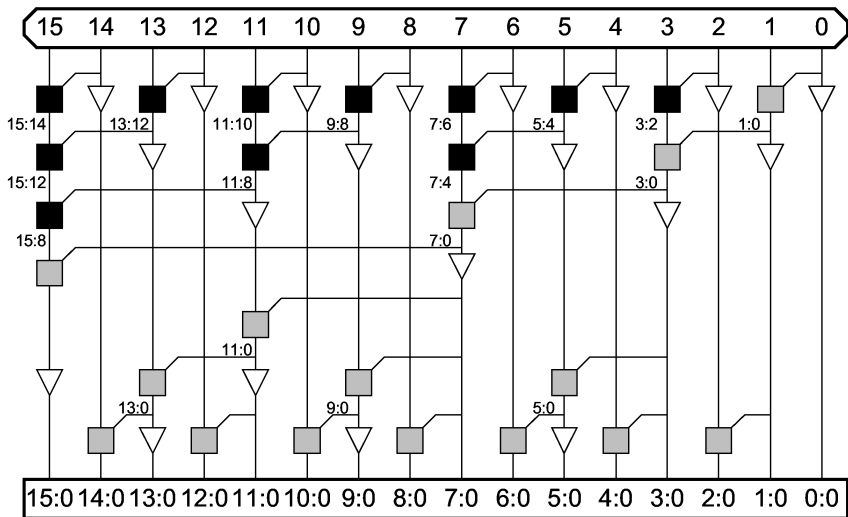
Tree Adders

- Tree structures can be used to speed up computations
- Look at computing the XOR of 8 bits using 2-input XOR-gates

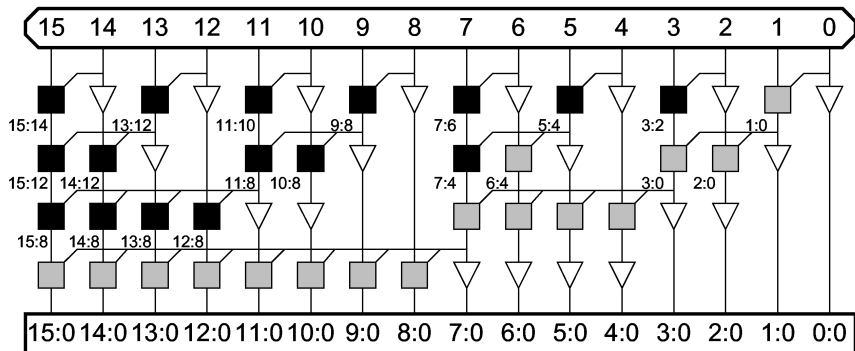


- If lookahead is good for adders, lookahead across lookahead!
 - Recursive lookahead gives $O(\log N)$ delay
- Many variations on tree adders

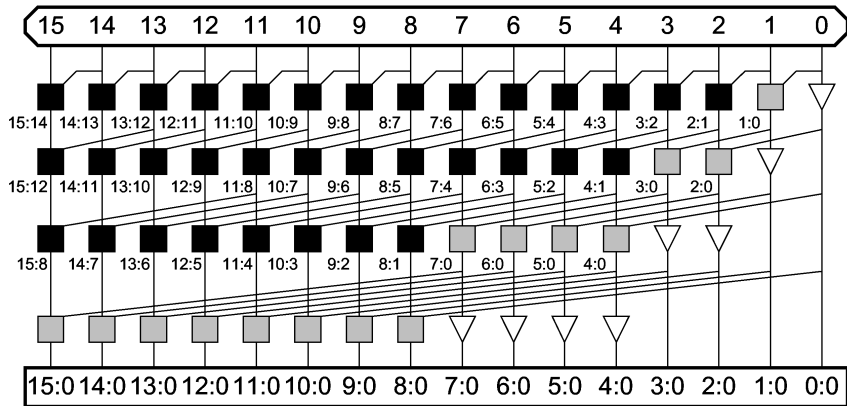
Brent-Kung Adder



Sklansky Adder



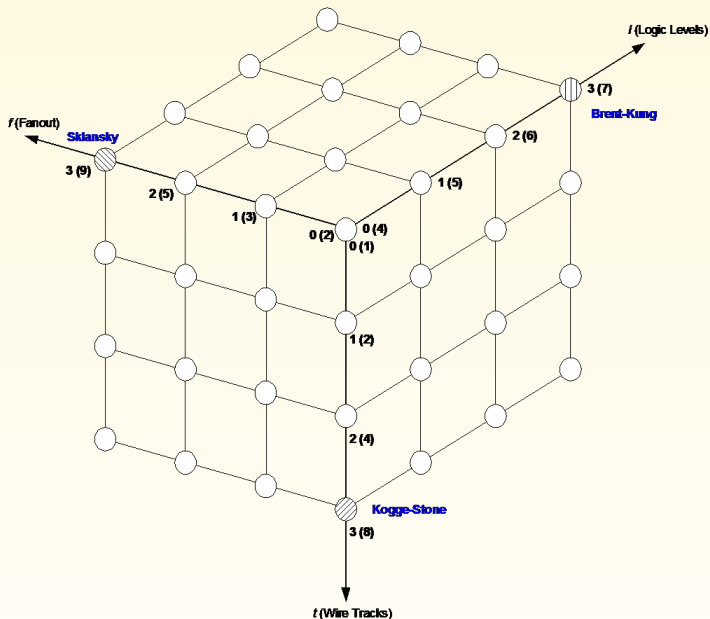
Kogge-Stone Adder



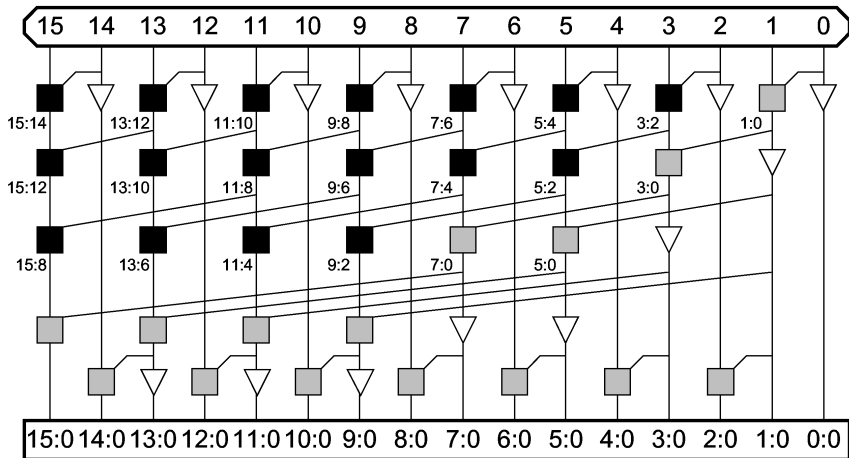
Tree Adder Taxonomy

- 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: $2f + 1$
 - Wiring tracks: 2^t
- Known tree adders sit on plane defined by $l + f + t = L - 1$

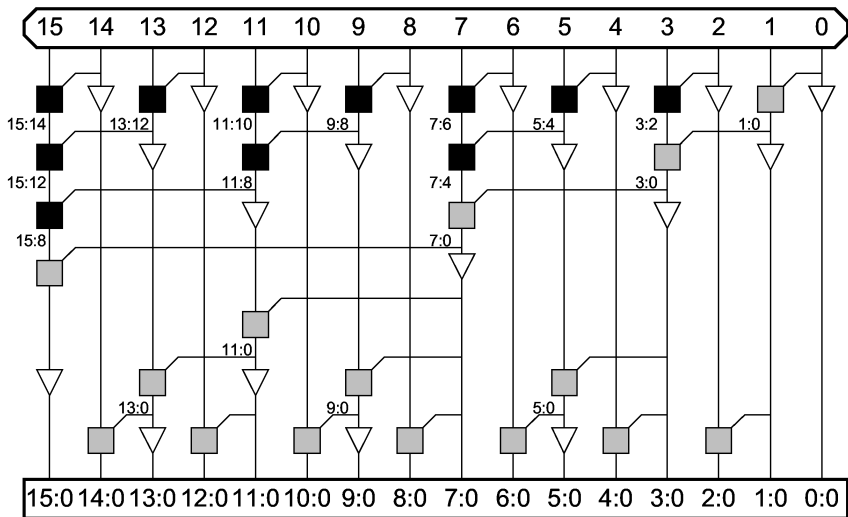
Tree Adder Taxonomy, Cont'd



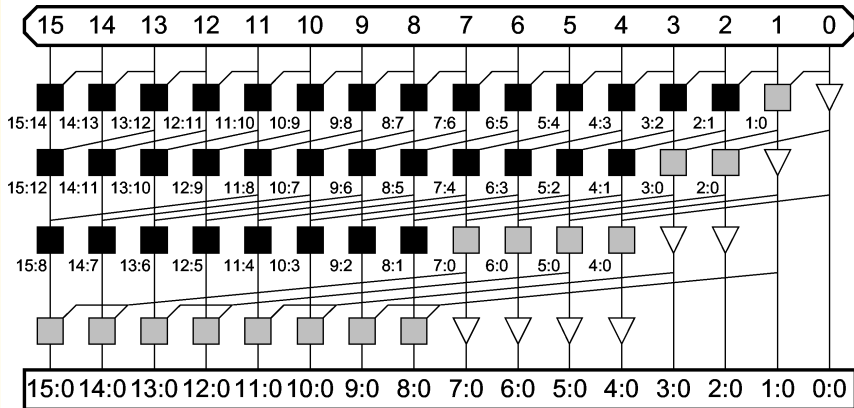
Han-Carlson Adder



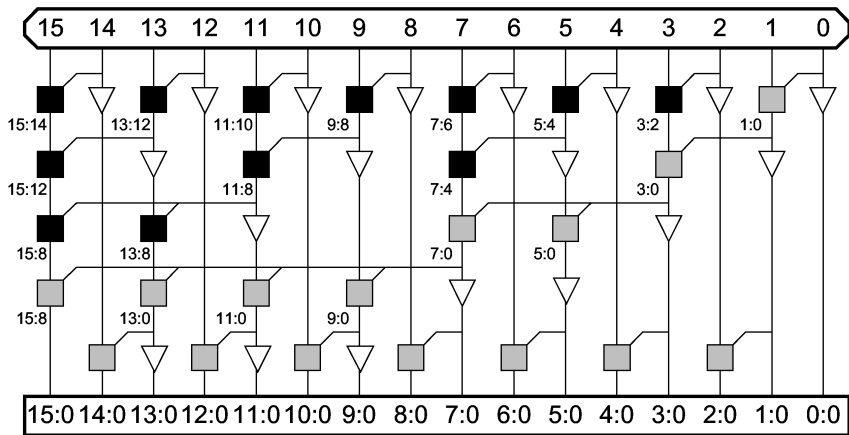
Brent-Kung Adder



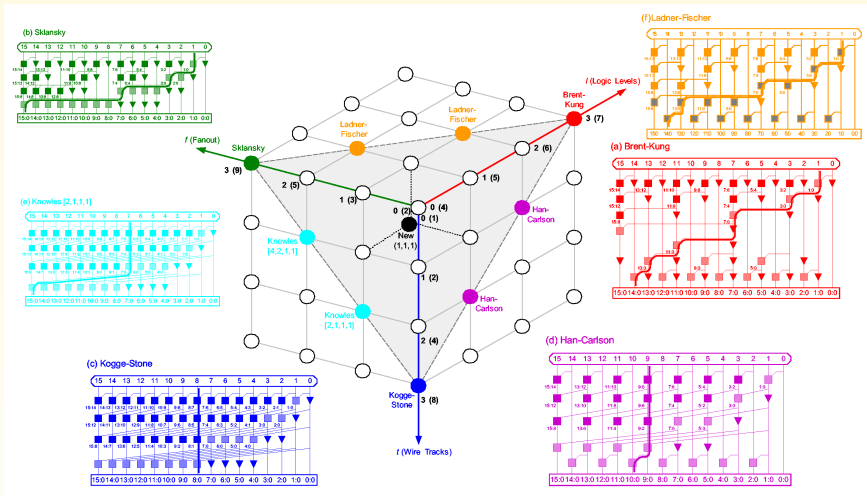
Knowles [2,1,1,1] Adder



Ladner-Fischer Adder



Tree Adder Taxonomy Revisited



Summary of Adders

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	$0.5N\log_2 N$
Kogge-Stone	(0,0,L-1)	$\log_2 N$	2	$N/2$	$N\log_2 N$