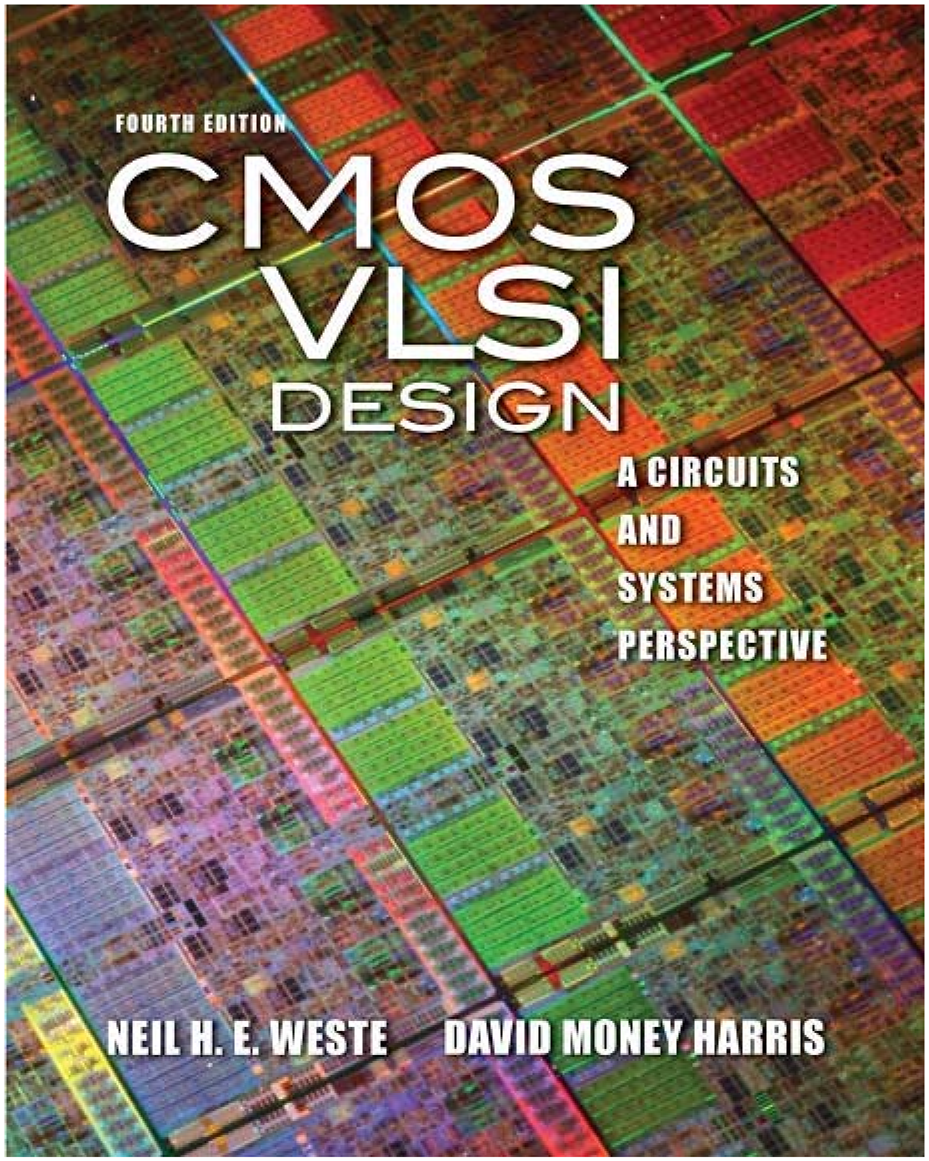


Lecture 17: Adders

Outline

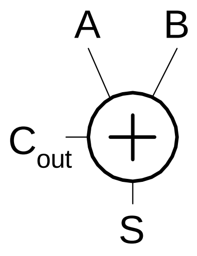
- ❑ Single-bit Addition
- ❑ Carry-Ripple Adder
- ❑ Carry-Skip Adder
- ❑ Carry-Lookahead Adder
- ❑ Carry-Select Adder
- ❑ Carry-Increment Adder
- ❑ Tree Adder

Single-Bit Addition

Half Adder

$$S = A \oplus B$$

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

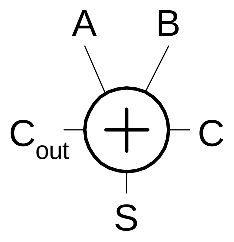


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

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	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

PGK

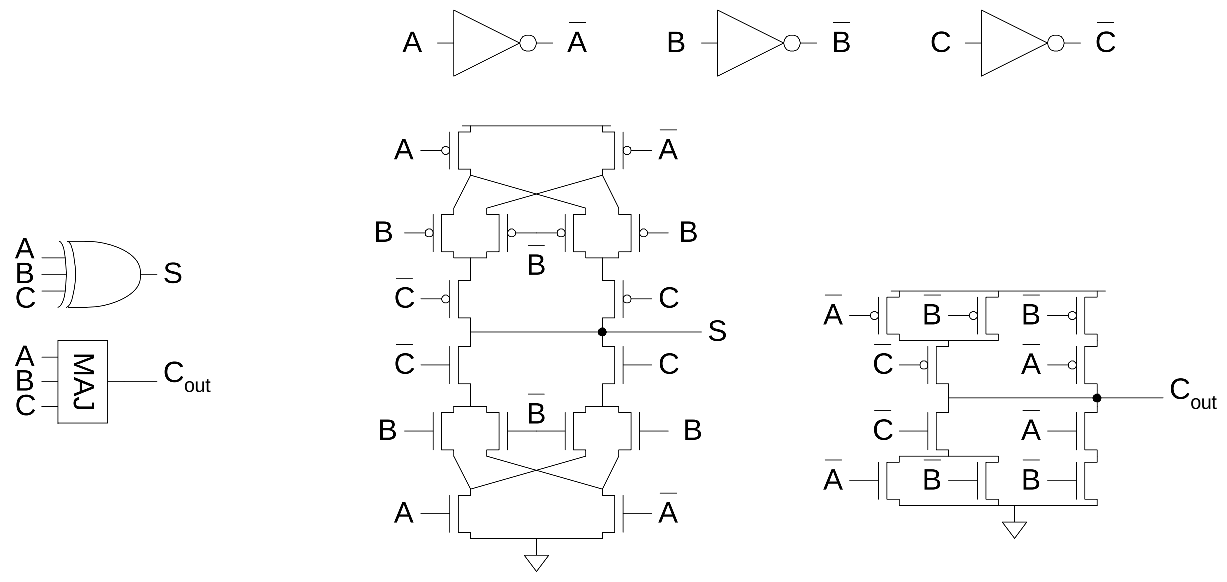
- For a full adder, define what happens to carries (in terms of A and B)
  - Generate: C_{out} = 1 independent of C
    - G =
  - Propagate: C_{out} = C
    - P =
  - Kill: C_{out} = 0 independent of C
    - K =

Full Adder Design I

- Brute force implementation from eqns

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

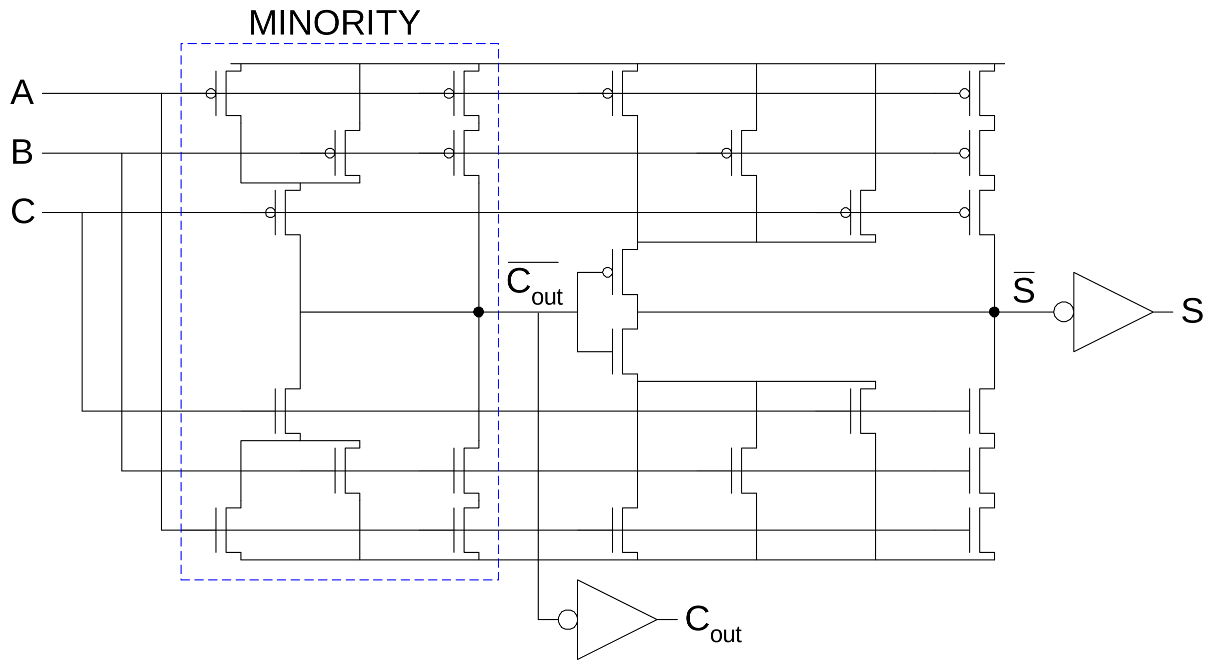
$$C_{\text{out}} = MAJ(A, B, C)$$



Full Adder Design II

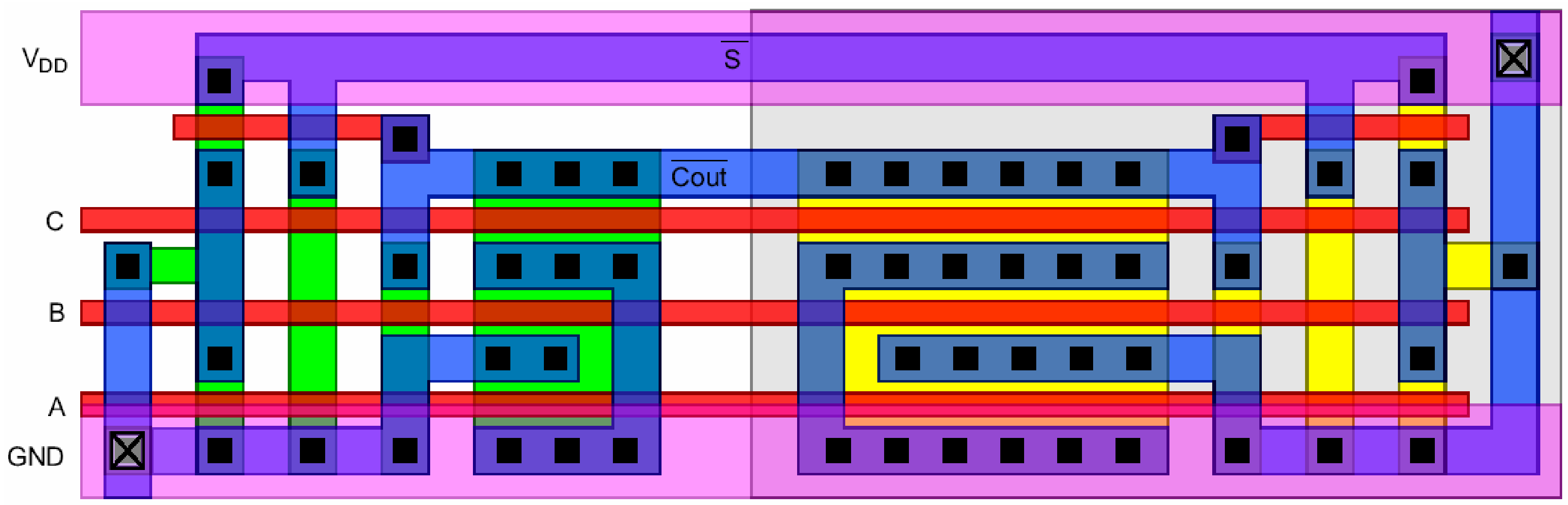
- Factor S in terms of C_{out}

  $S = ABC + (A + B + C)(\sim C_{out})$
- Critical path is usually C to C_{out} in ripple adder



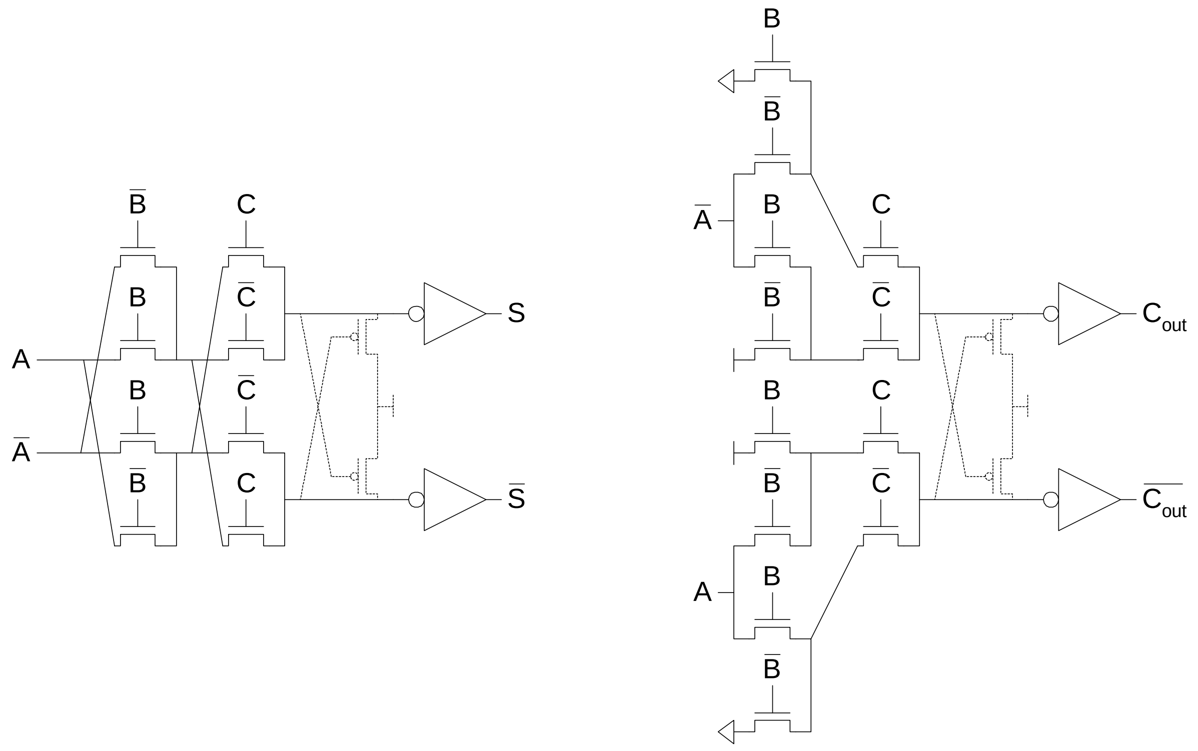
Layout

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



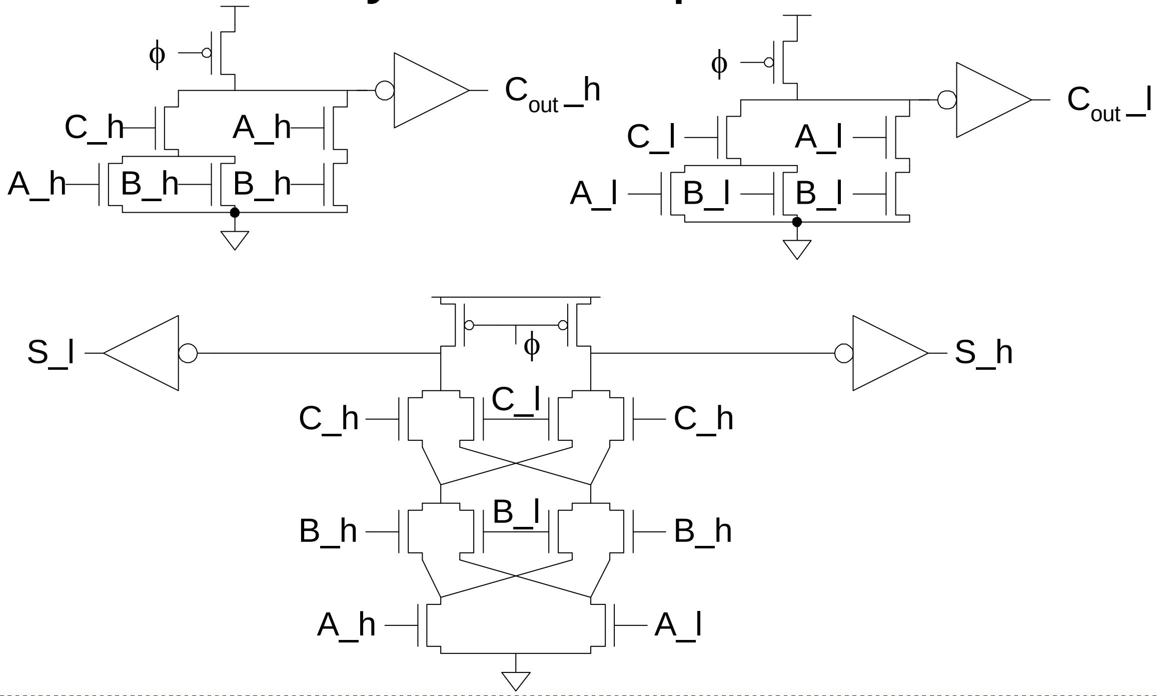
Full Adder Design III

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



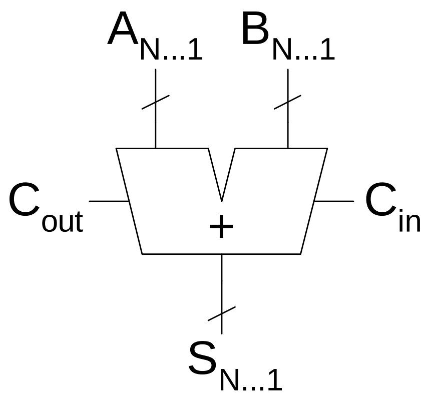
Full Adder Design IV

- Dual-rail domino
  - – Very fast, but large and power hungry
  - – Used in very fast multipliers



Carry Propagate Adders

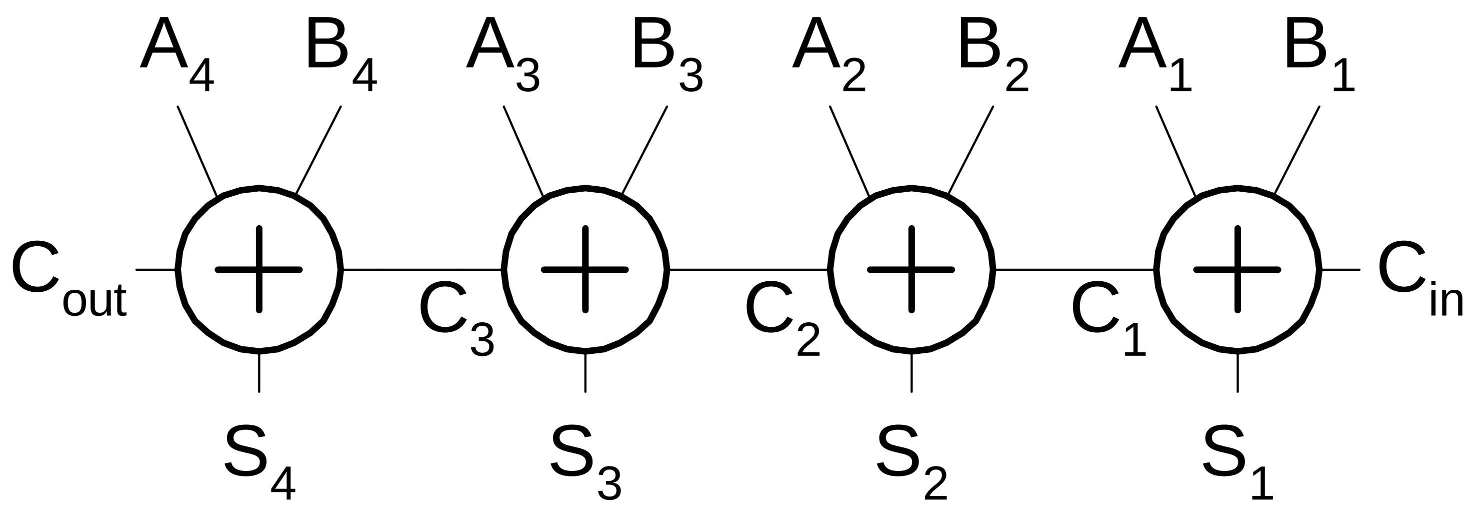
- N-bit adder called CPA
  - Each sum bit depends on all previous carries
  - How do we compute all these carries quickly?



```
  Cout   Cin         Cout   Cin
 00000              11111         carries
  1111               1111         A4...1
 +0000              +0000         B4...1
 -----              -----
  1111               0000         S4...1
```

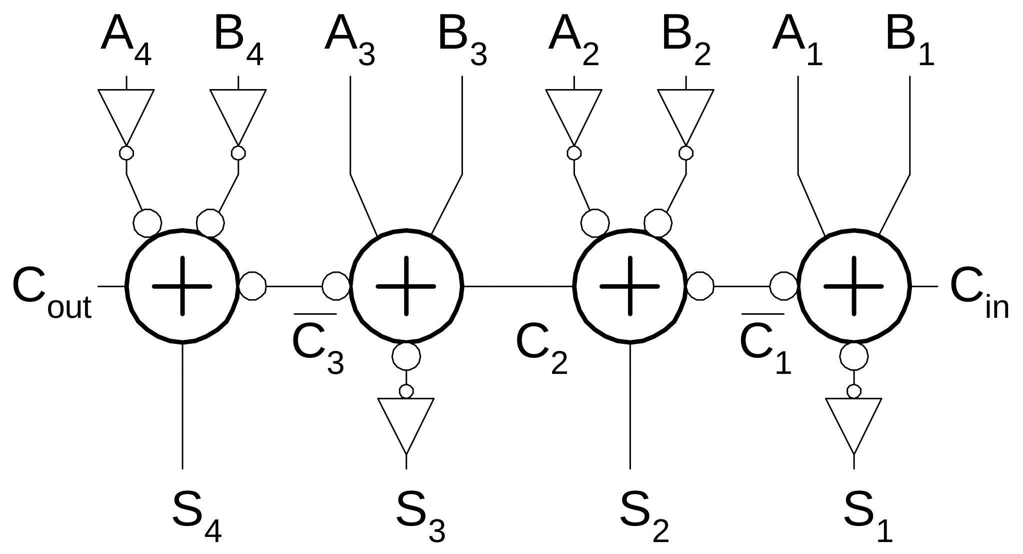
Carry-Ripple Adder

- Simplest design: cascade full adders
  - Critical path goes from C_{in} to C_{out}
  - 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



Generate / Propagate

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

$$G_{i:j} =$$

$$P_{i:j} =$$

- Base case

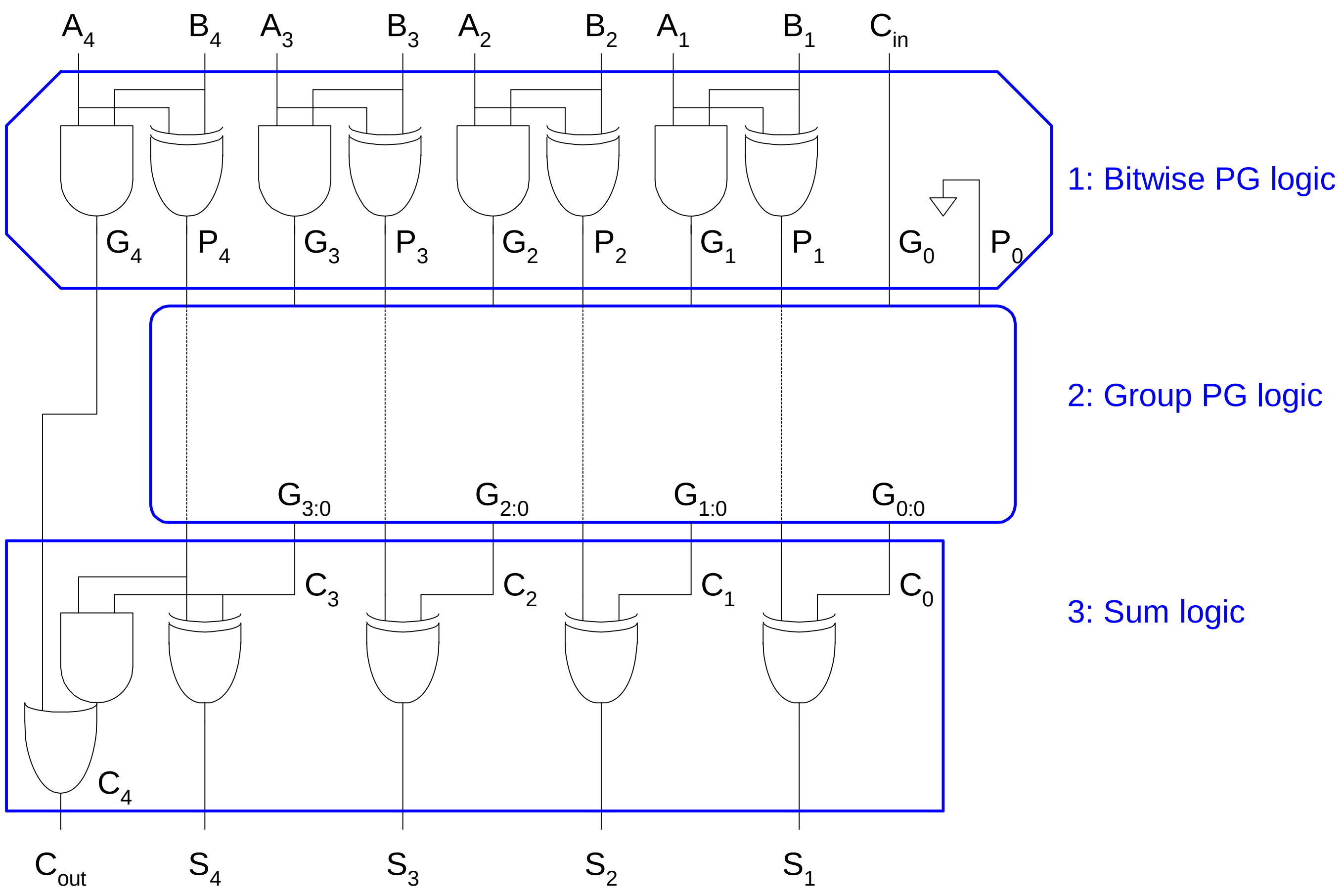
$$G_{i:i} \equiv \qquad G_{0:0} \equiv$$

$$P_{i:i} \equiv \qquad P_{0:0} \equiv$$

- Sum:

$$S_i =$$

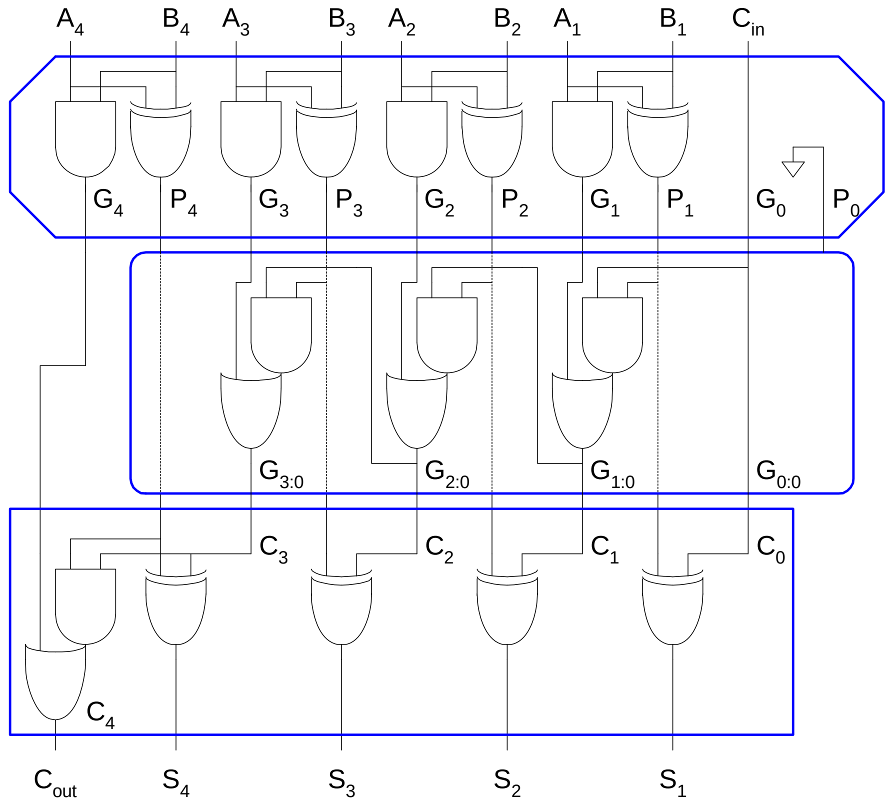
PG Logic



A_4
B_4
A_3
B_3
A_2
B_2
A_1
B_1
C_{in}
1: Bitwise PG logic
G_4
P_4
G_3
P_3
G_2
P_2
G_1
P_1
G_0
P_0
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

Carry-Ripple Revisited

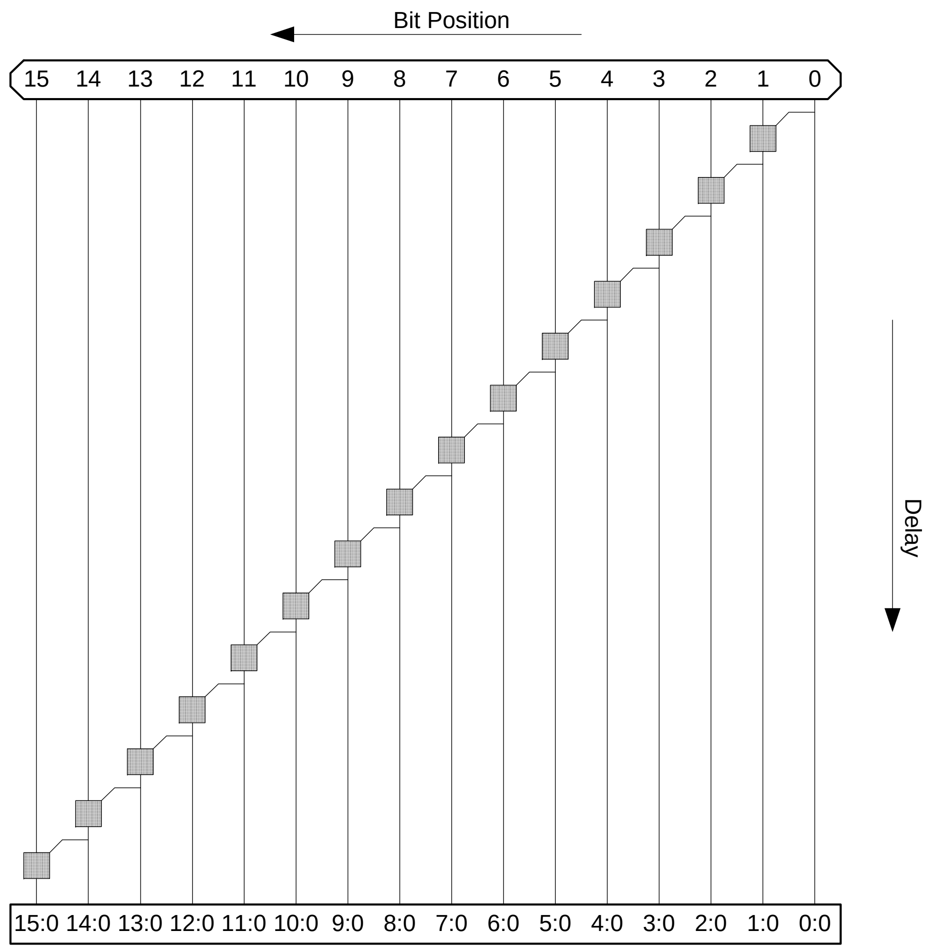
$$G_{i:0} = G_i + P_i \bullet G_{i-1:0}$$



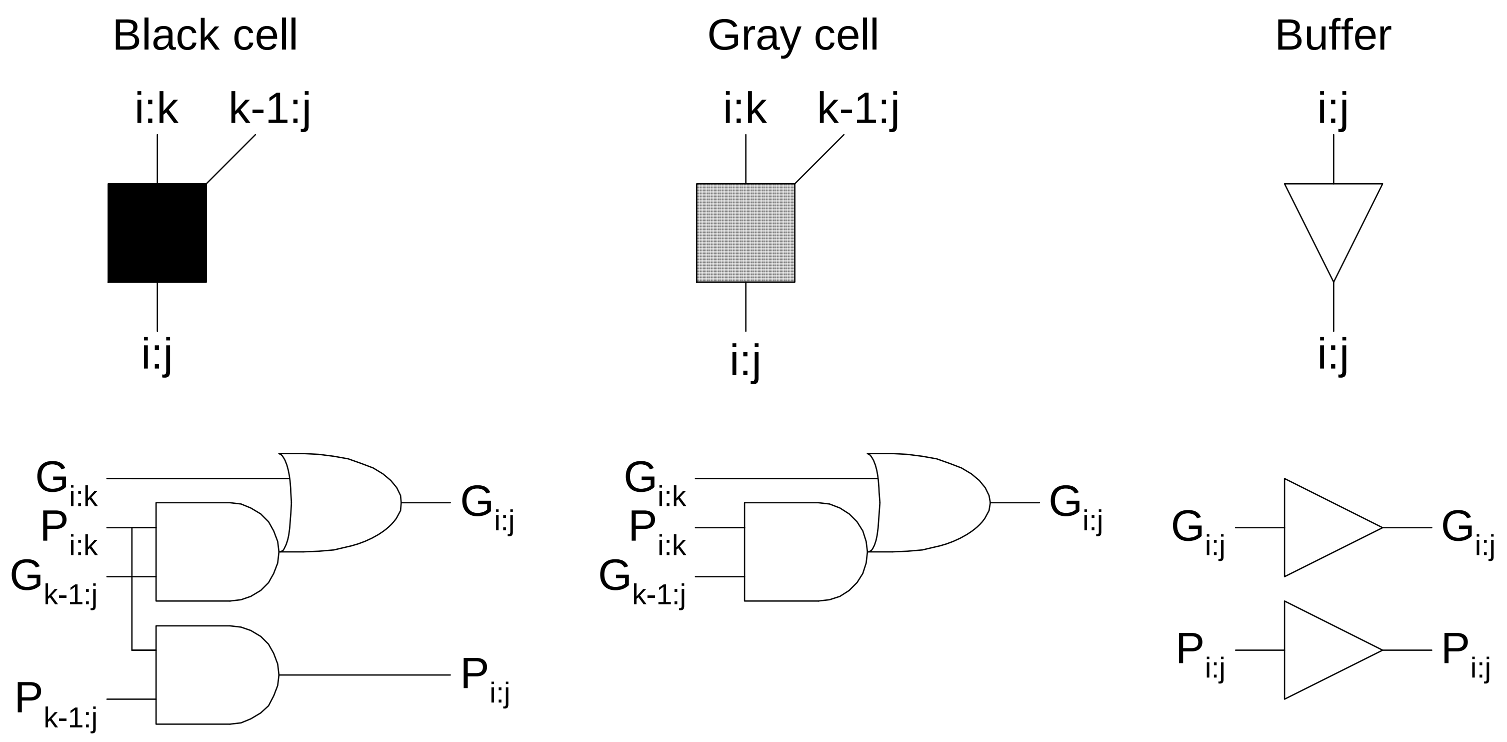
A_4 B_4 A_3 B_3 A_2 B_2 A_1 B_1 C_{in}
G_4 P_4 G_3 P_3 G_2 P_2 G_1 P_1 G_0 P_0
$G_{3:0}$ $G_{2:0}$ $G_{1:0}$ $G_{0:0}$
C_3 C_2 C_1 C_0
C_4
C_{out} S_4 S_3 S_2 S_1

Carry-Ripple PG Diagram

$t_{\text{ripple}} =$

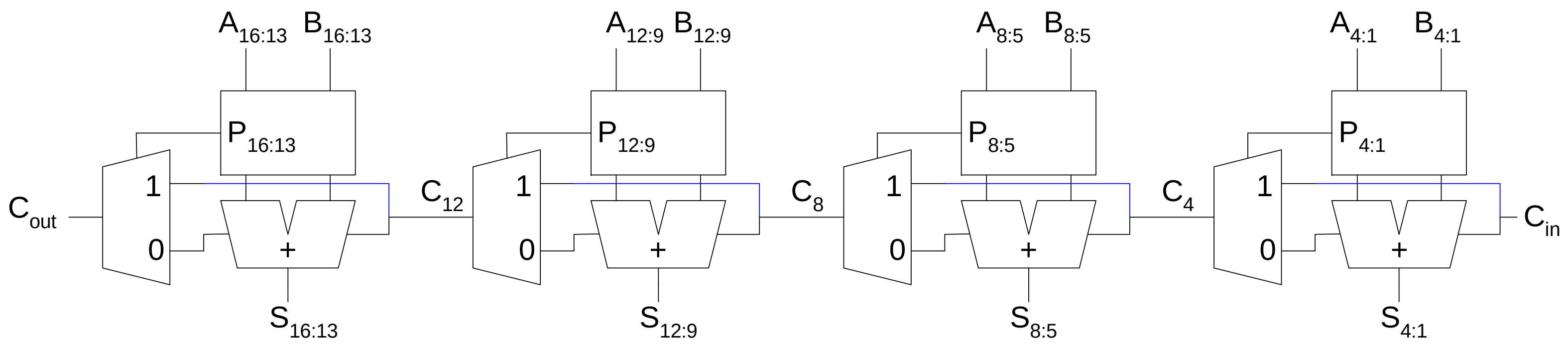


PG Diagram Notation

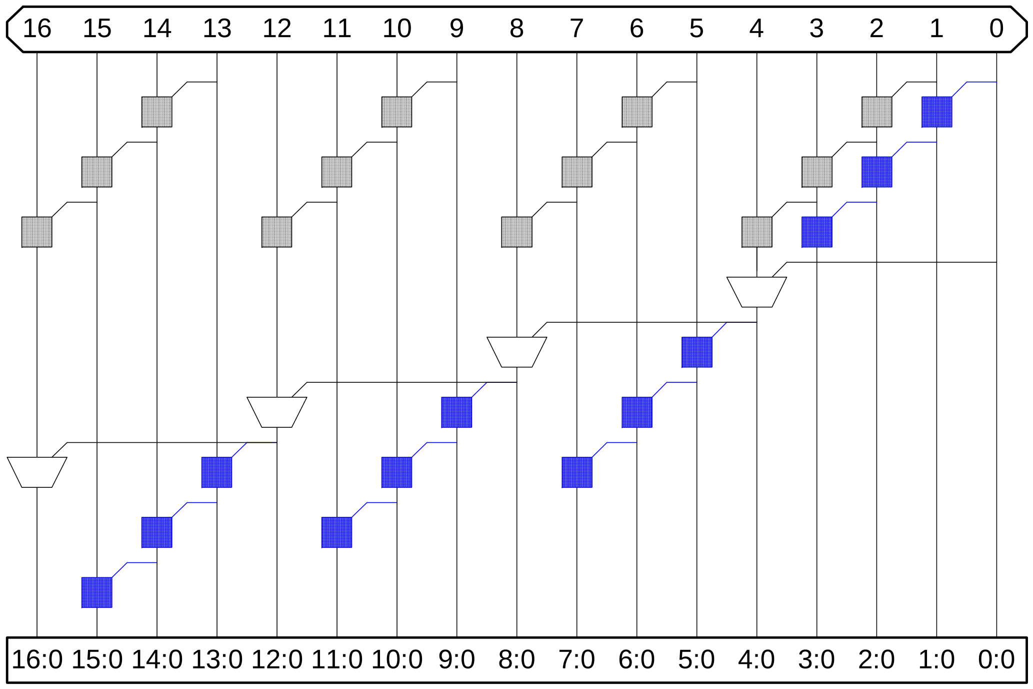


Carry-Skip Adder

- 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



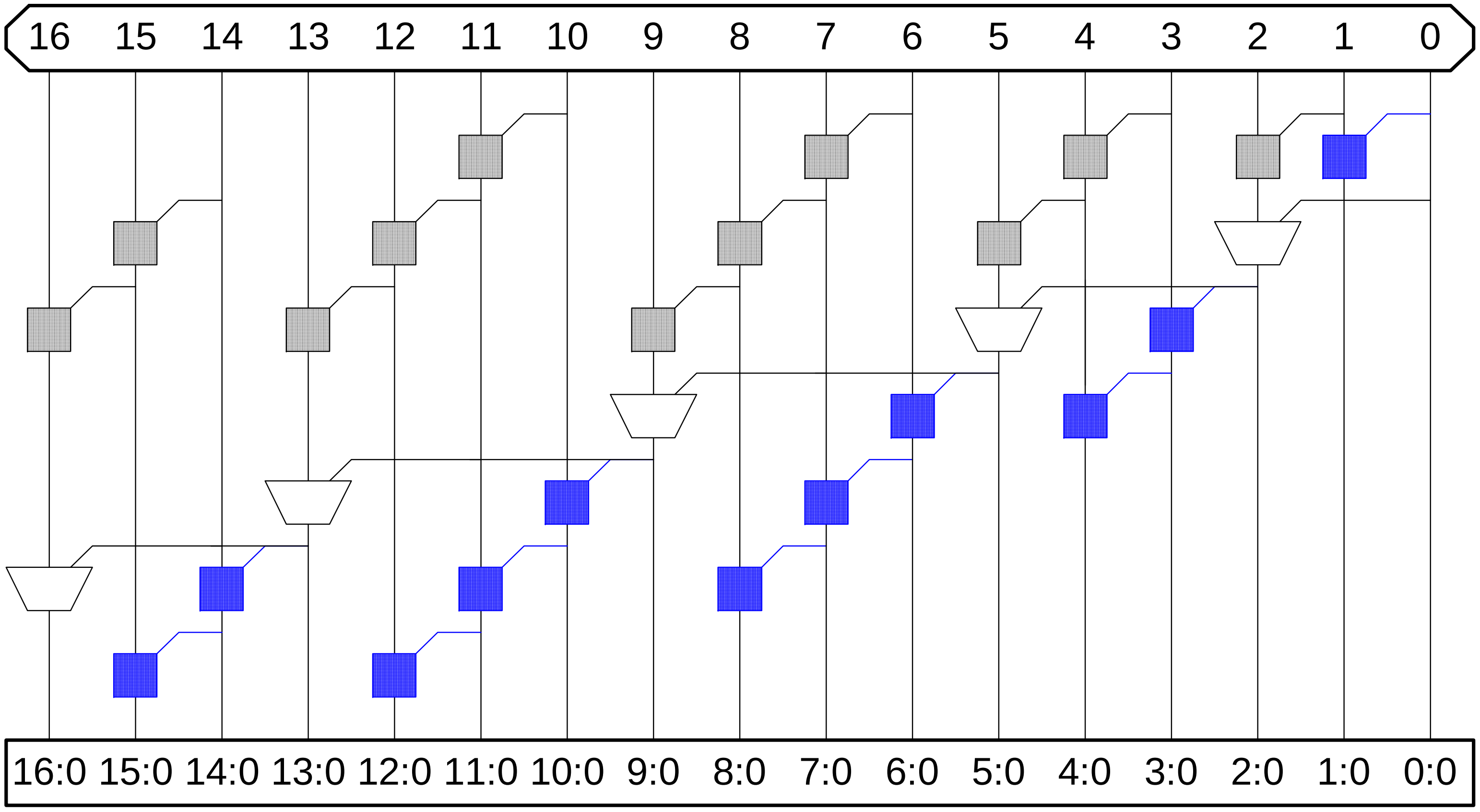
Carry-Skip PG Diagram



For k n-bit groups (N = nk)

$t_{\text{skip}} =$

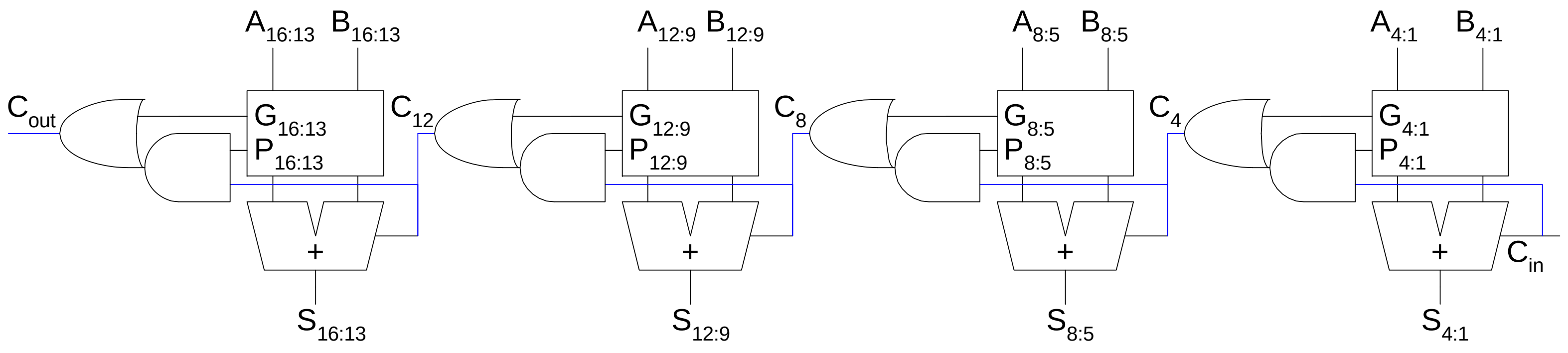
Variable Group Size



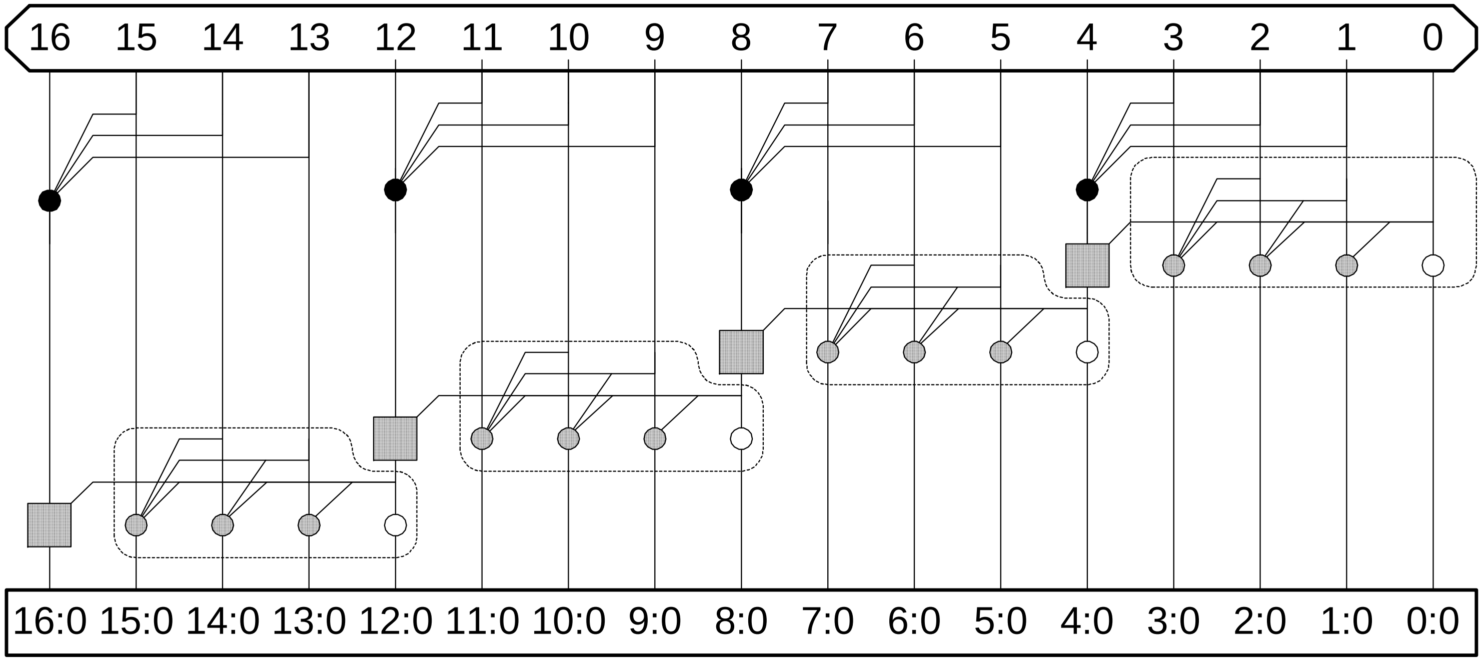
Delay grows as O(sqrt(N))

Carry-Lookahead Adder

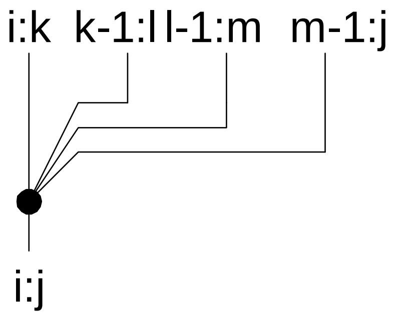
- Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel.
- Uses higher-valency cells with more than two inputs.

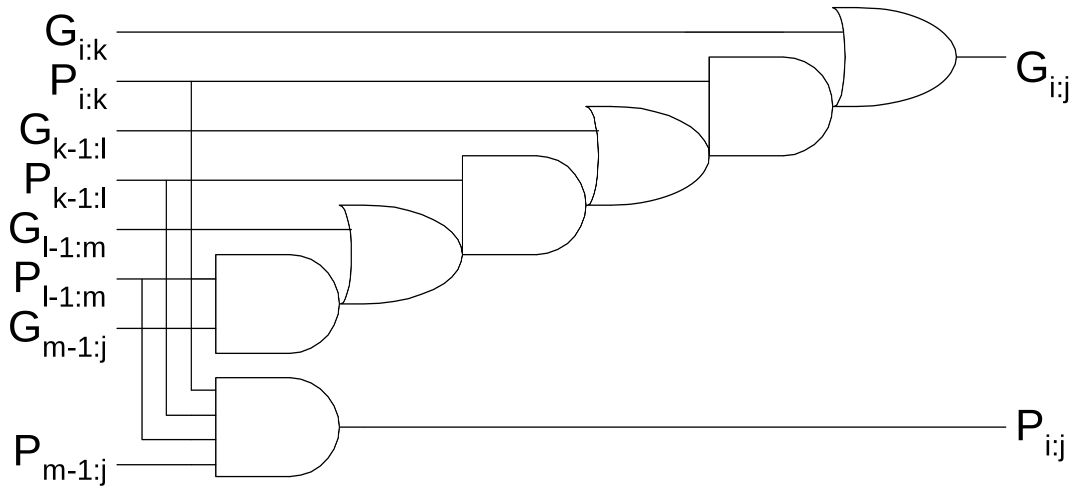


CLA PG Diagram



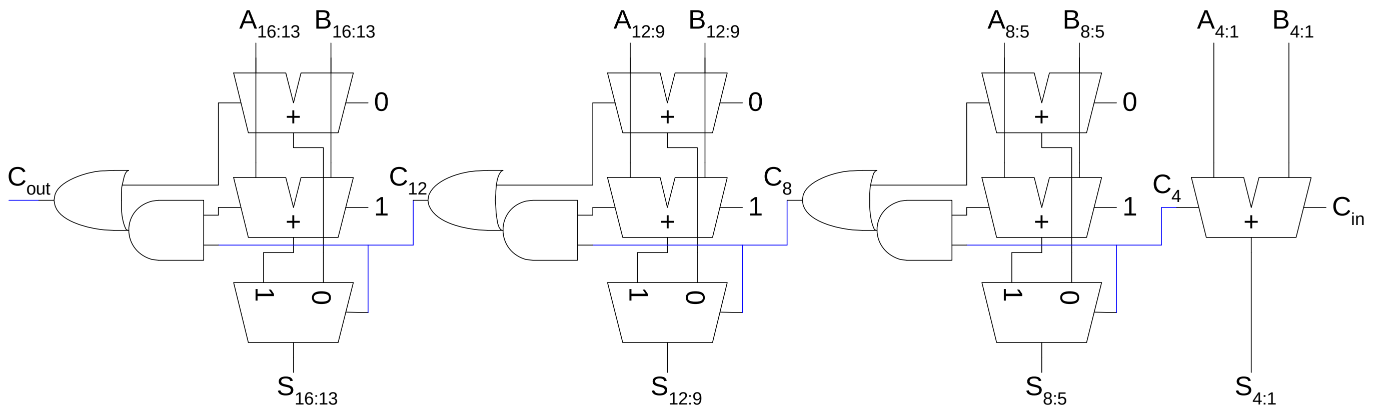
Higher-Valency Cells





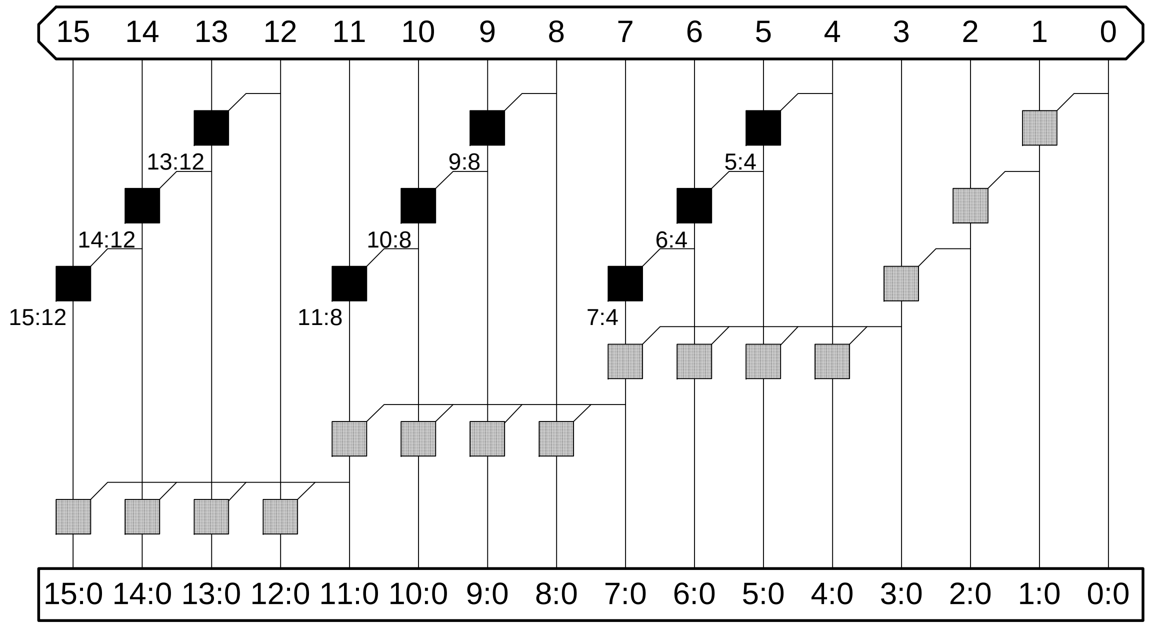
Carry-Select Adder

- ❑ 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

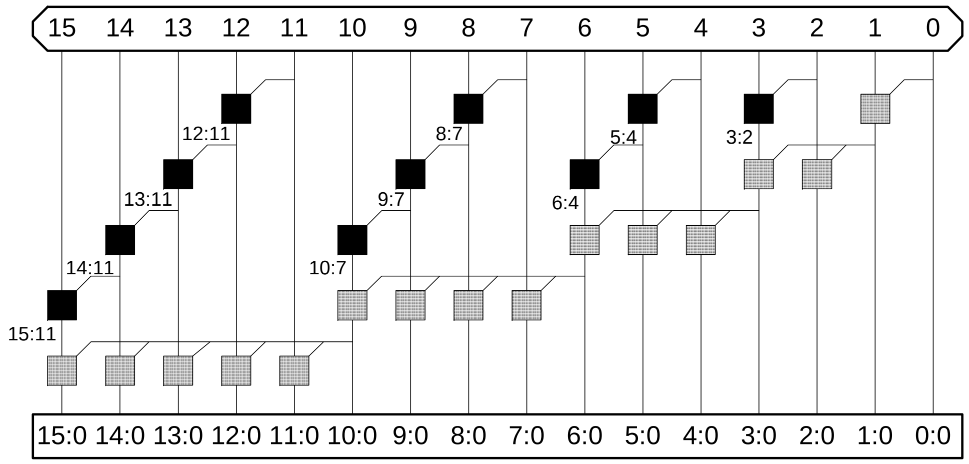
- Factor initial PG and final XOR out of carry-select

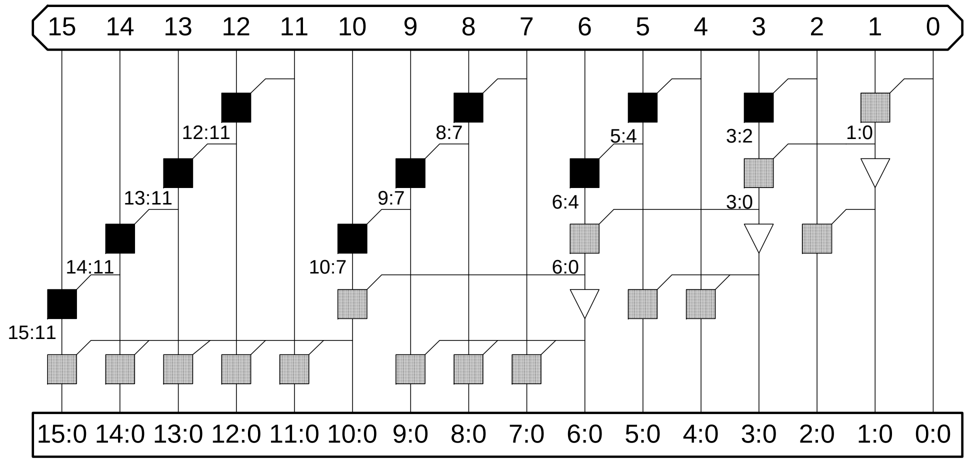


$t_{\text{increment}} =$

Variable Group Size

- Also buffer noncritical signals

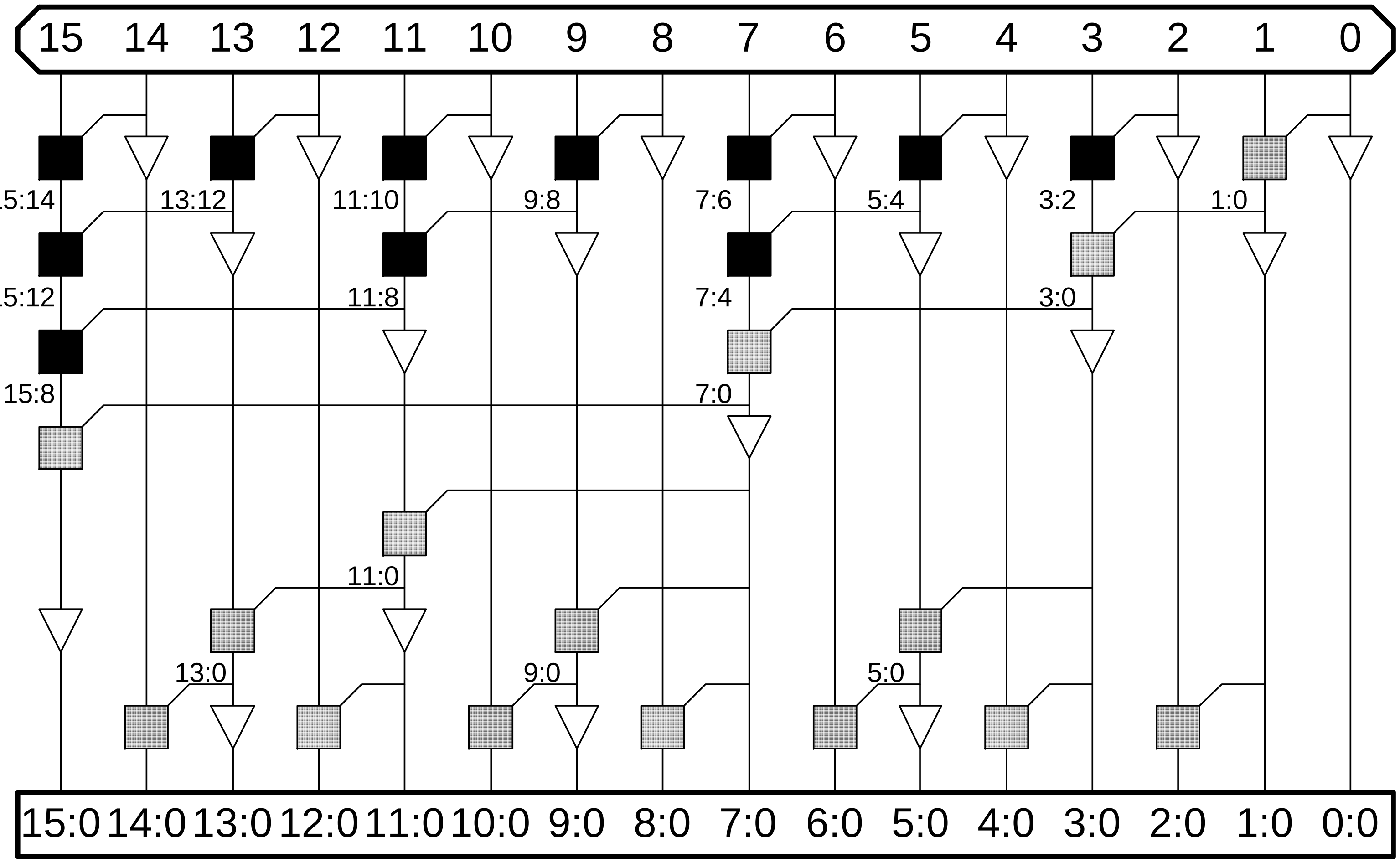




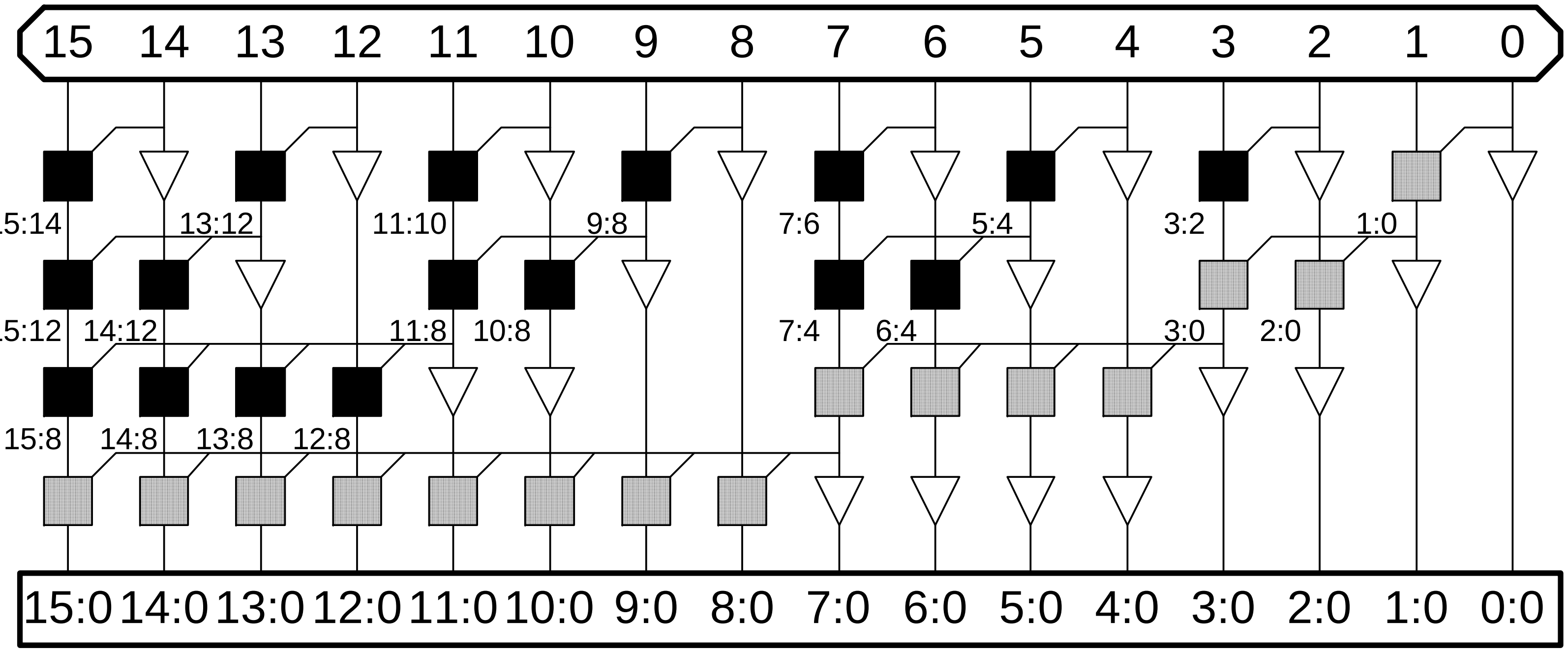
Tree Adder

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

Brent-Kung

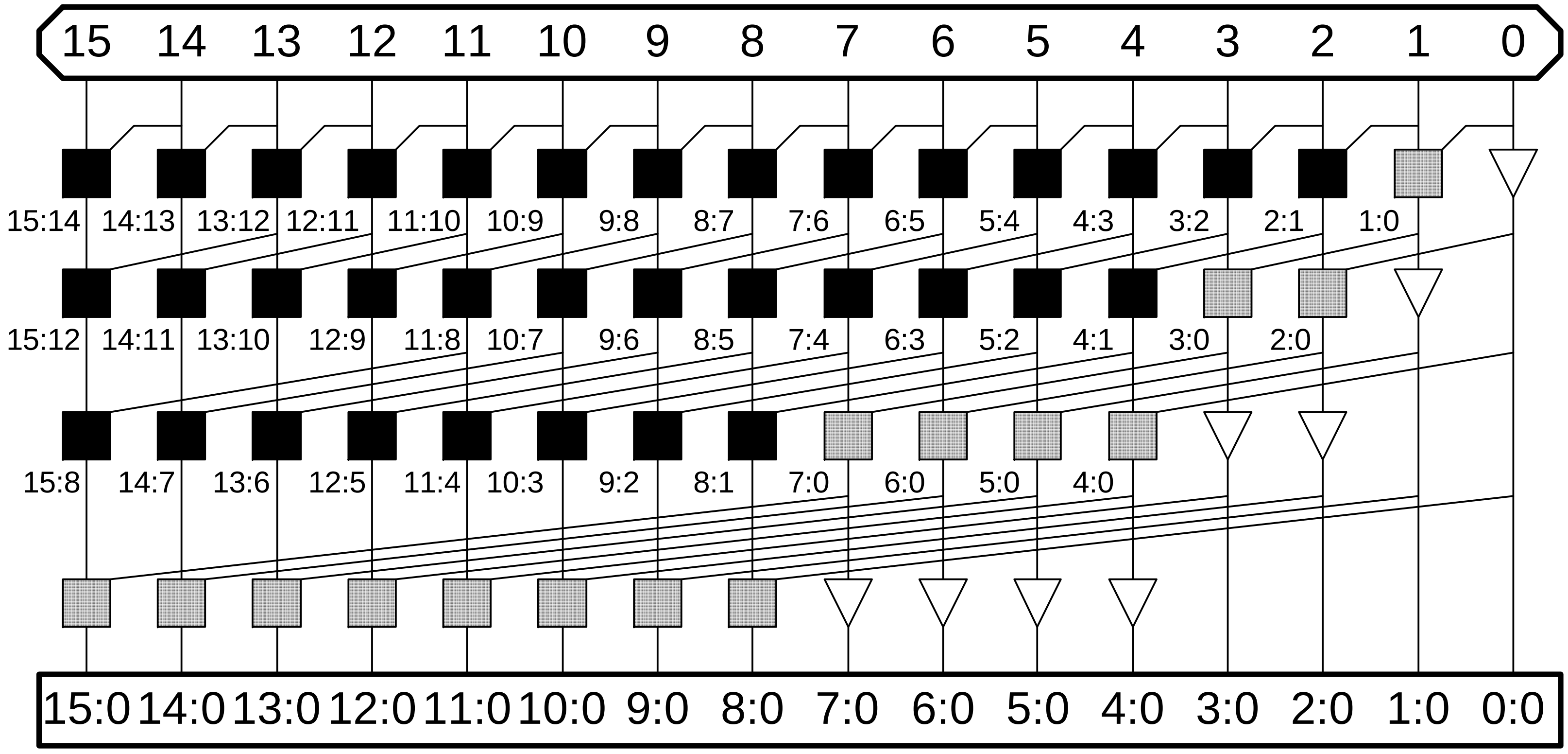

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14
13:12
11:10
9:8
7:6
5:4
3:2
1:0
15:12
11:8
7:4
3:0
15:8
7:0
11:0
13:0
9:0
5:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Sklansky



15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
15:14
13:12
11:10
9:8
7:6
5:4
3:2
1:0
15:12
14:12
11:8
10:8
7:4
6:4
3:0
2:0
15:8
14:8
13:8
12:8
15:0
14:0
13:0
12:0
11:0
10:0
9:0
8:0
7:0
6:0
5:0
4:0
3:0
2:0
1:0
0:0

Kogge-Stone

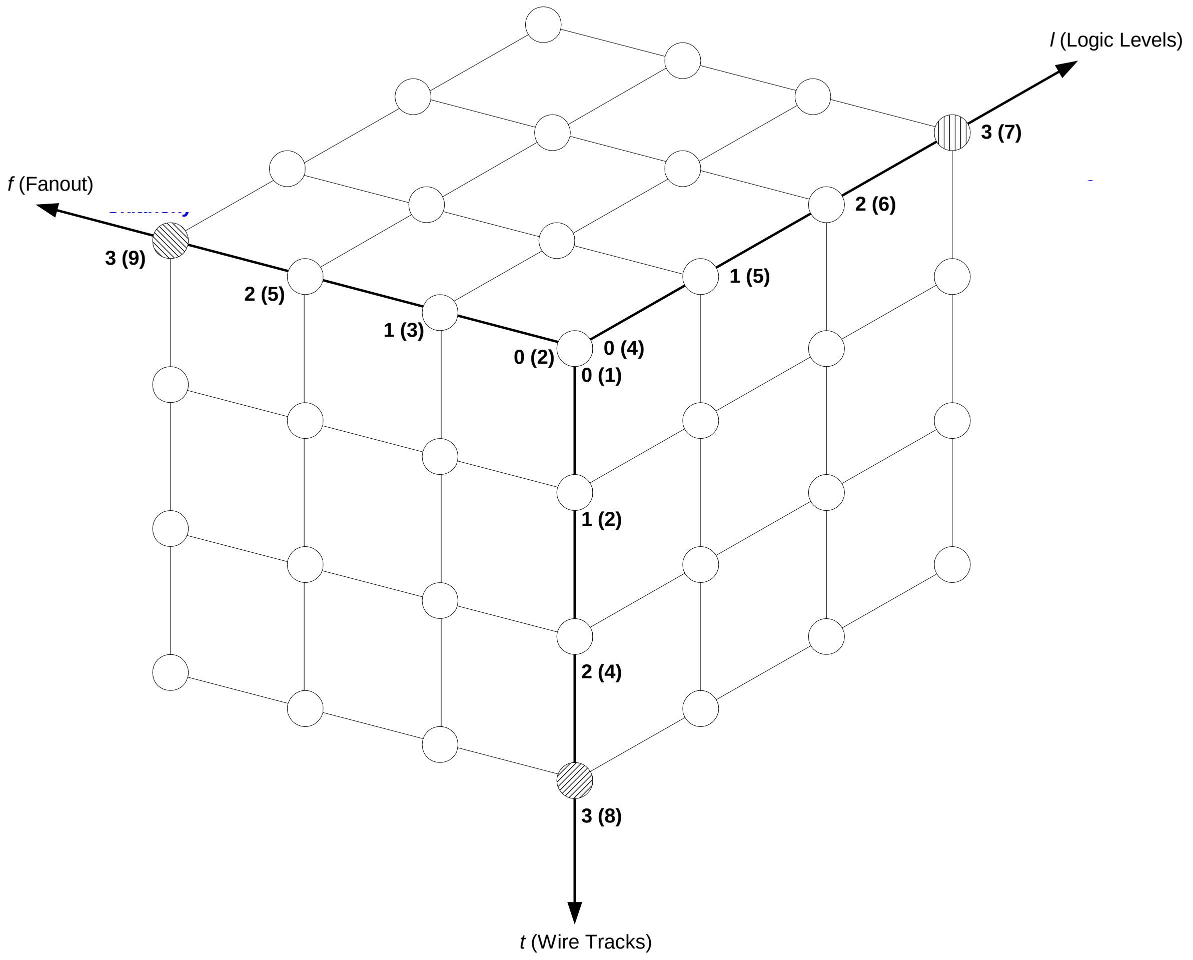

15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
15:14
14:13
13:12
12:11
11:10
10:9
9:8
8:7
7:6
6:5
5:4
4:3
3:2
2:1
1:0
15:12
14:11
13:10
12:9
11:8
10:7
9:6
8:5
7:4
6:3
5:2
4:1
3:0
2:0
15:8
14:7
13:6
12:5
11:4
10:3
9:2
8:1
7:0
6:0
5:0
4:0
15:0
14:0
13:0
12:0
11:0
10:0
9:0
8:0
7:0
6:0
5:0
4:0
3:0
2:0
1:0
0:0

Tree Adder Taxonomy

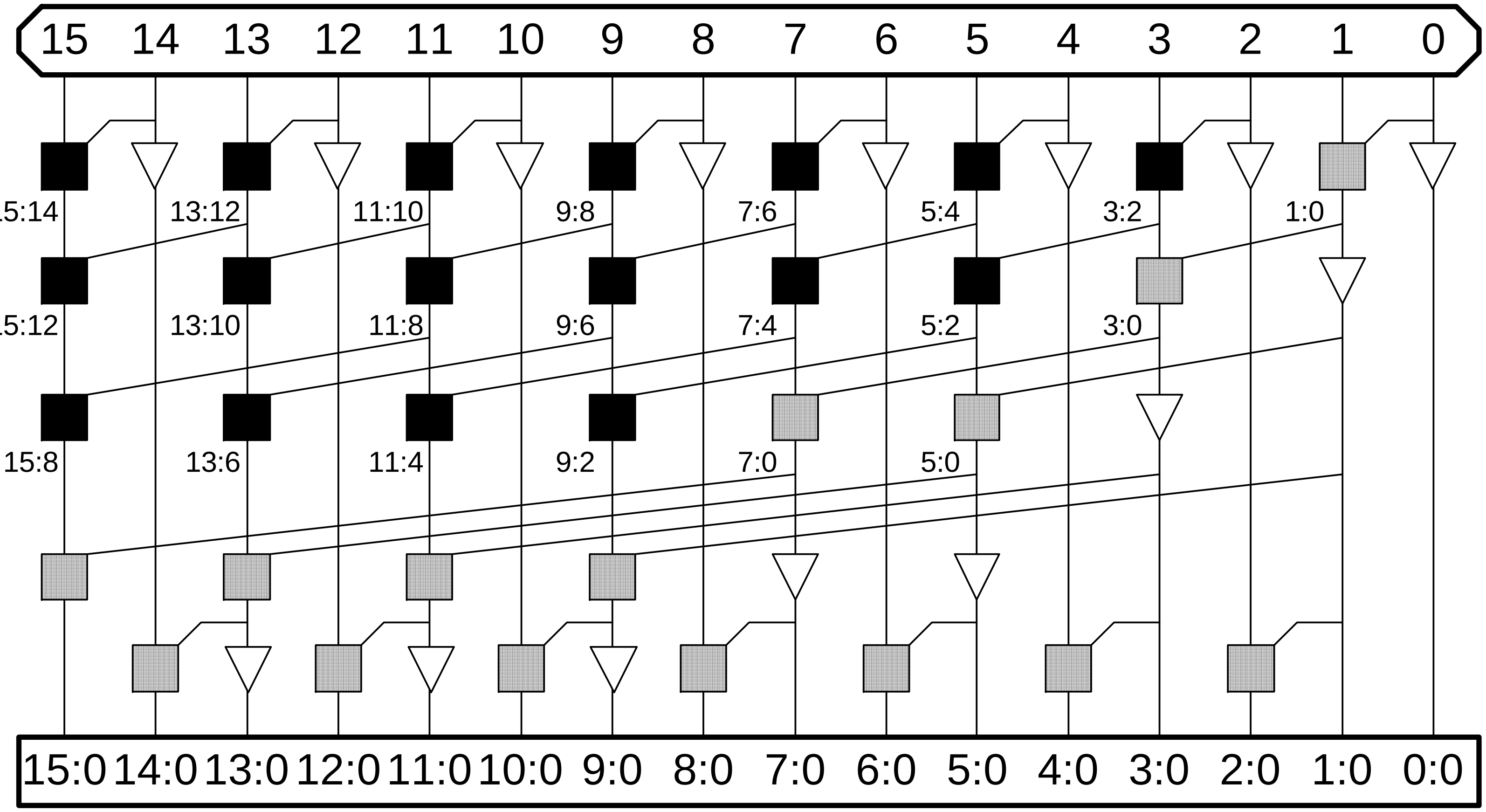
- Ideal N-bit tree adder would have
  - $L = \log N$ logic levels
  - Fanout never exceeding 2
  - No more than one wiring track between levels
- Describe adder with 3-D taxonomy (l, f, t)
  - Logic levels: $L + l$
  - Fanout: $2^f + 1$
  - Wiring tracks: 2^t
- Known tree adders sit on plane defined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

$$l + f + t = L - 1$$

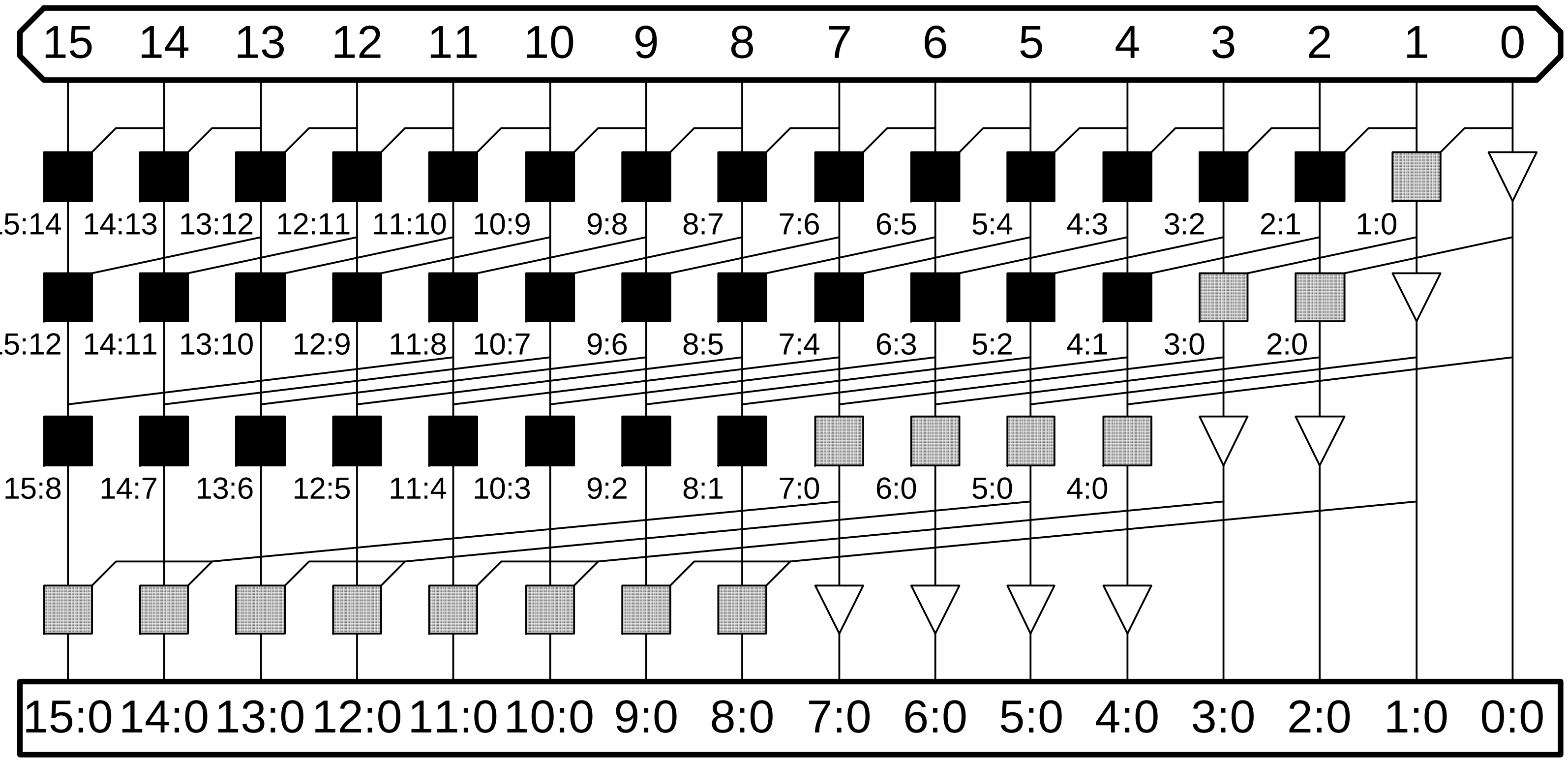
Tree Adder Taxonomy



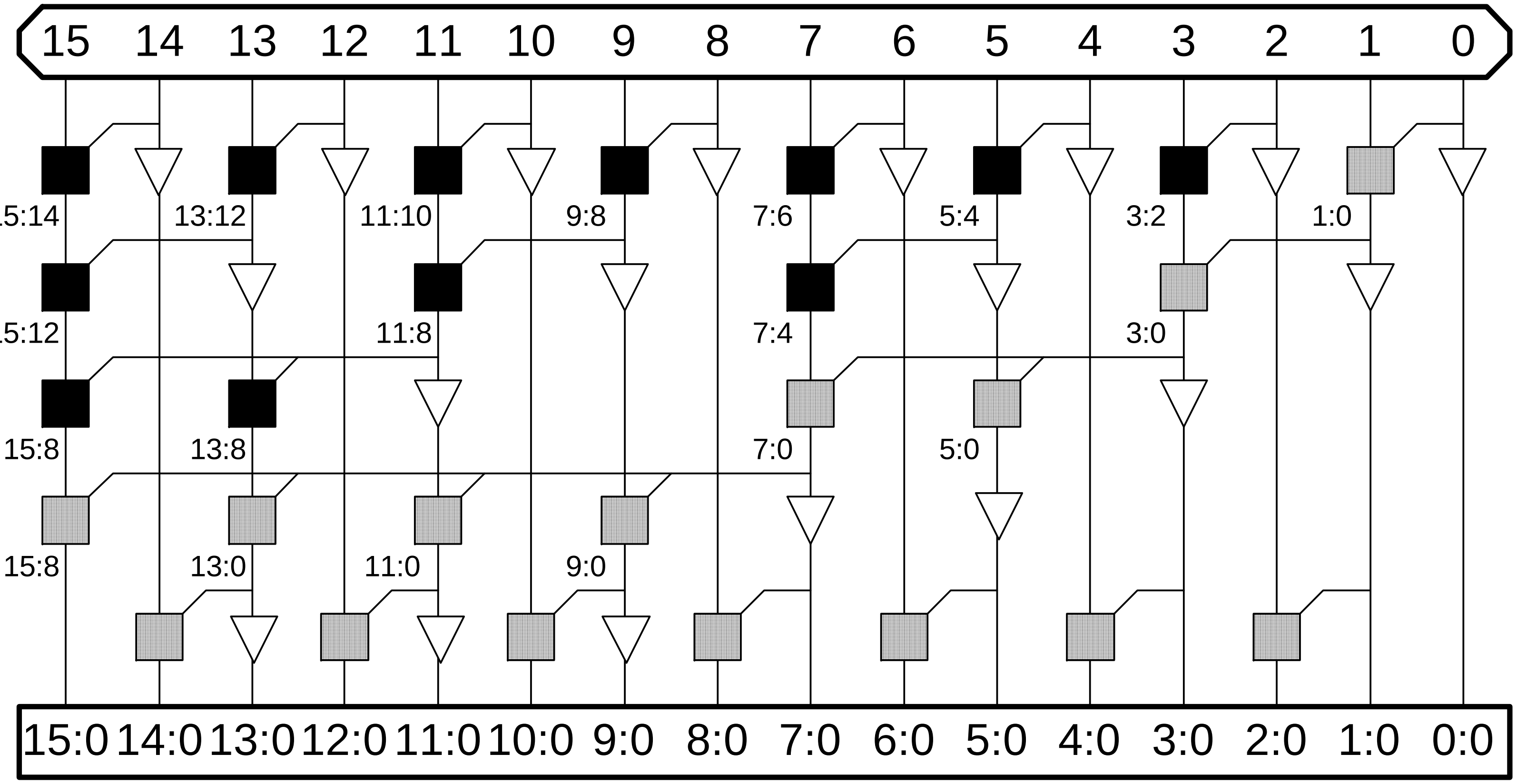
Han-Carlson


15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
15:14
13:12
11:10
9:8
7:6
5:4
3:2
1:0
15:12
13:10
11:8
9:6
7:4
5:2
3:0
15:8
13:6
11:4
9:2
7:0
5:0
15:0
14:0
13:0
12:0
11:0
10:0
9:0
8:0
7:0
6:0
5:0
4:0
3:0
2:0
1:0
0:0

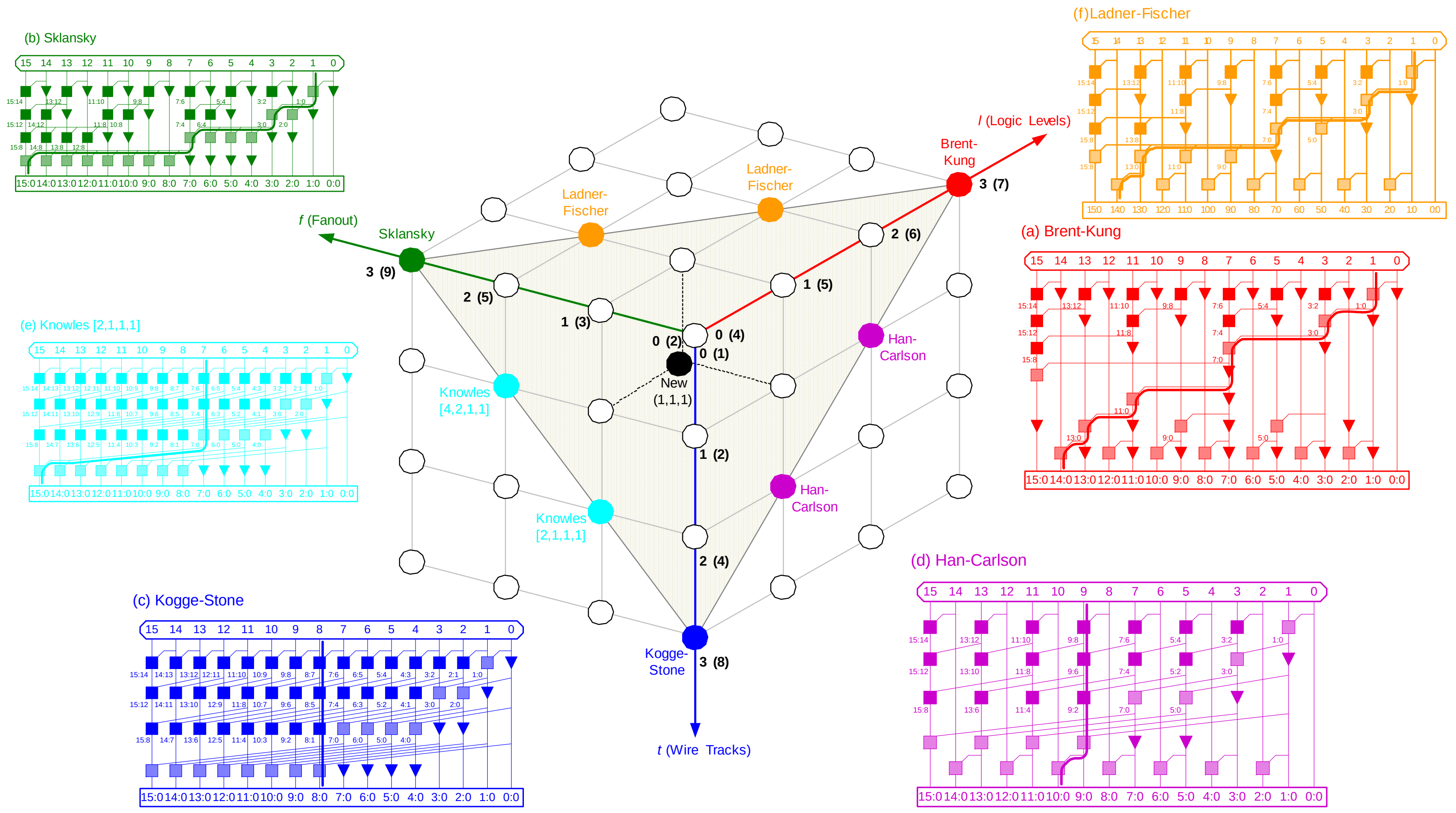
Knowles [2, 1, 1, 1]



Ladner-Fischer



Taxonomy Revisited



Summary

Adder architectures offer area / power / delay tradeoffs.

Choose the best one for your application.

Architecture	Classification	Logic Levels	Max Fanout	Tracks	Cells
Carry-Ripple		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	$N\log_2 N$