

CSEE 4823, Handout #17 “Advanced Adders”
Prof. Steven Nowick

revised version

Two-Operand Addition

Chapter two...

Excerpts from class notes of
Prof. Erik Brunvand
(used by courtesy)
Dept. of Computer Science
University of Utah

Prefix Adders (Tree Adders)

- ◆ More general form of carry lookahead tree
 - Built using different organizations of the same set of basic PG cells (PA cells)
 - All based on the fact that c_i corresponds to the generate signal spanning bit positions (-1) to $i-1$
 - Prefix adder is an interconnection of cells that produce $g_{(i-1,-1)}$ for all i
 - Cells connected to produce g signals that span an increasing number of bits

NOTE: Throughout this handout, assume “A” is the same as “P” (propagate).
(But... we will cover the real details later.)

PG (PA) cell

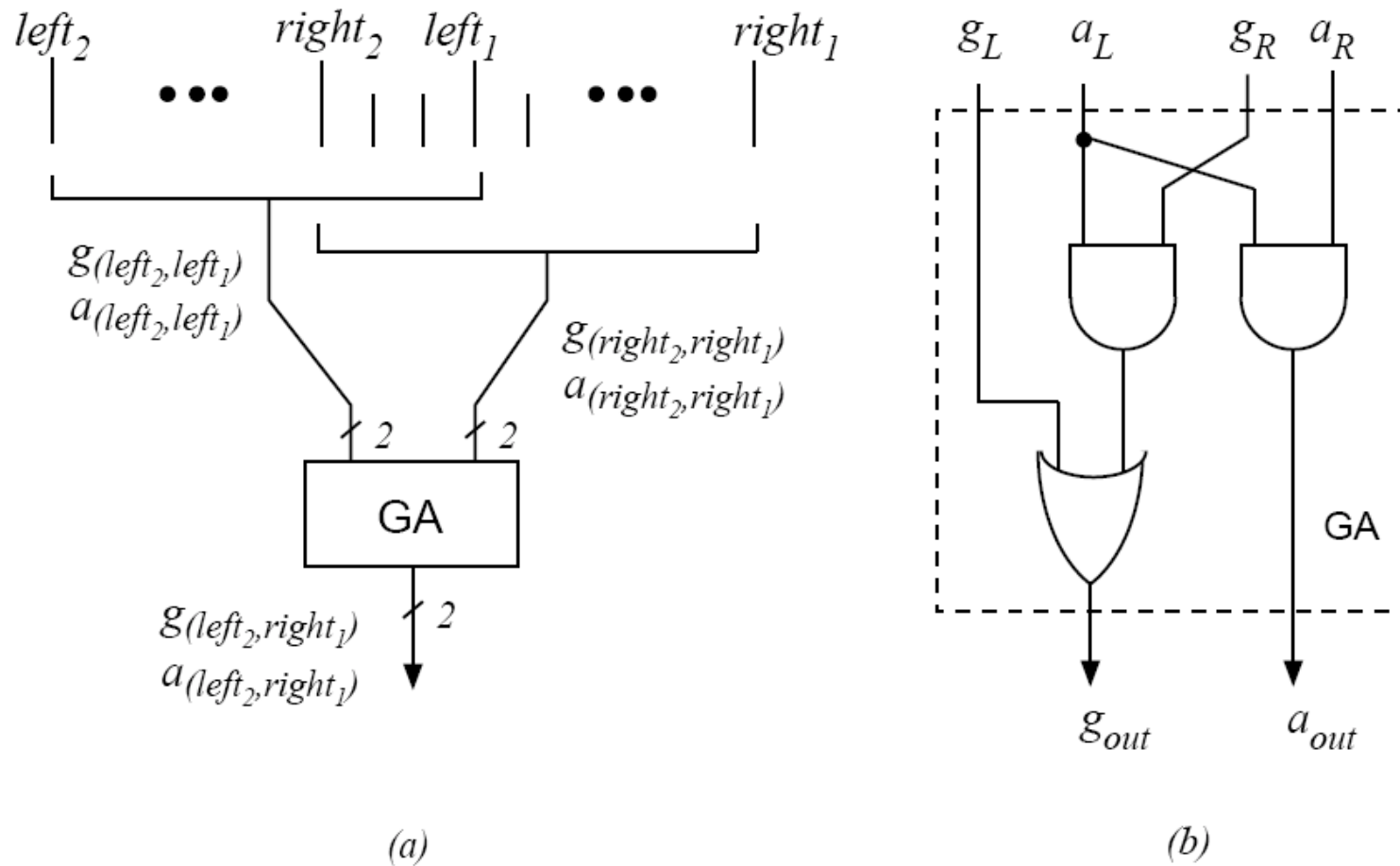
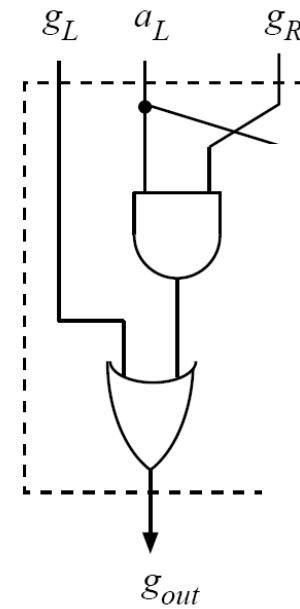
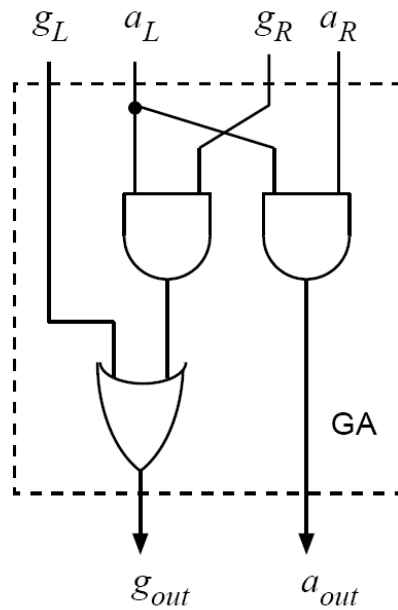
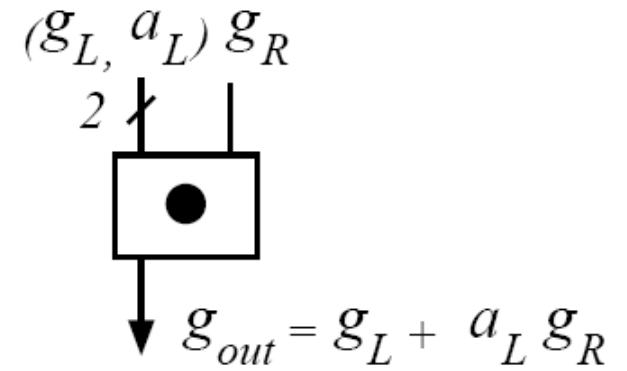
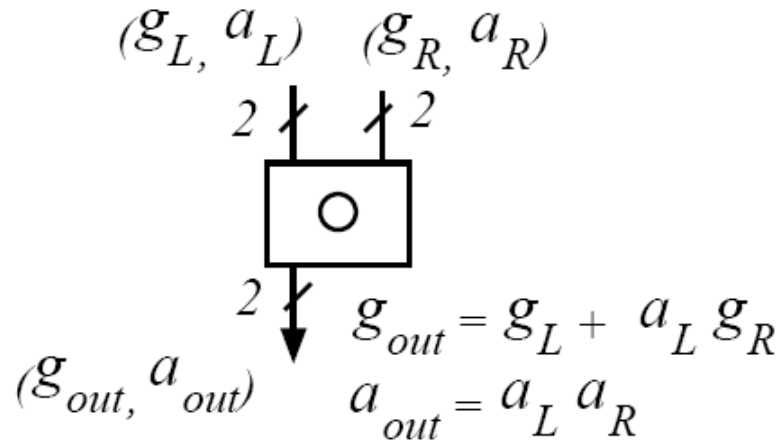


Figure 2.17: Composition of spans in computing (g, a) signals.

Overlapping Ranges

- ◆ Starting with g, a of each bit, first level generates g, a for two bits, then 4, etc.
 - If right input spans bits $[right2, right1]$, and left spans $[left2, left1]$, with $right2+1 \geq left1$
 - Then output spans bits $[left2, right1]$
 - For example $right[5,2]$ and $left[8,4]$ means output spans bits $[8,2]$

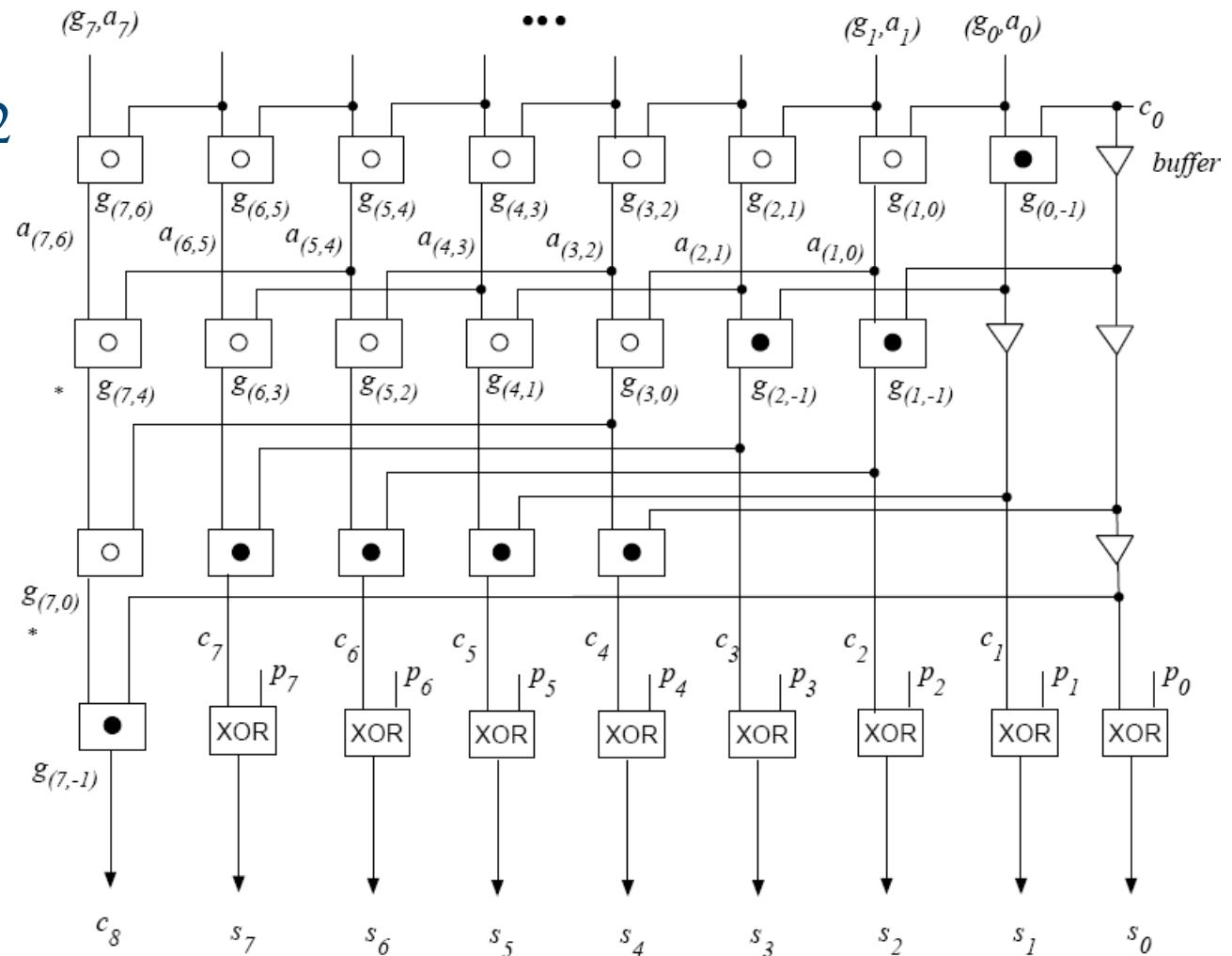
PG (PA) Cells



8-bit prefix adder

Max fanout 2
Min levels

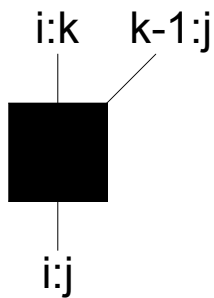
Kogge-Stone adder
(includes carryin c_0 ! [also called g_{-1}])



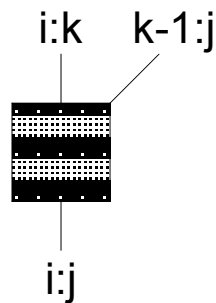
* $a_{(i,k)}$ not labeled

Another View of Prefix Adders

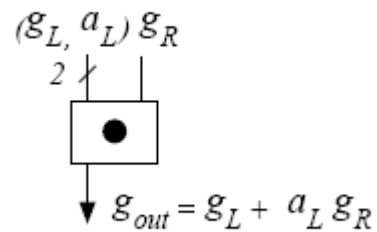
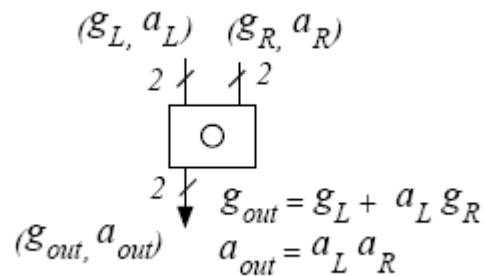
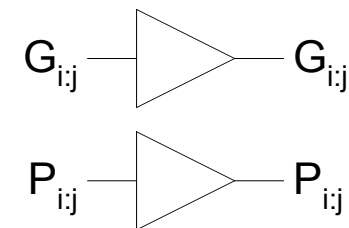
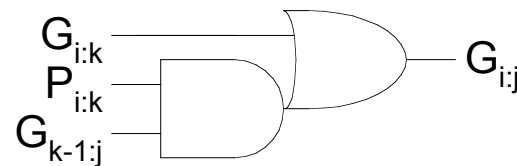
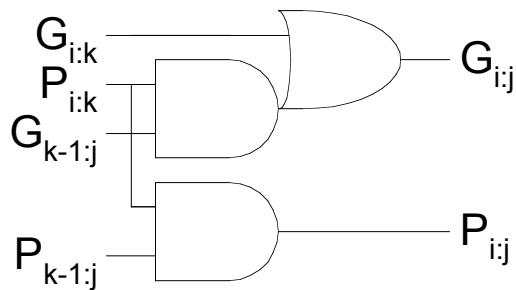
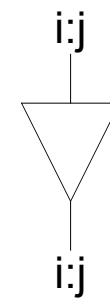
Black cell



Gray cell



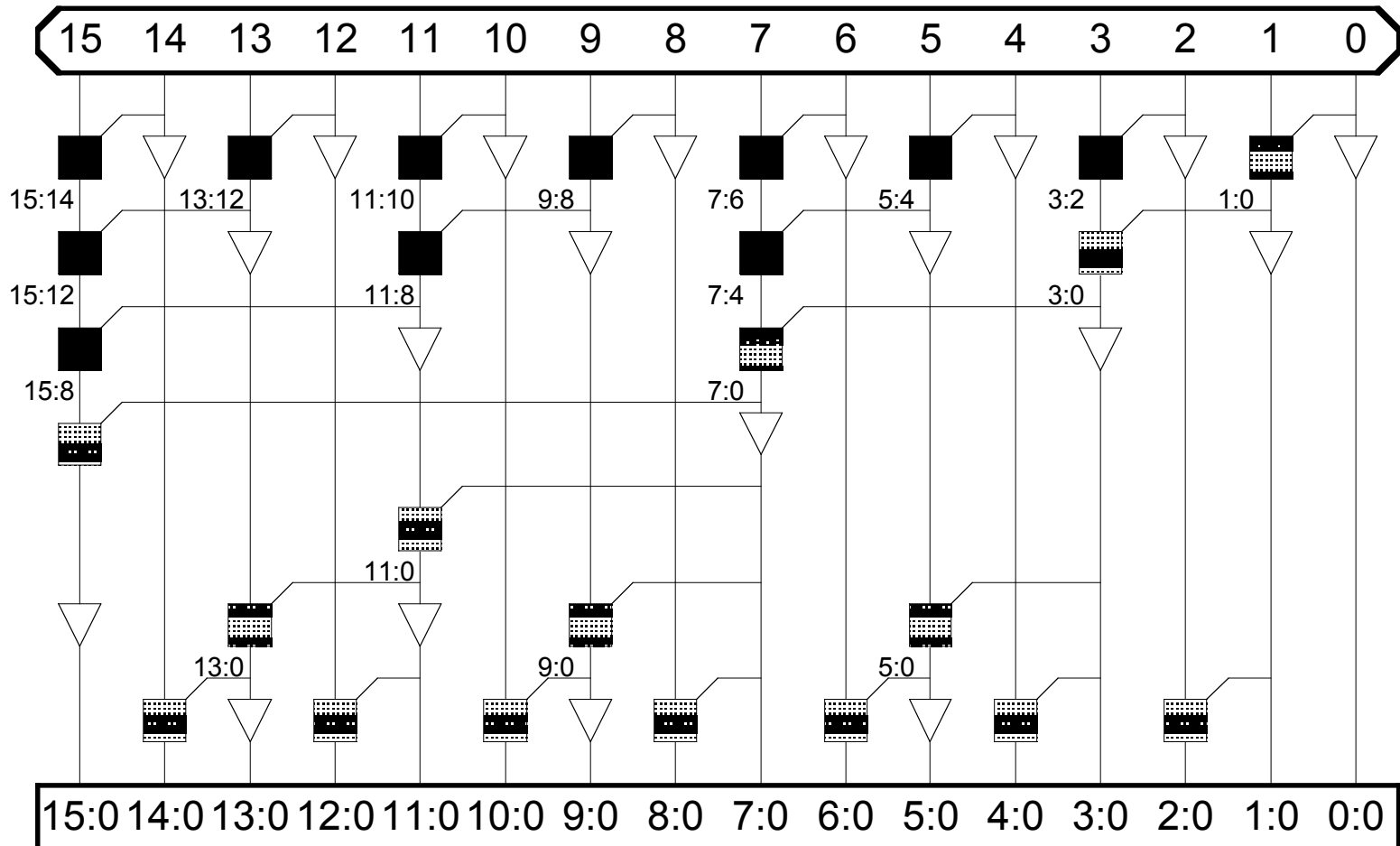
Buffer



Brent-Kung

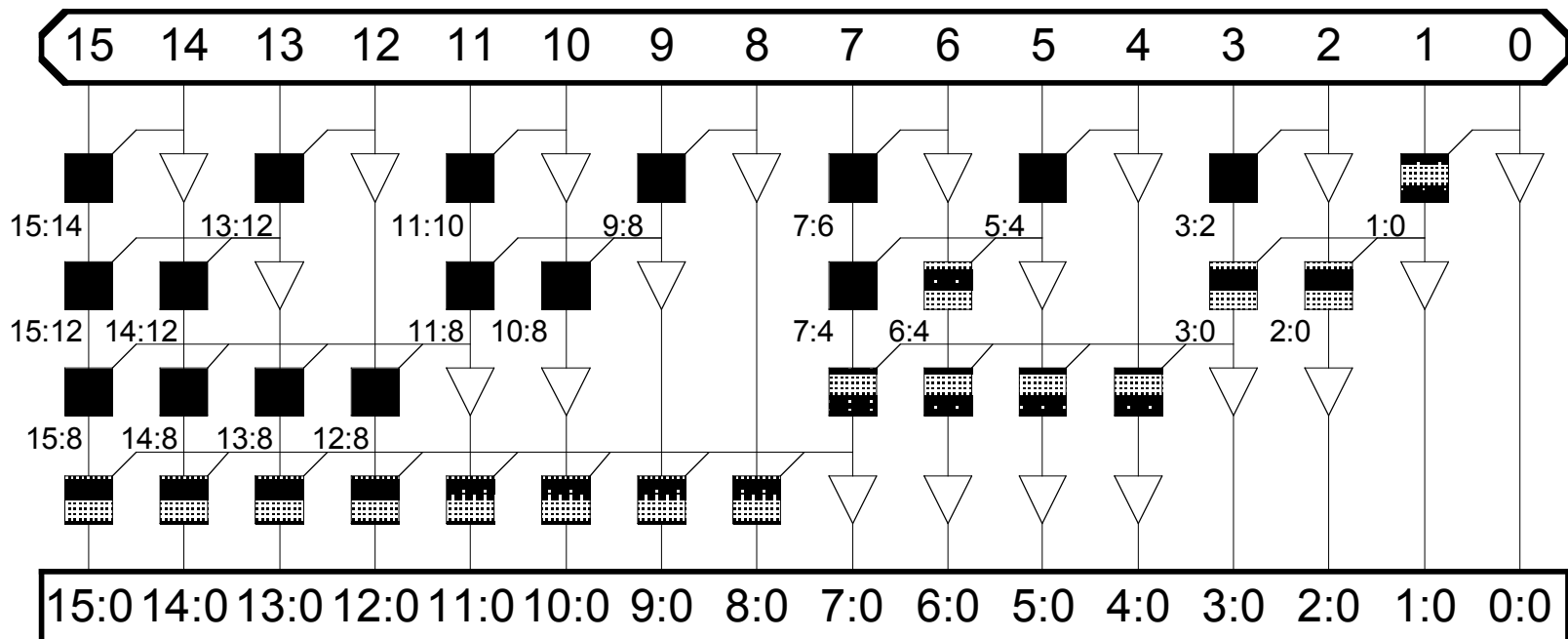
NOTE: these slides
from David Harris now
assume no carryin c0!
(but can modify to add c0)

7

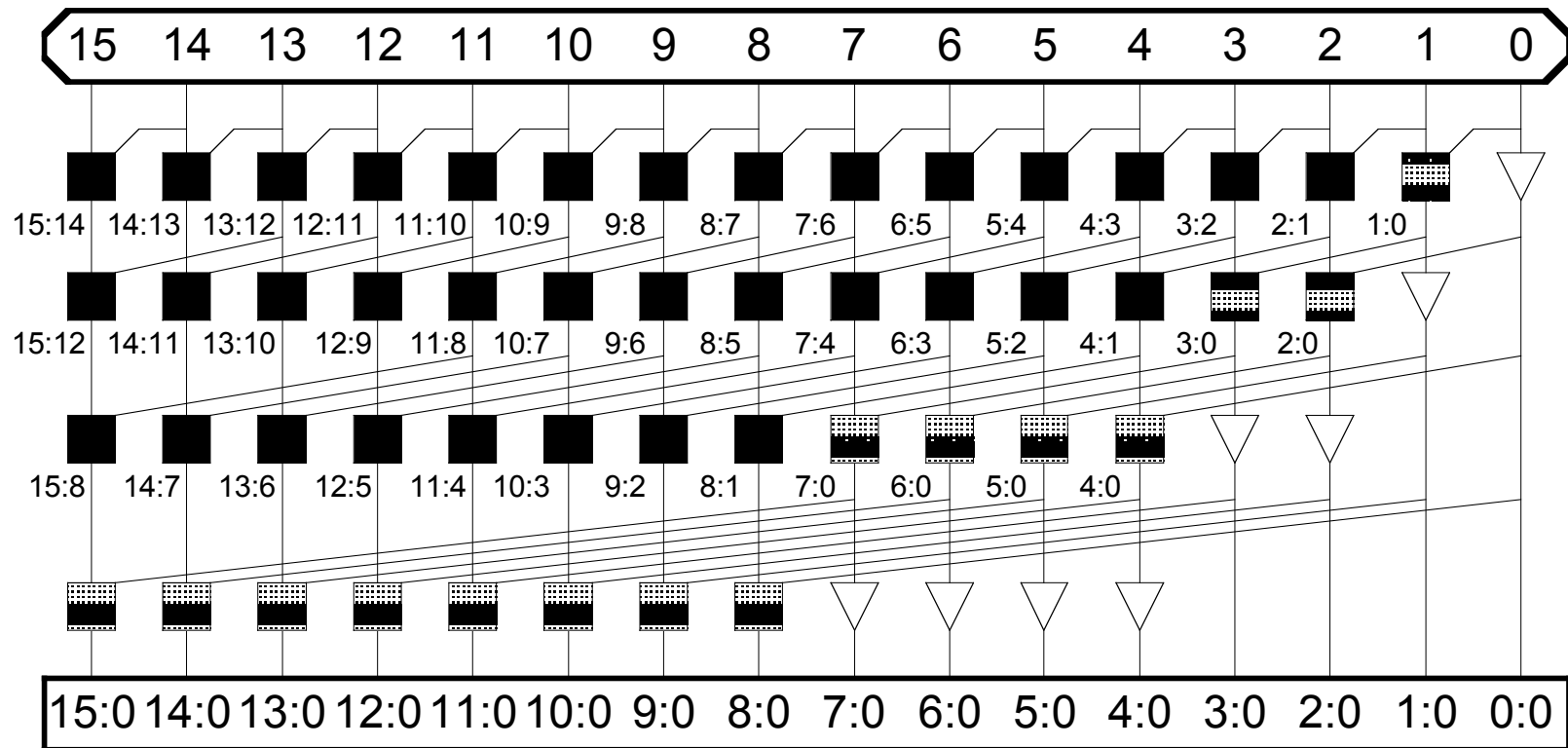


David Harris, Harvey Mudd

Sklansky Adder



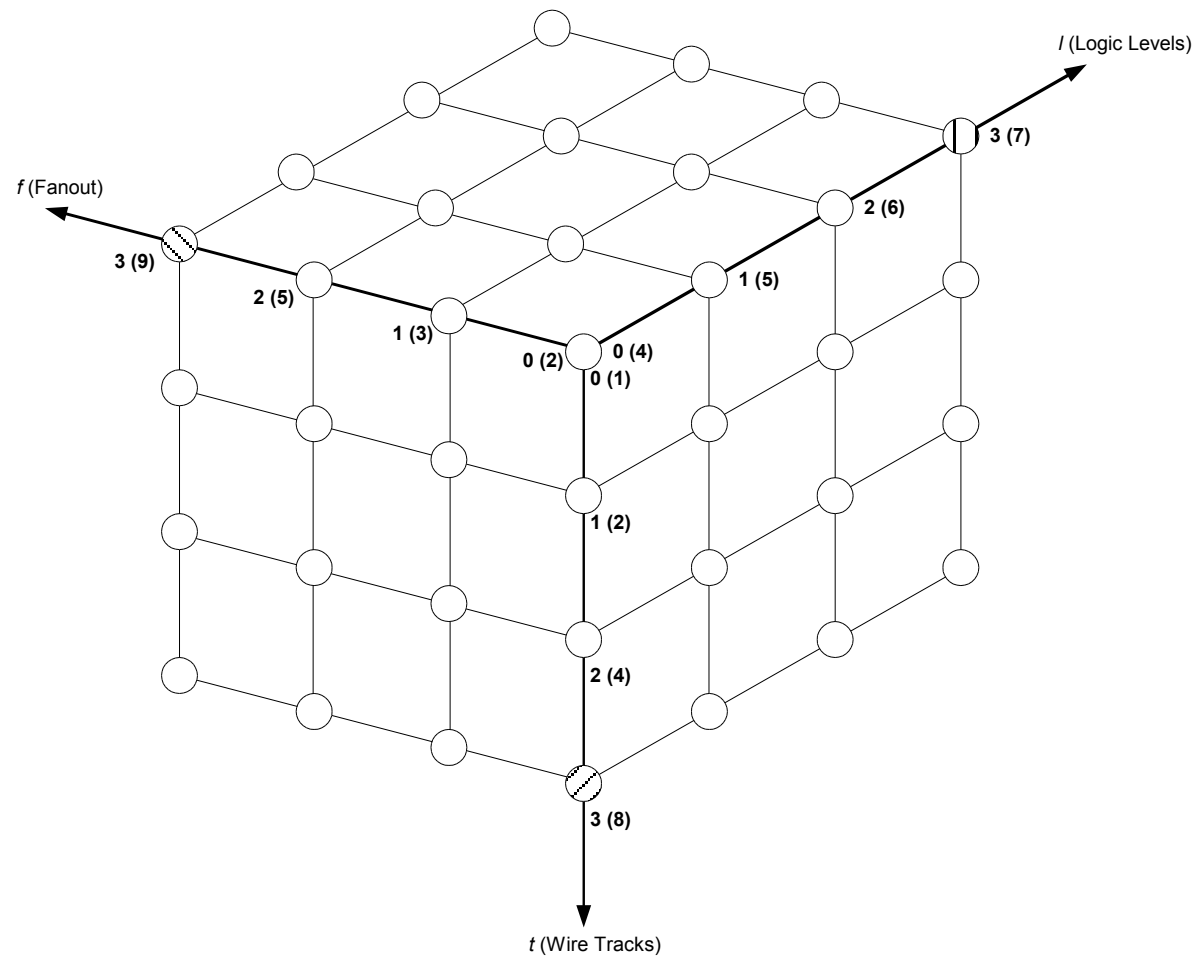
Kogge-Stone Adder



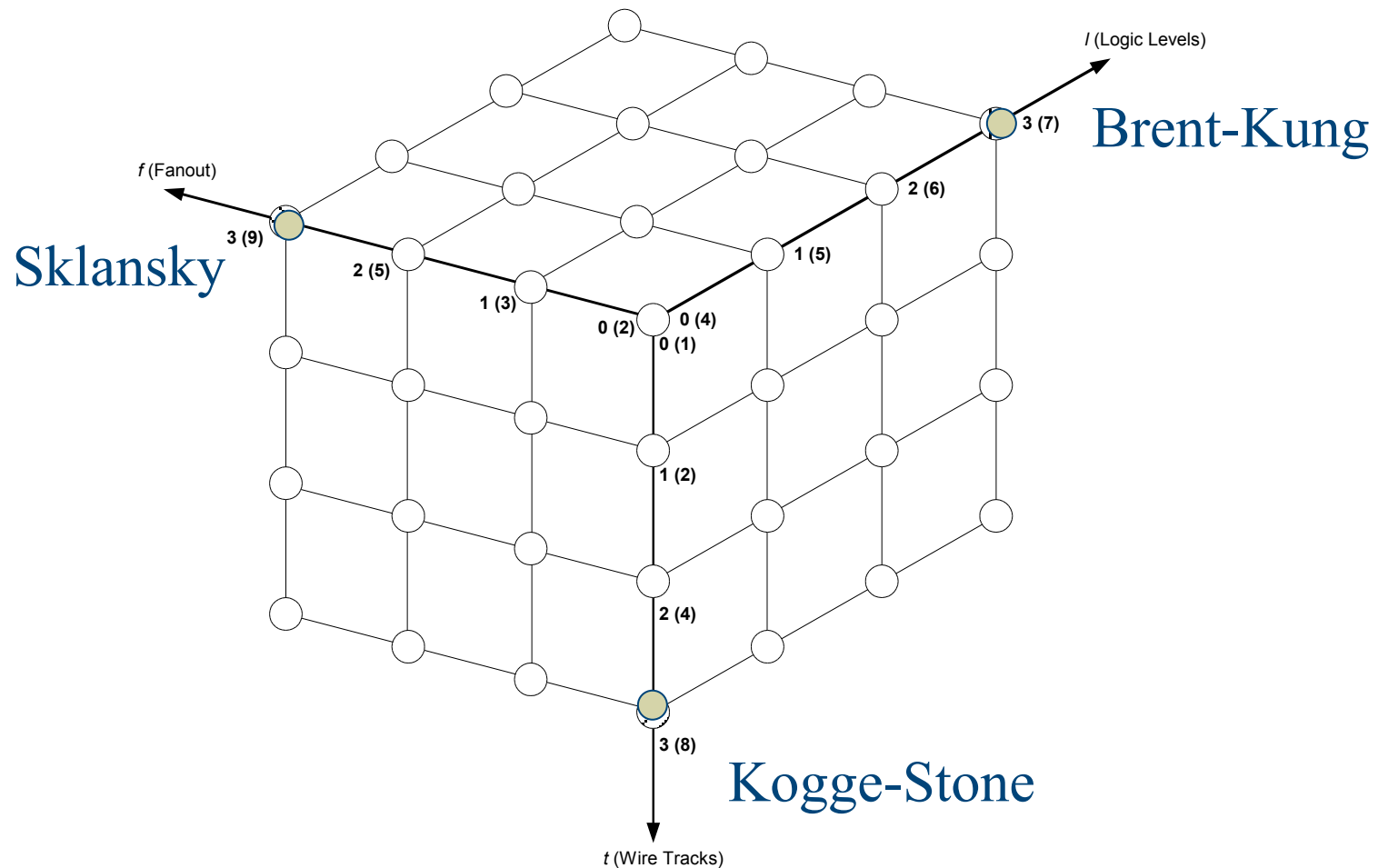
Tree Adder Taxonomy

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

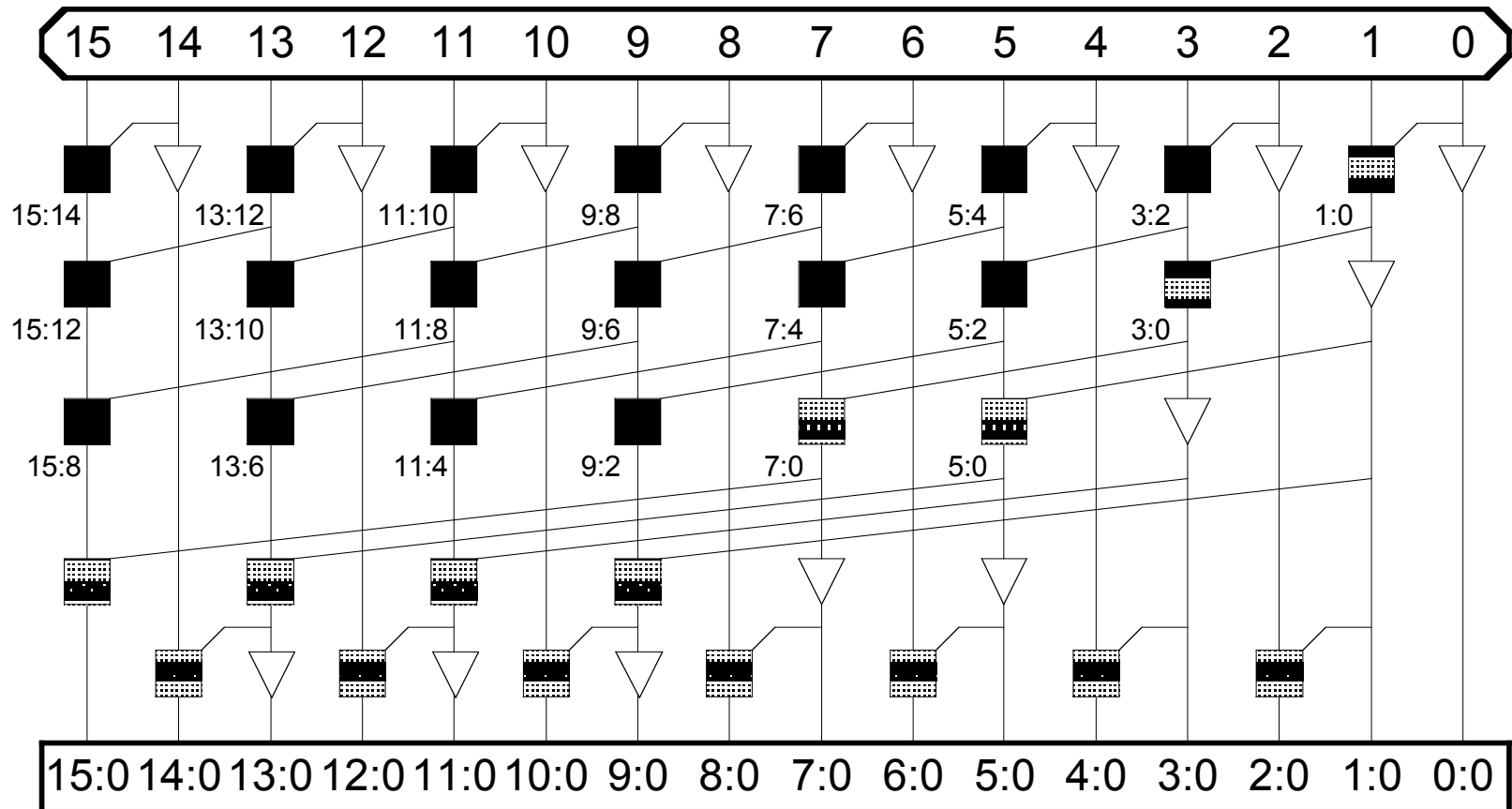
Tree Adder Taxonomy



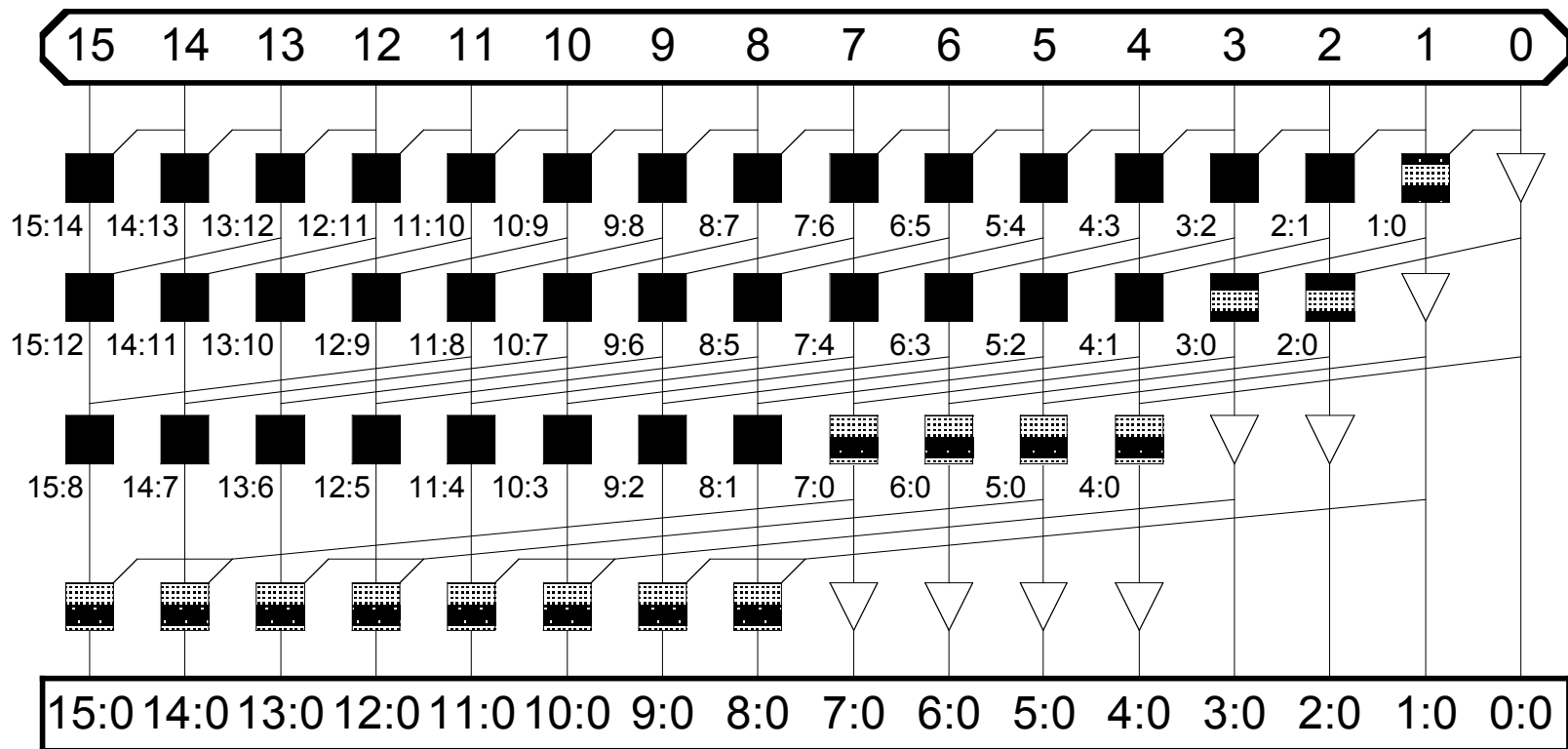
Tree Adder Taxonomy



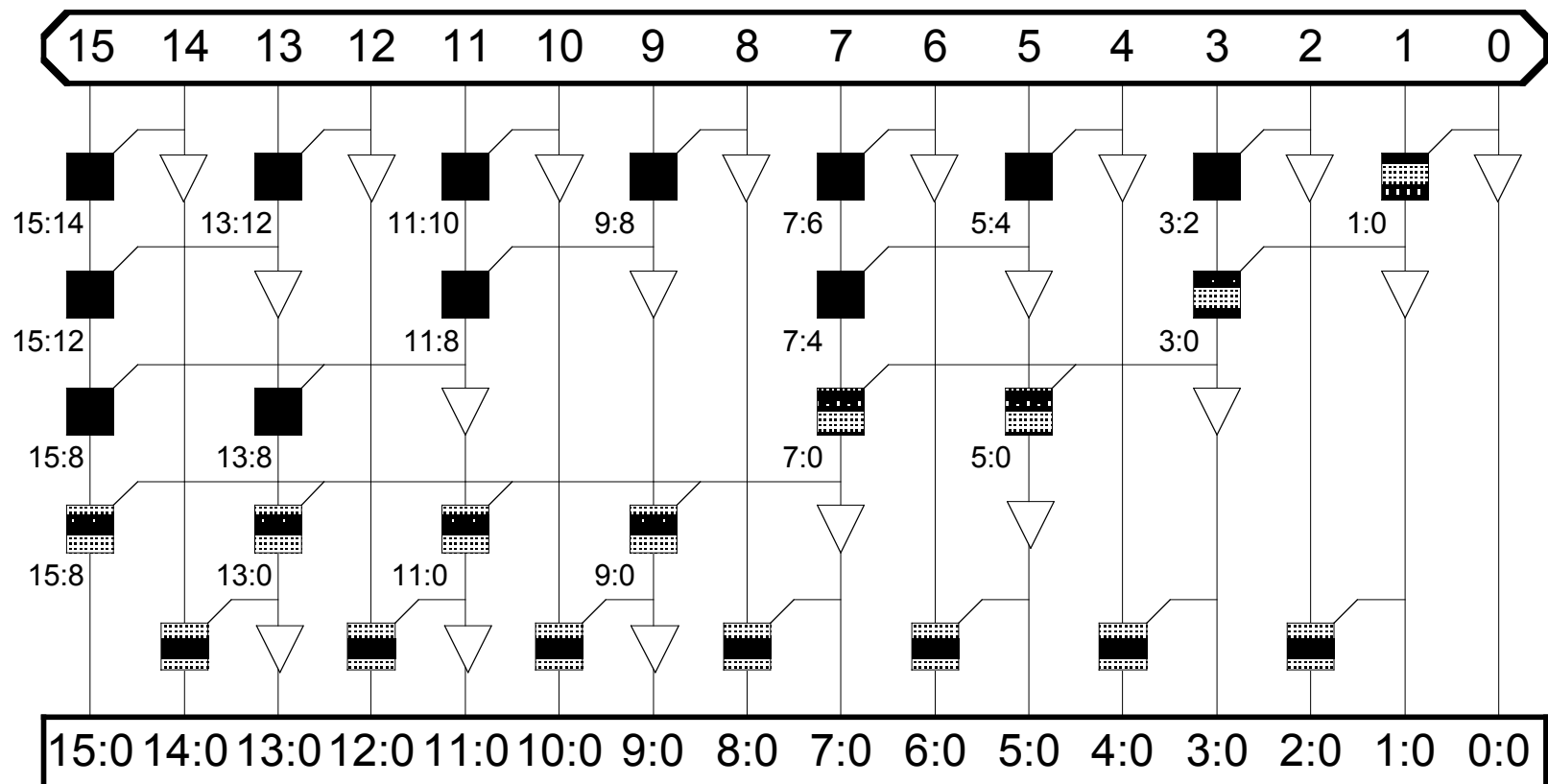
Han-Carlson



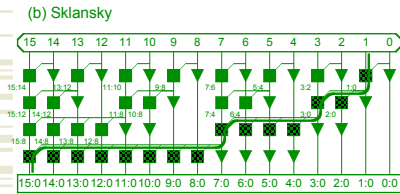
Knowles [2, 1, 1, 1]



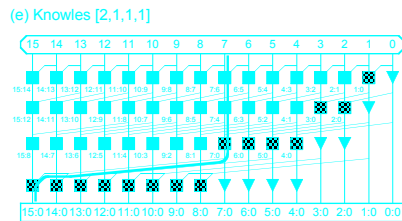
Ladner-Fischer



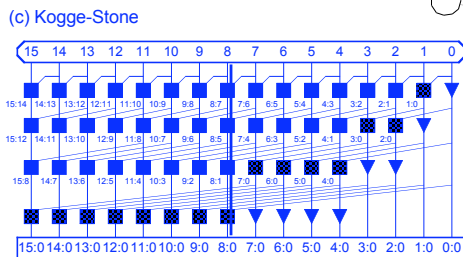
Taxonomy Revisited



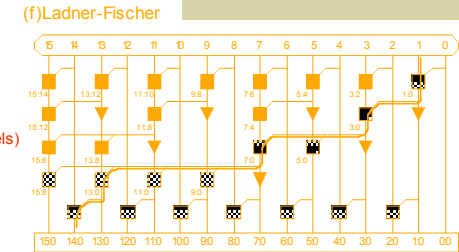
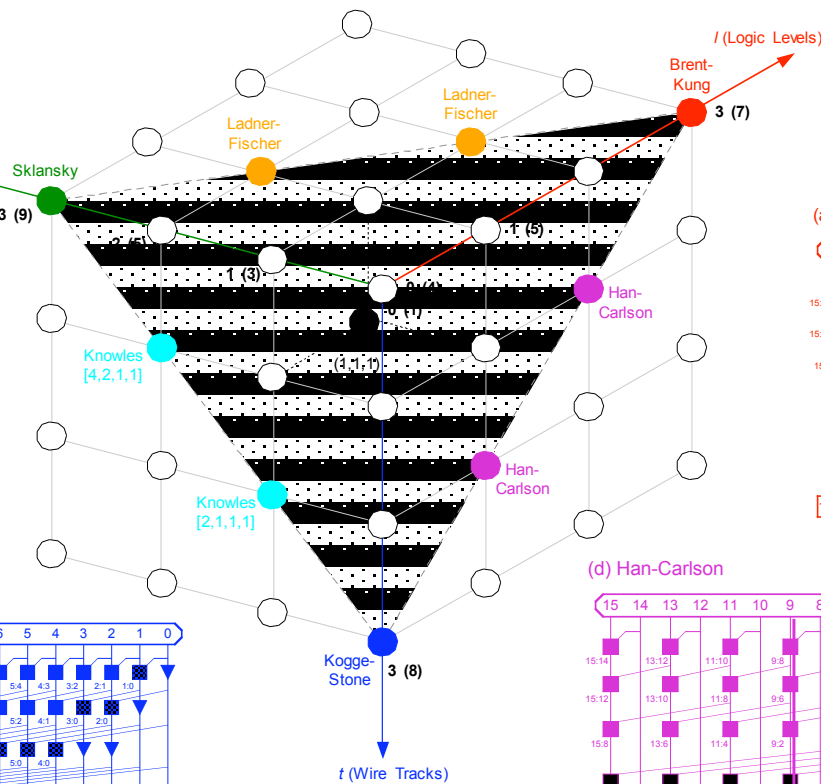
Sklansky f (Fanout)



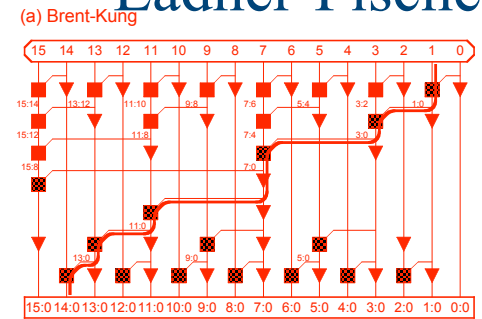
Knowles



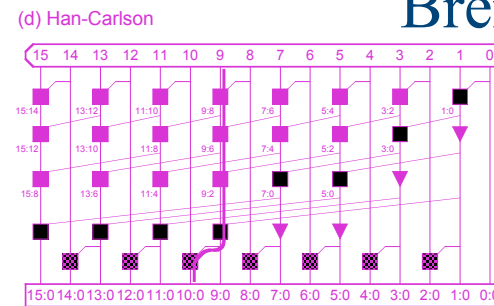
Kogge-Stone



Ladner-Fischer



Brent-Kung



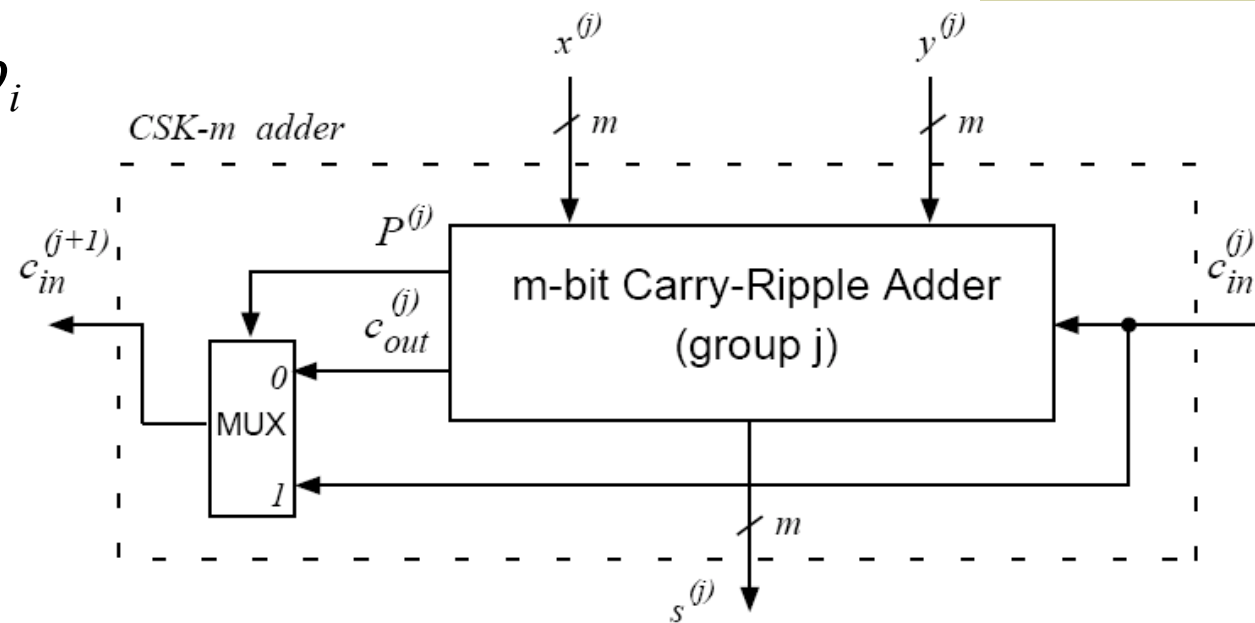
Han-Carlson
David Harris, Harvey Mudd

Carry Skip Adder

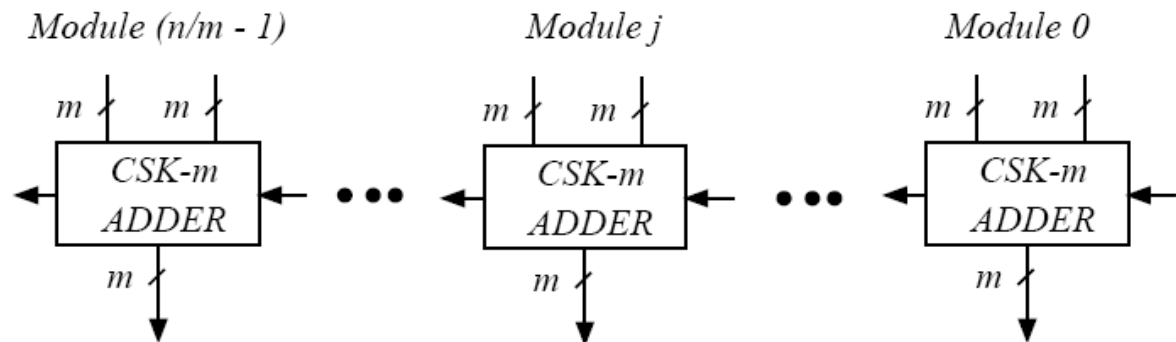
- ◆ The idea is to reduce the number of cells the worst-case carry must propagate through
 - Divide n -bit adder into groups of m -bits
 - Determine group propagate for each m -bits
 - If the entire group p is true, skip around it

Carry Skip Adder

$$p^{(j)} = \text{AND}_{i=0}^{m-1} p_i$$



(a)



Carry Skip example

=p
 =k
 =g

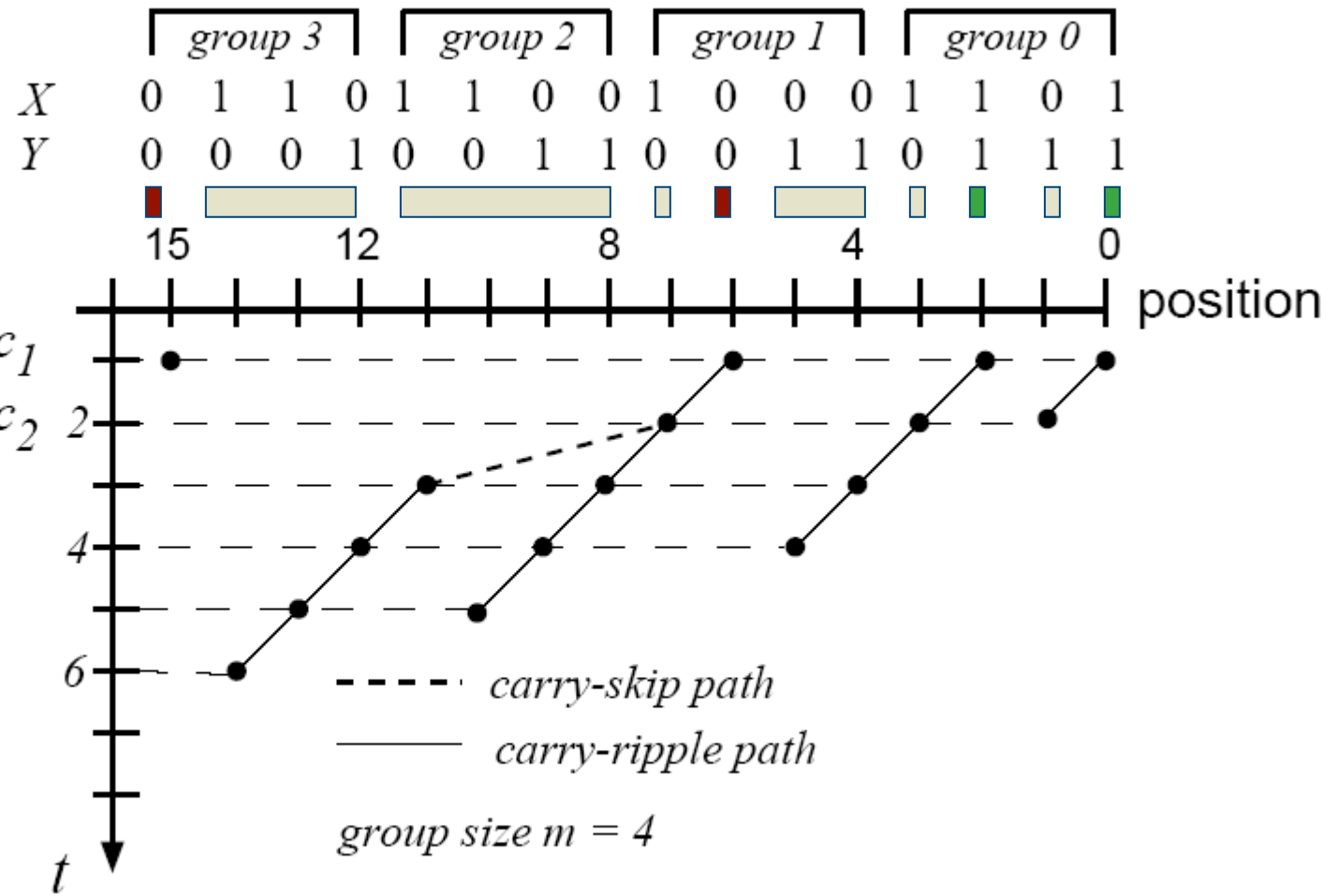
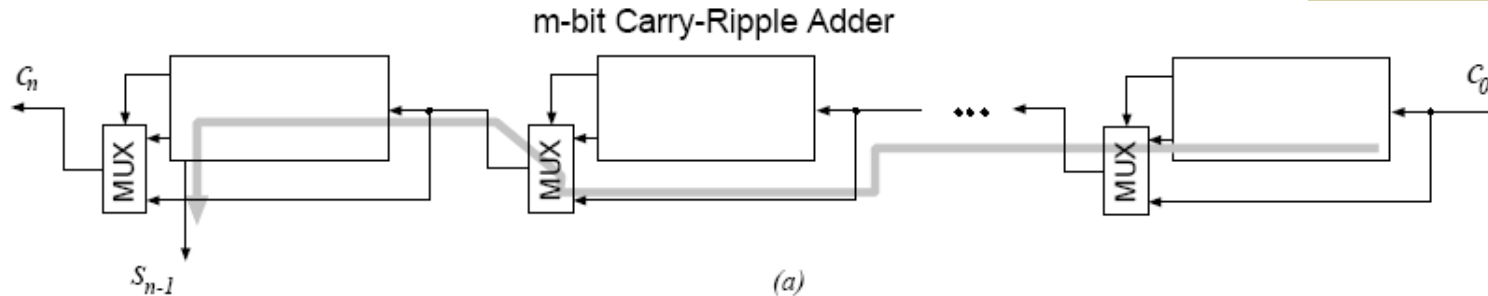
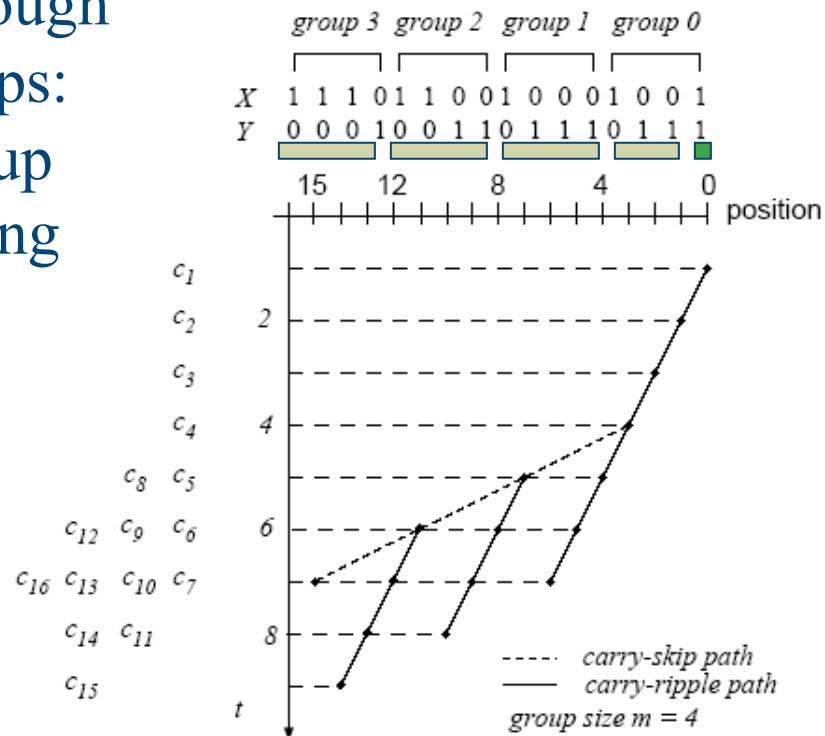


Figure 2.8: Carry chains in carry-skip adder: A case with several carry chains.

Carry Skip worst case



Carry travels through at most two groups: the initiating group and the terminating group.



Carry Skip delay

$$\begin{aligned} T_{CSK} &= mt_c + t_{mux} + \left(\frac{n}{m} - 2\right)t_{mux} + (m - 1)t_c + t_s \\ &= (2m - 1)t_c + \left(\frac{n}{m} - 1\right)t_{mux} + t_s \end{aligned}$$

- ◆ Worst case is when a carry is generated in the first bit of the adder
 - Then propagated through all bits up to but not including the high order bit
 - That is, skip all groups but the first and last

Problem with clearing carries

- ◆ Watch out – some books show an AND/OR version that doesn't really work!
 - Problem is that carries might be left over from previous addition and have to dribble out...

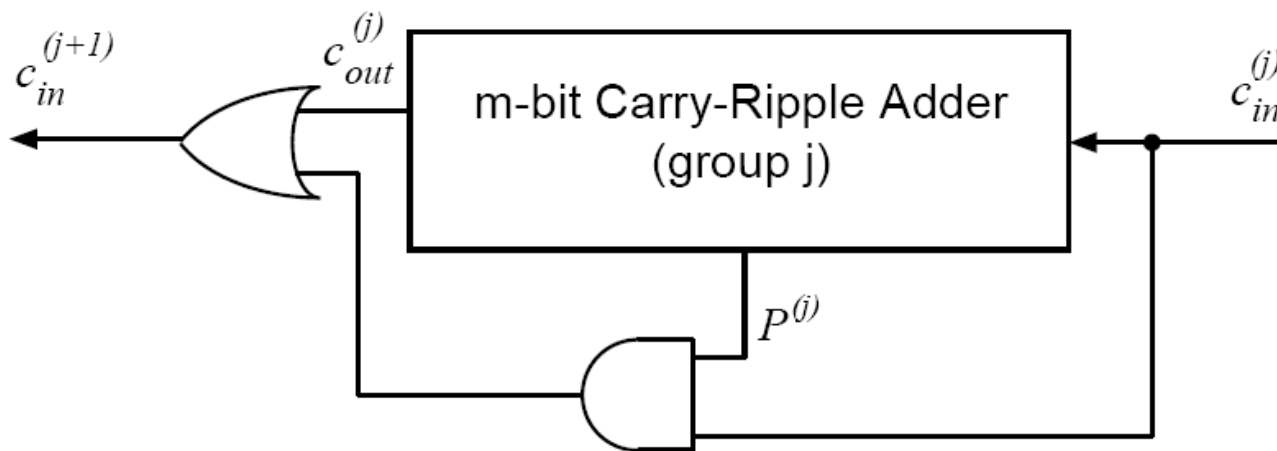


Figure 2.10: Carry-skip adder using AND-OR for bypass

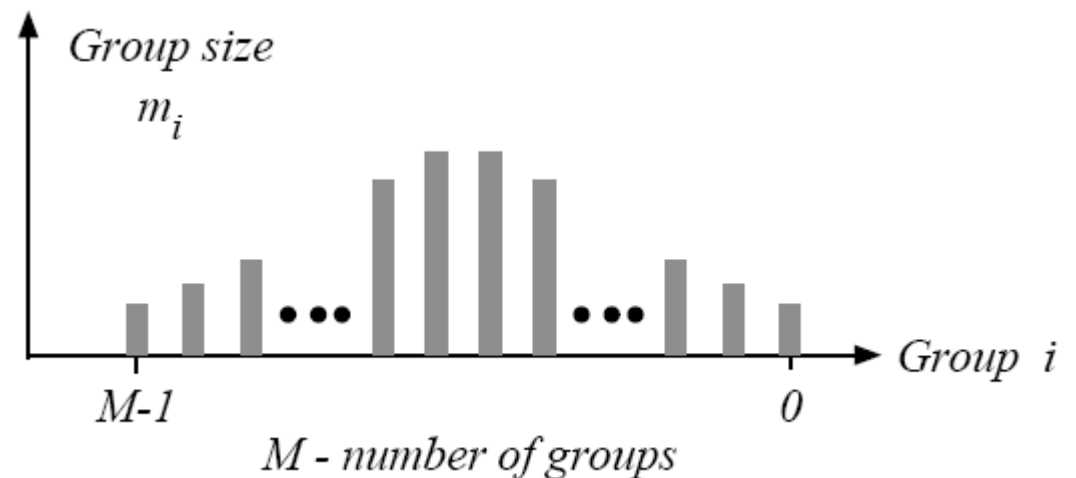
Group Size

- ◆ Previous delay analysis assumes all groups are the same size
 - This isn't the best for speed...
 - Carries generated in the first group have to skip more groups!
 - For fixed size:

$$m_{opt} = \left(\frac{t_{mux}n}{2t_c}\right)^{1/2} \quad (\text{minimum delay})$$
$$T_{opt} \approx (8t_{mux}t_cn)^{1/2}$$

Carry Skip with different m

- ◆ If you vary the group size with the groups at the ends shorter than the groups in the middle, you can speed things up



- $N=60, t_c=t_s=t_{\text{mux}}=\delta$
- $M=6, T_{\text{CSK}}=21 \delta$
- $M=4,5,6,7,8,8,7,6,5,4, T_{\text{CSK}}=17 \delta$

Summary

Scheme	Delay proportional to	Area proportional to
Linear structures:		
Carry ripple	n	n
Carry lookahead (one level)	n/m	$(k_m m)(n/m) = k_m n$
Carry select (one level)	n/m	$(k_m m)(n/m) = k_m n$
Carry skip (one level)	\sqrt{n}	n
Logarithmic structures:		
Carry lookahead (max. levels)	$2 \log_m n$	$(k_m m)(n/m) = k_m n$
Prefix	$\log_m n$	$((k_m m) \log_m n)n$
Conditional sum	$\log_2(n/m)$	$(k_m + \log_2(n/m))n$
Completion signal (avg. delay)	$(\log_2 n)/m$	$k_m m(n/m) = k_m n$
Redundant	<i>const.</i>	n

Case Study

- ◆ Dec Alpha 21064 64-bit adder
 - 5ns cycle time in a 0.75 μ CMOS process
 - Very high performance for the day!
 - A mix of multiple techniques!

Alpha 21064

- ◆ In 8-bit chunks – Manchester carry chain
 - Chain was also tapered to reduce the load caused by the remainder of the chain
 - Chain was pre-discharged at start of cycle
 - Three signals used: P, G, and K
 - Two Manchester chains:
 - One assuming $C_{in}=0$
 - One assuming $C_{in}=1$

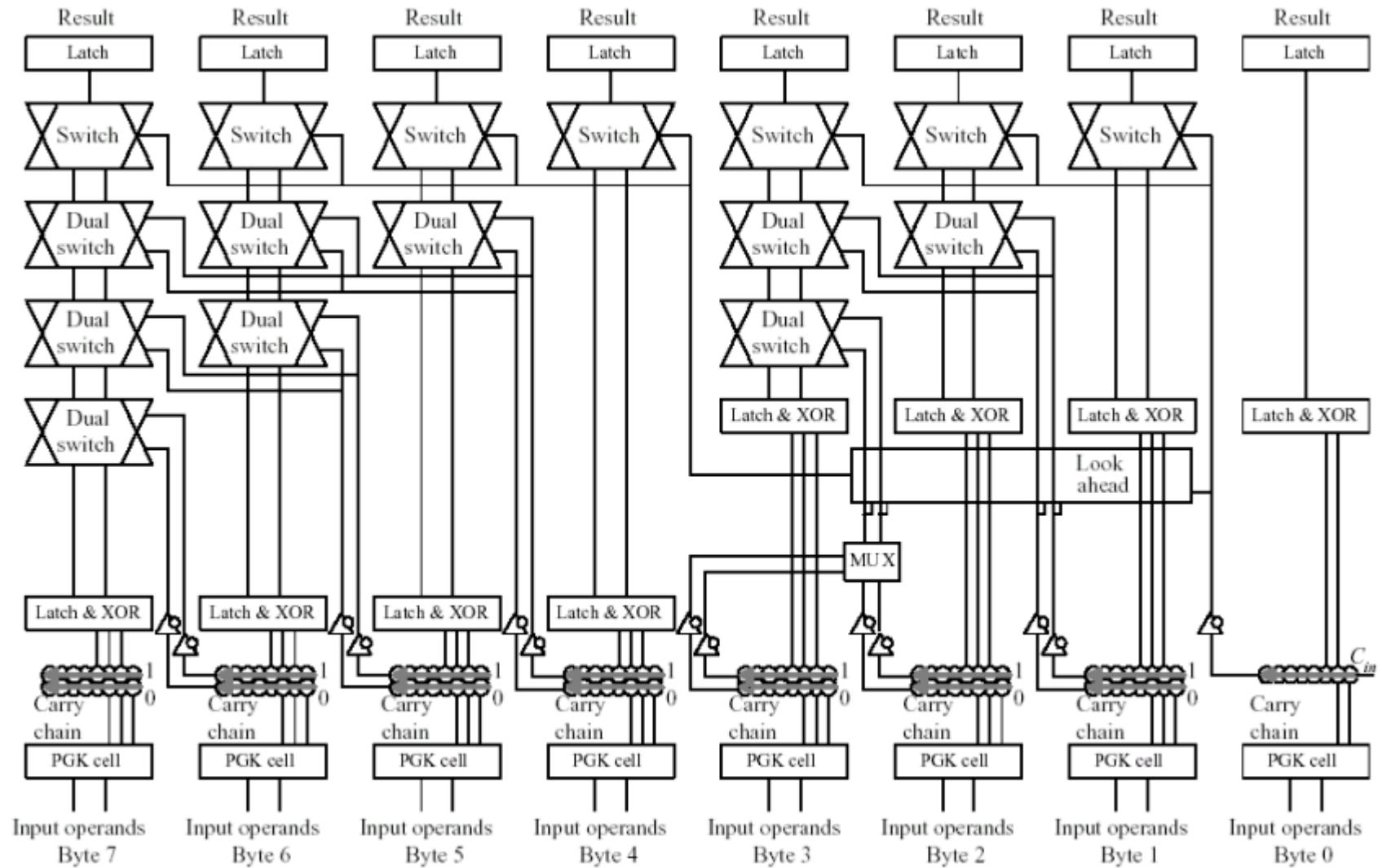
Alpha 21064

- ◆ Carry Lookahead used on least significant 32 bits
 - Implemented as distributed differential circuits
 - Provide carry that controls most significant 32
- ◆ Conditional Sum used for most significant 32
 - Six 8-bit select switches used to implement conditional sum on the 8-bit level

Alpha 21064

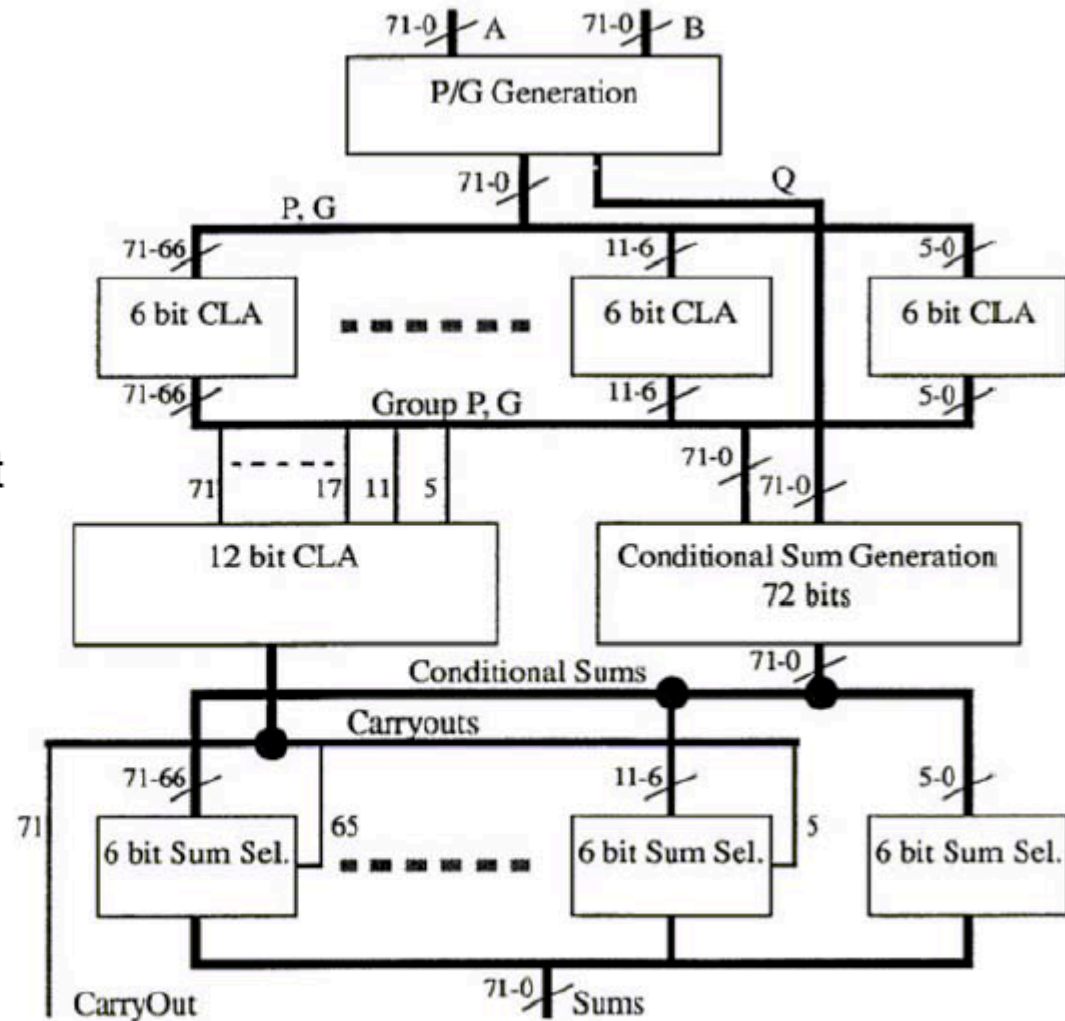
- ◆ Finally, Carry Select used to produce the most significant 32 bits.
 - Final selection done using NMOS carry-select byte-wide muxes
- ◆ Also apparently pipelined with a row of latches after the lookahead...

Alpha 21064



72-bit Pentium II Adder

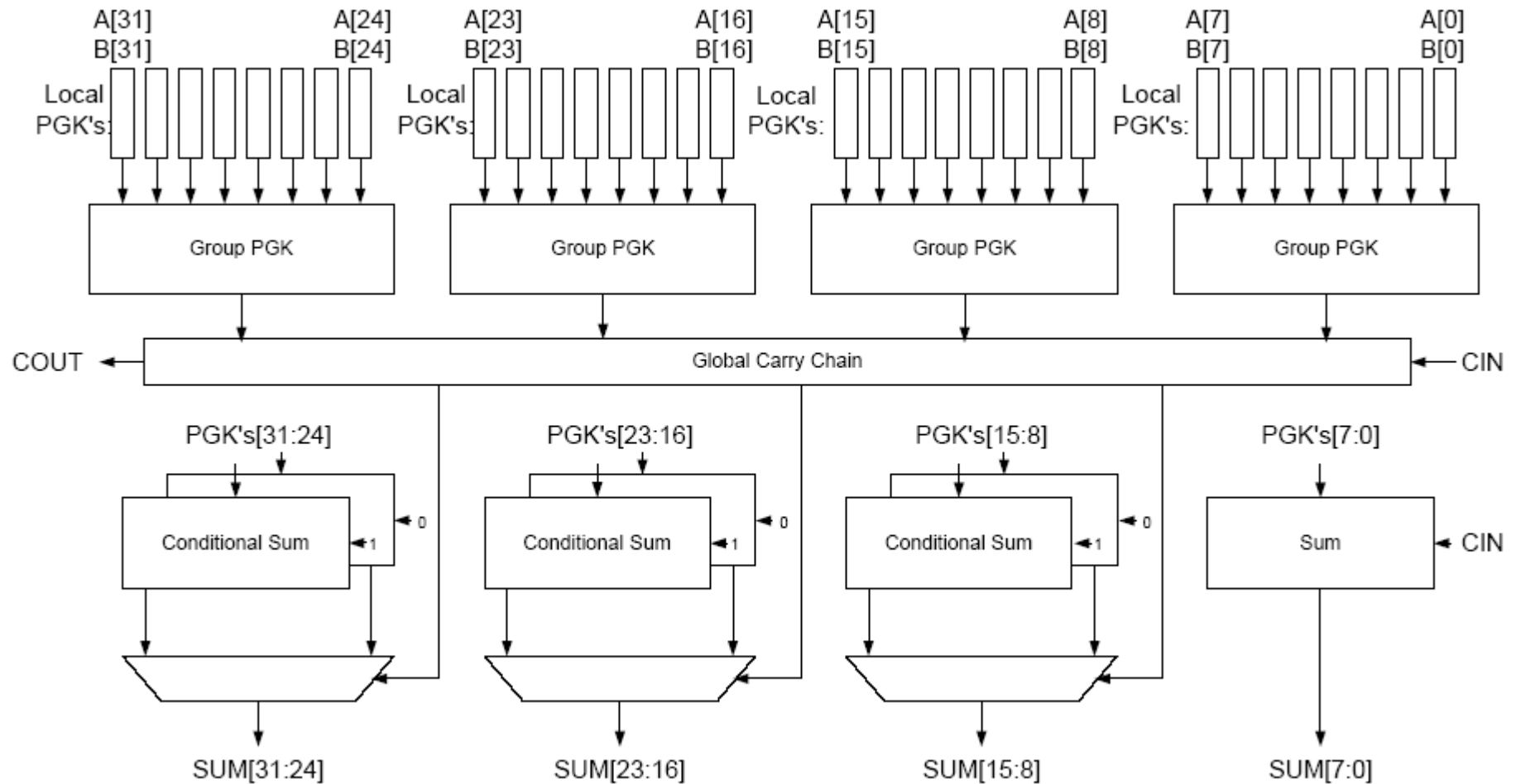
- 72-bit adder (Jason Stinson)
 - 0.35u process
 - Domino
 - Kogge-Stone
 - CLA+sumselect
 - Combines terms in both domino and CMOS stages



Adder from Imagine

- Part of Imagine
 - A high-performance media processor designed at Stanford
- 32-bit segmented integer adder
 - Two-level tree to compute global carries
 - Uses carry-select to compute final sums from global carries
- Static CMOS logic
 - Also pass gate logic
- Design constraints
 - Area
 - Design complexity (modularity)
 - Speed

Adder from Imagine



Adder from Imagine

- It is balancing design/logic complexity and speed
 - It uses large groups which will ultimately limit performance
- It does use some tree structures
 - It does not ripple carries
 - But the group generation is a little slow
- Also uses large block sizes (8 bits)
 - Does not move the carry select input to lower significance
 - Need to worry about how outputs in block are generated

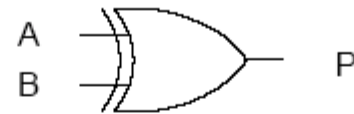
Adder from Imagine

- Local PGK's:
 - Convert input operands into Propagate (P), Generate (G), Kill(K)
- Group PGK's:
 - Determine P,G,K for groups of 8 bits
- Global carry chain:
 - Compute $cin[8]$, $cin[16]$, $cin[24]$, $cout(cin[32])$ from group PGK's and cin
- Conditional sums:
 - Compute 8-bit sum for $cin=0$ and 8-bit sum for $cin=1$ as soon as PGK's are known
- Final Mux:
 - Use cin 's from global carry chain to select conditional sums

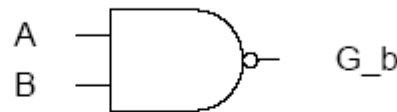
Local PGK Logic (Imagine)

- Pre-computation necessary to do fast carry computation

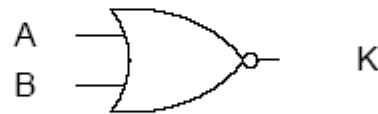
- $P = a \oplus b$



- $G = ab$



- $K = \sim(a+b)$

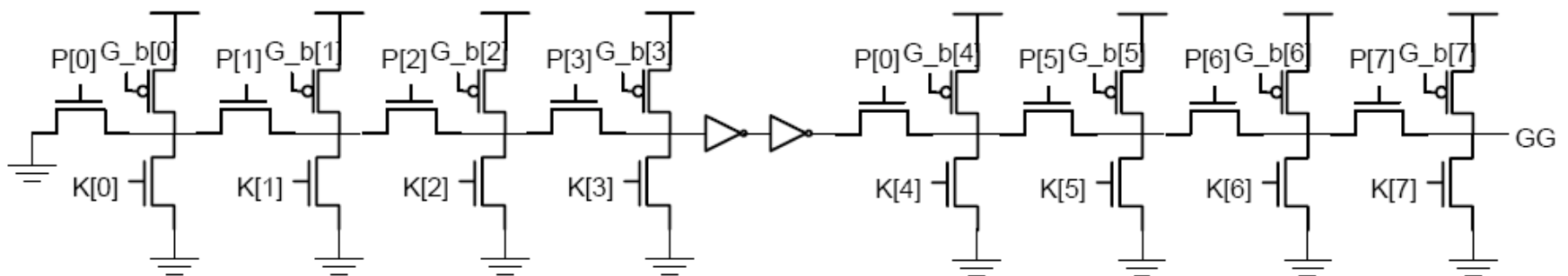


- Size gates to fan-out to four carry chains
- Note: To do $A-B$, use $\sim B$ here

Group PKG (Imagine)

Manchester Carry Chains.

- Usually dynamic, but still works with static logic
- Group PKG's:
 - $GP = P[7].P[6].P[5].P[4].P[3].P[2].P[1].P[0]$
 - $GG = G[7] + G[6].P[7] + G[5].P[6].P[7]+...$
 - $GK = \sim(GG+GP)$
- Use Carry chains
- Example for GG (group generate):



Slide from Mark Horowitz, Stanford

Arithmetic for Media Processing

- Used in Media processing
 - DSP's, multimedia extensions to instruction set architectures (MMX, VIS)
- Consider three variations of conventional arithmetic:
 - Segmented Arithmetic
 - Break carry chain
 - Arithmetic operations similar to add/subtract
 - Example: 4 parallel 8-bit unsigned absolute differences
 - Saturation
 - Don't wraparound on overflow

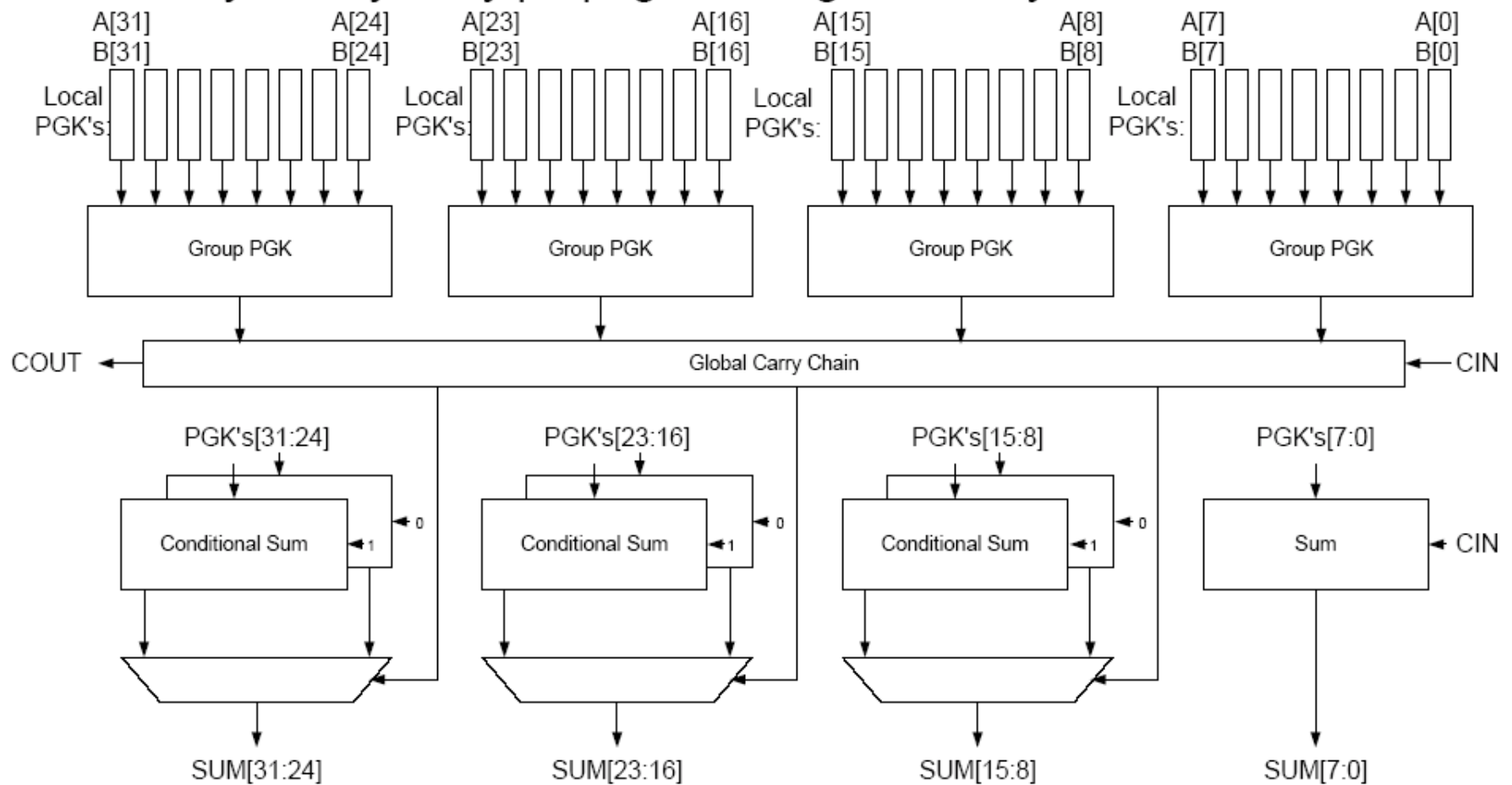
Segmented Add Operation

- Support 32-bit, dual half-word, or quad byte ops
- Example: 4 parallel byte additions
- Treat each byte as a separate 2's complement number
- Don't propagate the carries across byte boundaries

	1 4	0 2	F F	F F
+	F E	0 2	0 1	0 2
	1 2	0 4	0 0	0 1

Modify for Segmentation

- Only modify carry propagation in global carry chain



Saturation

- Often in media and signal processors, saturating arithmetic is supported:
 - Don't wraparound on overflow
 - Result should be largest (or smallest) value possible
- Examples:
 - 32-bit saturating integer add:
 - $IADDS32(0x7FFFFFFF, 0x00000001) = 0x7FFFFFFF$
 - 8-bit saturating unsigned subtract:
 - $USUBS8(0x02FE02FE, 0x03FE01FF) = 0x00000100$

Hardware Support for Saturation

- Overflow detection
 - Example: signed addition
 - Can look at sign bits of inputs and outputs
 - Or can compute using $ovf = cin_{msb} \oplus cout_{msb}$
- Overflow propagation
 - Similar to segmented, global carry chain, except for overflows
- Output muxing
 - Need a many-to-one mux for each byte to choose between: 0xff, 0x00, 0x7f, 0x80 and the unsaturated value
- Methods for speeding up saturation
 - Could probably do “carry-select saturation detection”

Summary (from Harris/Weste)

- ◆ If they're fast enough, use ripple-carry
 - Compact, simple
- ◆ Carry skip and carry select work well for small bit sizes (8-16)
 - Hybrids combining techniques are popular
- ◆ At 32, 64, and beyond, tree adders are much faster
 - Again, hybrids are common

Adder Summary

Table 10.3 Comparison of adder architectures

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-Increment ($n = 4$)		$N/4 + 2$	4	1	$2N$
Carry-Increment (variable group)		$\sqrt{2N}$	$\sqrt{2N}$	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	$N/2$	$N \log_2 N$
Han-Carlson	$(1, 0, L-2)$	$\log_2 N + 1$	2	$N/4$	$0.5 N \log_2 N$
Ladner Fischer ($l = 1$)	$(1, L-2, 0)$	$\log_2 N + 1$	$N/4 + 1$	1	$0.25 N \log_2 N$
Knowles $[2,1,\dots,1]$	$(0, 1, L-2)$	$\log_2 N$	3	$N/4$	$N \log_2 N$

Synthesized Adders (Harris/Weste)

- ◆ Similar to my experiment
 - But with 0.18u library, Synopsys DesignWare
 - Synopsys can map “+” to carry-ripple, carry-select, carry-lookahead, and some prefix adders
 - Fastest are tree adders with (prelayout) speeds of 7.0 and 8.5 FO4 delays for 32 and 64 bit adders

Area vs. Delay, Synthesized Adders

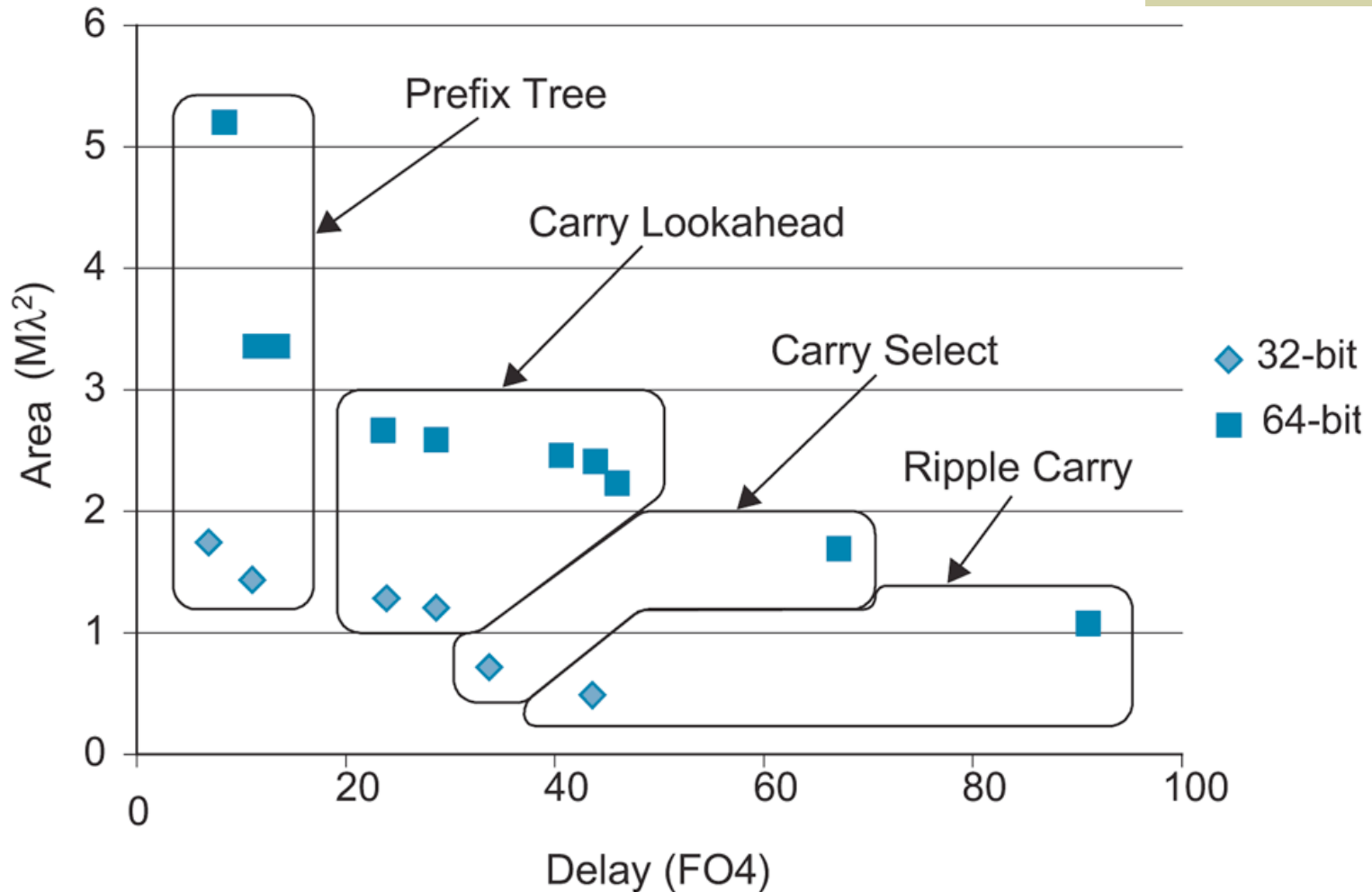


FIG 10.47 Area vs. delay of synthesized adders