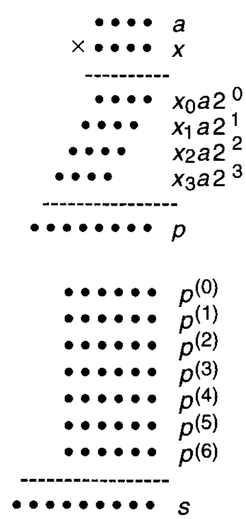
Lecture 8
Multioperand Addition

Uses of Multioperand Addition

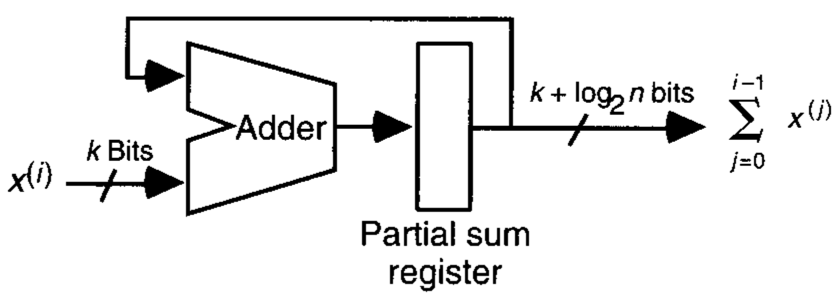
- Multiplication
  - partial products are formed and must be added
- Inner-product computation (Dot Product, Convolution, FIR filter, IIR filter, etc.)
  - terms must be added



"Dot Notation"

- Useful when positioning or alignment of the bits, rather that there values, is important.
  - Each dot represents a digit in a positional number system.
  - Dots in the same column have the same positional weight.
  - Rightmost column is the least significant position.

Serial Multioperand Addition

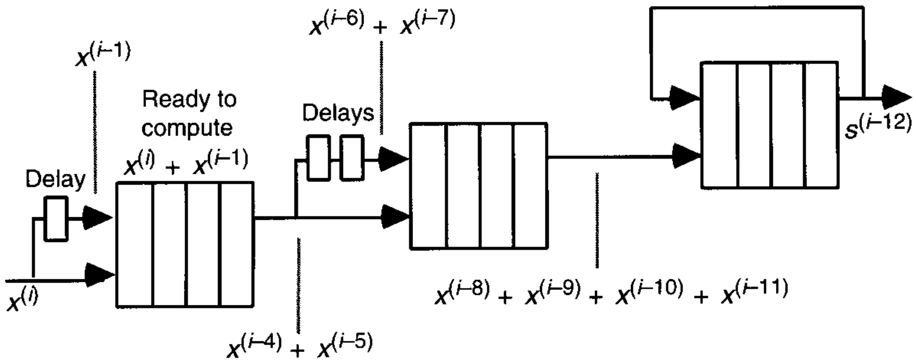


Operands $x^{(0)}$, $x^{(1)}$, … , $x^{(n-1)}$ are shifted in, one per clock cycle.

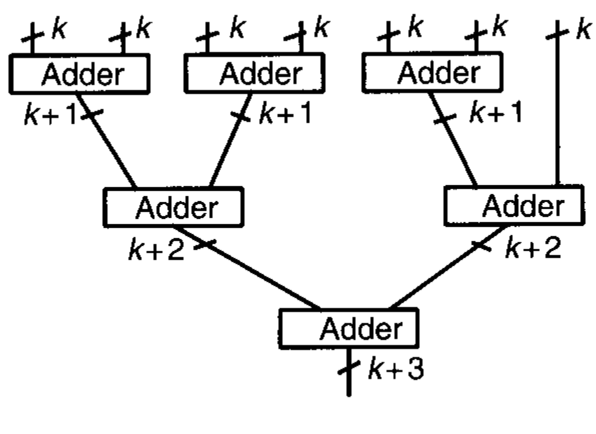
Final sum can be as large as $n(2^k - 1)$.

Partial sum register must be $\log_2(n2^k – n+1) \approx k+\log_2 n$ bits wide.

Pipelined Serial Addition

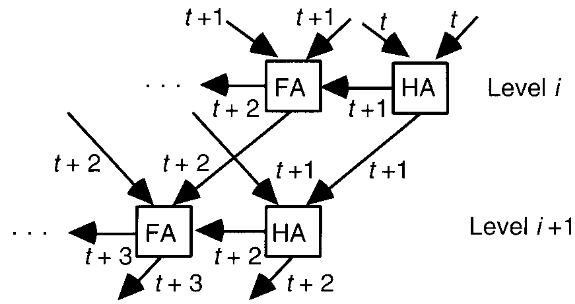


Binary Adder Tree



Ripple-carry might deliver better times than carry-lookahead !?

Analysis of Ripple-Carry Tree Adder



$$T_{\text{tree-ripple-multi-add}} = O(k + \log\ n)$$

Whereas, for carry-lookahead adders ….

$$\begin{aligned} T_{\text{tree-fast-multi-add}} &= O(\log\ k + \log(k+1) + \cdots + \log(k + \lceil \log_2 n \rceil - 1)) \\ &= O(\log\ n\ \ \log\ k + \log\ n \log\ \log\ n) \end{aligned}$$

Can we do better?

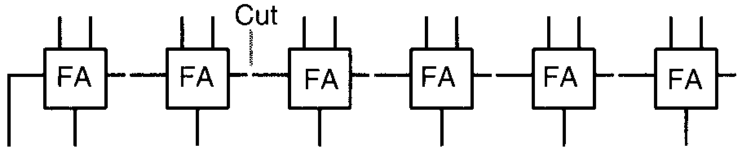
$$T_{\text{tree-ripple-multi-add}} = O(k + \log\ n)$$

The absolute minimum time is $O(\log(kn)) = O(\log\ k + \log\ n)$, where kn is the total number of input bits.

The minimum is achievable with ….
(next slide please)

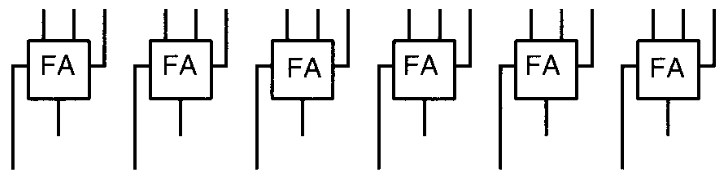
Carry-Save Adders

Ripple-Carry



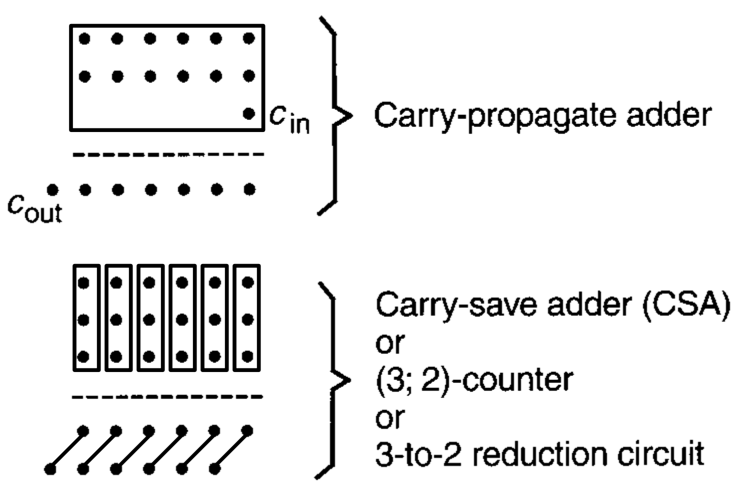
Reduce 2 numbers to their sum.

Carry-Save

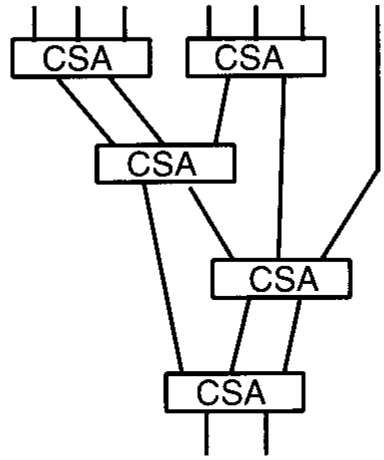


Reduce 3 numbers to two numbers.

More "Dot Notation"



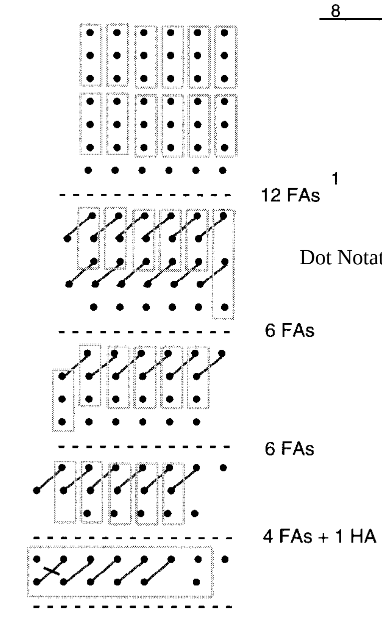
Carry-Save Adder Tree



A carry save tree can reduce *n* binary numbers to two numbers have the same sum in O(log *n*) levels.

Assumes fast logarithmic time adder

$$T_{\text{carry-save-multi-add}} = O(\text{tree height} + T_{\text{CPA}}) = O(\log n + \log k)$$



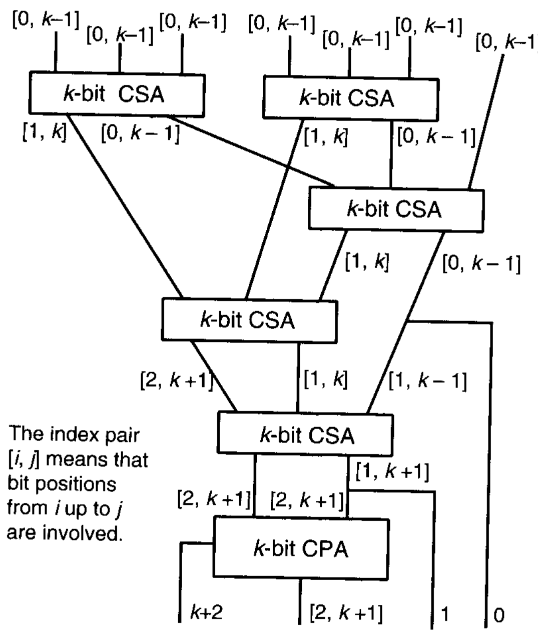
Dot Notation Form

8	7	6	5	4	3	2	1	0	Bit position
			7	7	7	7	7	7	6 × 2 = 12 FAs
		2	5	5	5	5	5	3	6 FAs
		3	4	4	4	4	4	1	6 FAs
	1	2	3	3	3	3	2	1	4 FAs + 1 HA
	2	2	2	2	2	1	2	1	7-bit adder
	Carry-propagate adder								
1	1	1	1	1	1	1	1	1	

Tabular Form

Total cost = 7-bit adder + 28 FAs + 1 HA

Adding seven 6-bit numbers



Seven Input Wallace Tree

In general, an n-input Wallace tree reduces its k-bit inputs to two $(k + \log_2 n - 1)$-bit outputs.

Analysis of Wallace Trees

- The smallest height $h(n)$ of an n-input Wallace tree, satisfies the recurrence:

$$h(n) = 1 + h(\lceil 2n/3 \rceil)$$

solution: $h(n) \geq \log_{1.5}(n/2)$

- The number of inputs n(h) that can be reduced to two outputs by an h-level tree, satisfies the recurrence:

$$n(h) = \lfloor 3n(h-1)/2 \rfloor$$

solution: upper bound: $n(h) \leq 2(3/2)^h$

lower bound: $n(h) > 2(3/2)^{h-1}$

Max number of inputs $n(h)$ for an h-level tree

h	$n(h)$	h	$n(h)$	h	$n(h)$
0	2	7	28	14	474
1	3	8	42	15	711
2	4	9	63	16	1066
3	6	10	94	17	1599
4	9	11	141	18	2398
5	13	12	211	19	3597
6	19	13	316	20	5395

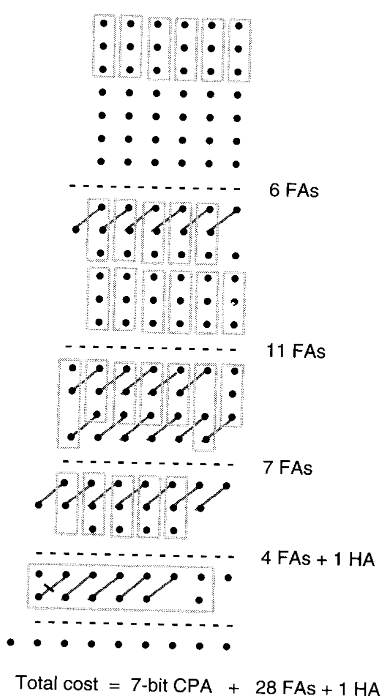
Wallace Tree

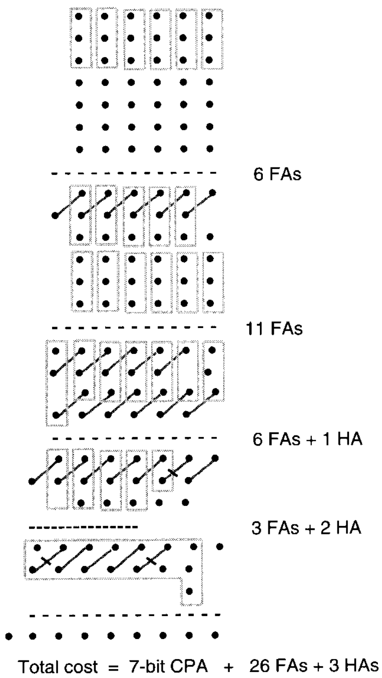
- Reduce the number of operands at the earliest opportunity.
- If there are m dots in a column, apply $\lfloor m/3 \rfloor$ full adders to that column.
- Tends to minimize overall delay by making the final CPA as short as possible.

Dadda Trees

- Reduce the number of operands in the tree to the next lower n(h) number in the table using the fewest FA's and HA's possible.
- Reduces the hardware cost without increasing the number of levels in the tree.

Dadda Tree for 7-input 6-bit addition



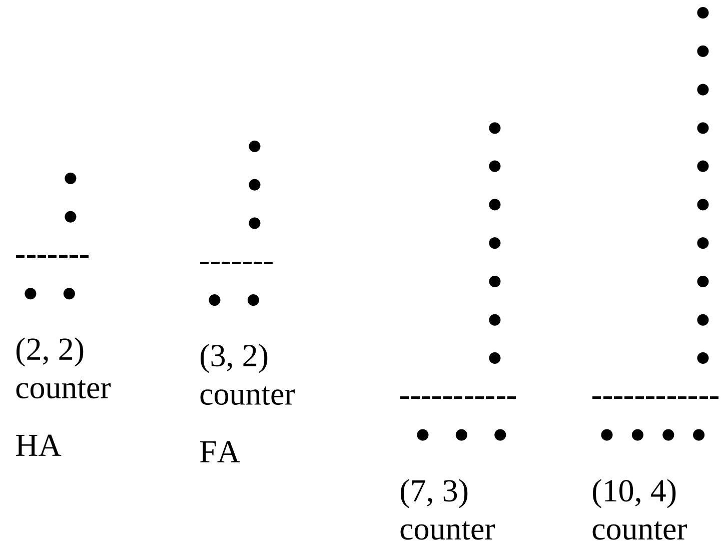


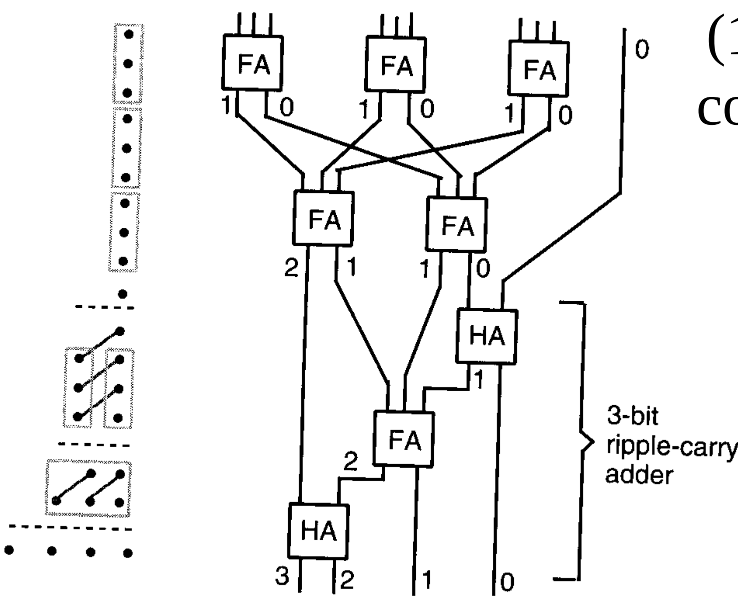
Taking advantage of the carry-in of the final CPA stage

Parallel Counters

- Receives *n* inputs
- Counts the number of 1's among the *n* inputs
- Outputs a $\lfloor \log_2(n+1) \rfloor$ bit number
- Reduces n dots in the same bit position to $\lfloor \log_2(n+1) \rfloor$ dots in different positions.

Parallel Counters



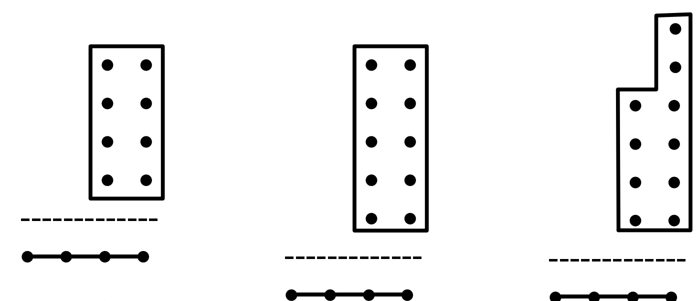


(10, 4) counter

Generalized Parallel Adders

- Reduces “dot patterns” (not necessarily in the same column) to other dot patterns (not necessarily only one in the each column).
- Book speaks less generally, and restricts output to only one dot in each column.

4 Examples



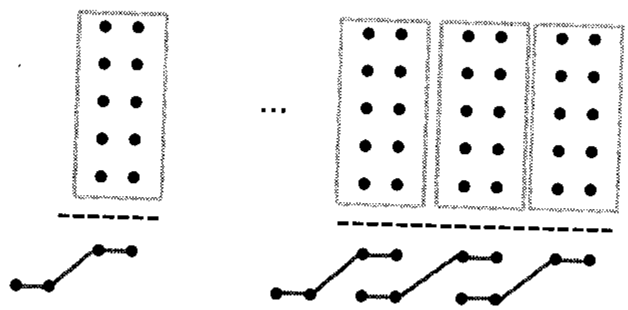
(4, 4; 4) counter

(5, 5; 4) counter

(4, 6; 4) counter

4-bit binary full adder, with carry in, is a (2, 2, 2, 3; 5) counter

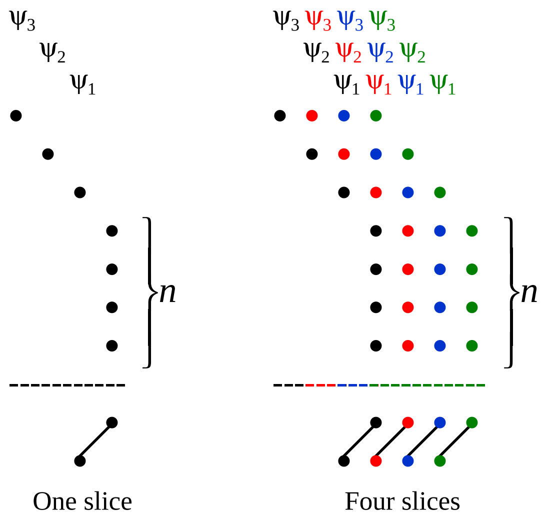
Reducing 5 Numbers with (5, 5 ; 4) Counters



(n; 2) Counters

- Difference in notation from other counters.
- Reduce n (larger than 3) numbers to two numbers.
- Each slice i of an (n; 2) counter:
  - receives carry bits from one or more positions to the right (i-1, i-2, ….)
  - produces outputs to positions i and i+1
  - produces carries to one or more positions to the left (i+1, i+2, ….)

(n; 2) Counters
Slice by Slice



$n + \psi_1 + \psi_2 + \psi_3 \leq$
$3 + 2\psi_1 + 4\psi_2 + 8\psi_3$

Adding Multiple Signed Numbers

- By means of sign extension

Extended positions					Sign	Magnitude positions
x_{k-1}	x_{k-1}	x_{k-1}	x_{k-1}	x_{k-1}	x_{k-1}	$x_{k-2}\ x_{k-3}\ x_{k-4} \cdots$
y_{k-1}	y_{k-1}	y_{k-1}	y_{k-1}	y_{k-1}	y_{k-1}	$y_{k-2}\ y_{k-3}\ y_{k-4} \cdots$
z_{k-1}	z_{k-1}	z_{k-1}	z_{k-1}	z_{k-1}	z_{k-1}	$z_{k-2}\ z_{k-3}\ z_{k-4} \cdots$

- By method of negative weighted sign bits

Extended positions					Sign	Magnitude positions
1	1	1	1	0	$\bar{x}_{k-1}$	$x_{k-2}\ x_{k-3}\ x_{k-4} \cdots$
					$\bar{y}_{k-1}$	$y_{k-2}\ y_{k-3}\ y_{k-4} \cdots$
					$\bar{z}_{k-1}$	$z_{k-2}\ z_{k-3}\ z_{k-4} \cdots$
					1	