The slide features a blue background with a grid of computer components. In the center, the text "Lecture 6: Multi-Operand Addition" is displayed in a bold, dark blue font. Below this, "ECE 645—Computer Arithmetic" and "3/5/08" are written in a smaller, dark blue font. The background includes various icons: a CPU, a keyboard, a mouse, a monitor, and a printer. The text "High Performance" is on the left, "CoolClock" is on the right, and "CoolRunner-II" is in the center. A blue horizontal line is at the bottom of the slide area.

**Lecture 6:
Multi-Operand Addition**

ECE 645—Computer Arithmetic
3/5/08

ECE 645 – Computer Arithmetic

Lecture Roadmap

- Multi-Operand Addition
- Adder Trees
- Carry-Save Adders and Carry-Save Adder Trees
- Wallace and Dadda Trees
- 5:3 Counter and Generalized Counters

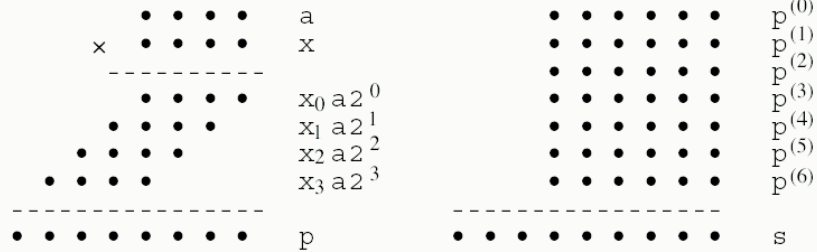
Required Reading

- B. Parhami, *Computer Arithmetic: Algorithms and Hardware Design*
 - Chapter 8, Multioperand Addition
- Note errata at:
 - http://www.ece.ucsb.edu/~parhami/text_comp_arit.htm#errors

3



Applications of Multi-Operand Addition



Multiplication

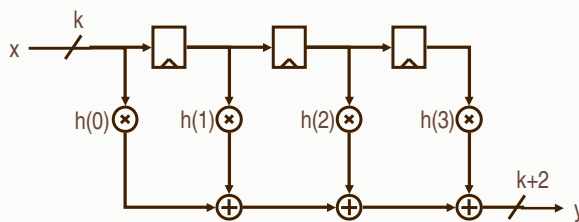
$$p = a \cdot x$$

Inner product

$$s = \sum_{i=0}^{n-1} x^{(i)} y^{(i)} = \sum_{i=0}^{n-1} p^{(i)}$$

5

Applications of Multi-Operand Addition



- Finite Impulse Response (FIR) filter
- 4-tap filter requires 4 numbers to be summed
 - T-tap filter requires T numbers to be summed
- In the following slides, begin by considering multi-operand addition using two-input **k-bit** adders
- O is the "big O" notation, representing an order of magnitude approximation without units
 - Helps as rough approximation for comparisons, ignoring minor components (small constants, etc.)
- Ripple carry: propagation delay is $O(k)$
- Fast adders (carry-select, carry-lookahead, prefix, etc): propagation delay is $O(\log_2(k))$

6

Number of Bits of the Result (Unsigned Binary)

$$S = \sum_{i=0}^{n-1} x^{(i)} \quad x^{(i)} \in [0..2^k-1]$$

$$S_{\min} = 0 \quad S_{\max} = n(2^k-1)$$

$$\begin{aligned} \# \text{ of bits of } S &= \lceil \log_2 (S_{\max} + 1) \rceil = \\ &= \lceil \log_2 (n(2^k-1) + 1) \rceil \leq \lceil \log_2 n 2^k \rceil = \\ &= k + \lceil \log_2 n \rceil \end{aligned}$$

7

Serial Implementation of Multi-Operand Addition

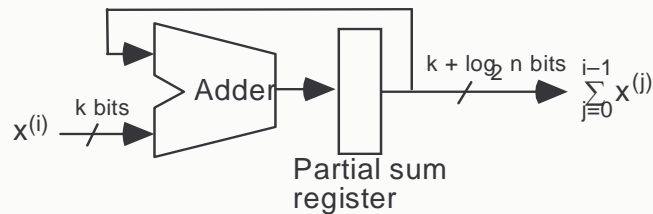


Fig. 8.1 Serial implementation of multi-operand addition with a single 2-operand adder.

$$\begin{aligned} T_{\text{serial-multi-add}} &= O(n \log(k + \log n)) \quad \leftarrow \text{assume fast adder propagation time} \\ &= O(n \log k + n \log \log n) \end{aligned}$$

Therefore, addition time grows superlinearly with n when k is fixed and logarithmically with k for a given n

8

Pipelined Implementation for Higher Throughput

Problem to think about: Ignoring start-up and other overheads, this scheme achieves a speedup of 4 with 3 adders. How is this possible?

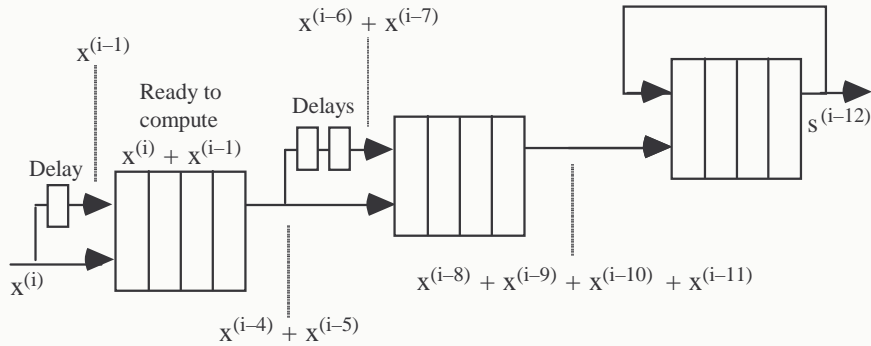


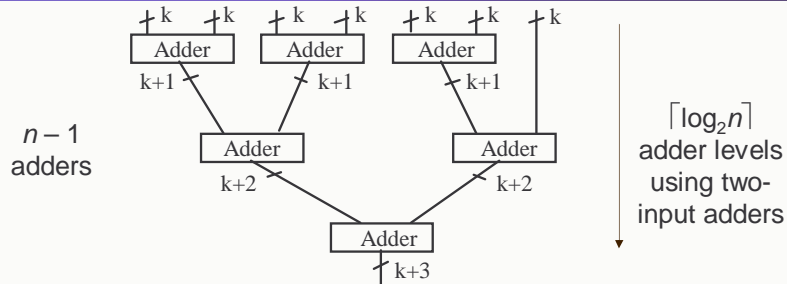
Fig. 8.1 Serial multi-operand addition when each adder is a 4-stage pipeline.

Adder Trees

High Performance
CoolClock
CoolRunner-II
Low Power
DataGate

ECE 645 – Computer Arithmetic

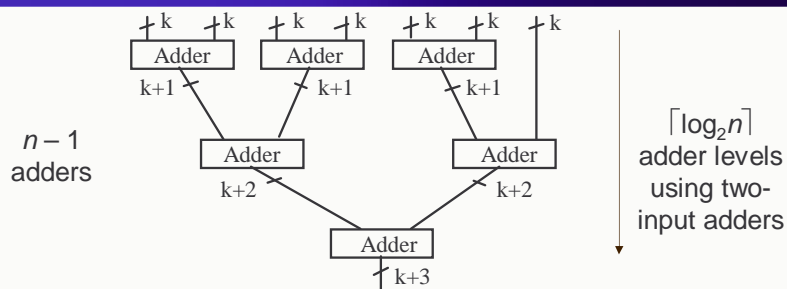
Parallel Addition via Adder Trees



- We will look at three methods of addition trees
 - 1) fast adder tree (carry-lookahead, carry-select, etc.)
 - 2) ripple carry adder tree
 - 3) carry adder tree

11

Binary Adder Trees



$$T_{\text{tree-fast-multi-add}} = O(\log k + \log(k+1) + \dots + \log(k + \lceil \log_2 n \rceil - 1))$$

$$= O(\log n \log k + \log n \log \log n)$$

explanation: $\log_2(n)$ stages, each stage increasing by one bit

$$T_{\text{tree-ripple-multi-add}} = O(k + (k+1) + \dots + (k + \lceil \log_2 n \rceil - 1))$$

explanation: $\log_2(n)$ stages, each stage increasing by one bit

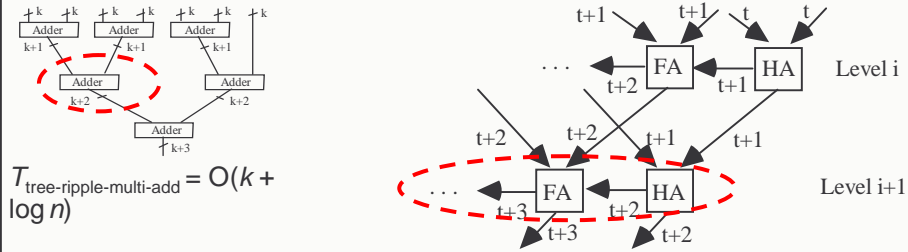
This can be optimized to:

$$= O(k + \log n)$$

[Justified on the next slide]

12

Elaboration on Tree of Ripple-Carry Adders



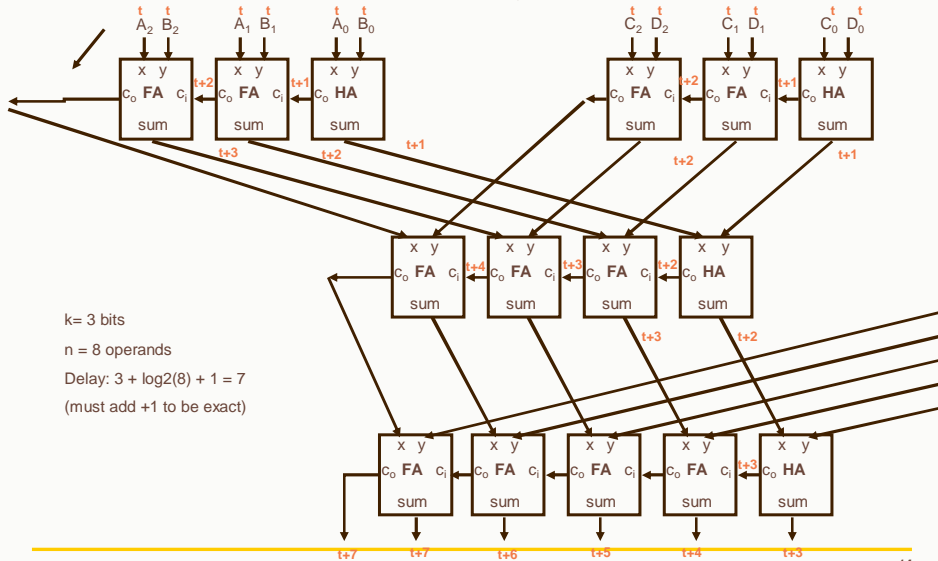
$$T_{\text{tree-ripple-multi-add}} = O(k + \log n)$$

Fig. 8.5 Ripple-carry adders at levels i and $i+1$ in the tree of adders used for multi-operand addition.

- Sometimes ripple-carry tree can have advantage over fast adder in that the computations can "overlap" in time between levels
- May or may not be true with fast adders, depending on architecture
 - In general, fast adders need previous stage to complete before going to next stage

Example: Adding 8 three bit numbers

recall: carry-out becomes the msb of the new sum in unsigned arithmetic



Latency of Multi-Operand Addition in Parallel

$$T_{\text{tree-fast-multi-add}} = O(\log k + \log(k+1) + \dots + \log(k + \lceil \log_2 n \rceil - 1))$$

$$= O(\log n \log k + \log n \log \log n)$$

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

In the next slides we will present a carry-save adder tree:

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

$$= O(\log n + \log k)$$

15

Example Calculations of Latency (Propagation Delay)

Latency for different adder trees (in units of 1 full-adder delay)

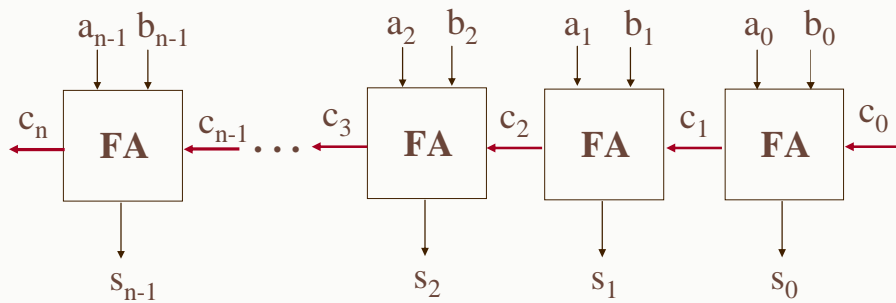
	n=4 k=8	n=4 k=32	n=4 k=128	n=16 k=8	n=16 k=32	n=16 k=128	n=128 k=8	n=128 k=32	n=128 k=128
Fast Adder Tree	7	11	15	13	21	29	25	36	50
Ripple Carry Tree	10	34	130	12	36	132	15	39	135
Carry Save Tree	5	7	9	7	9	11	10	12	14

- These are **very rough** approximations of order of magnitude accuracy only! They are used to show trends.
- Conclusions of latency/propagation delay:
 - Comparing fast adder tree versus ripple-carry tree:
 - Fast adder usually superior for larger values of k
 - Ripple carry can be superior for large values of n with small values of k
 - Carry-save tree is superior to both

16

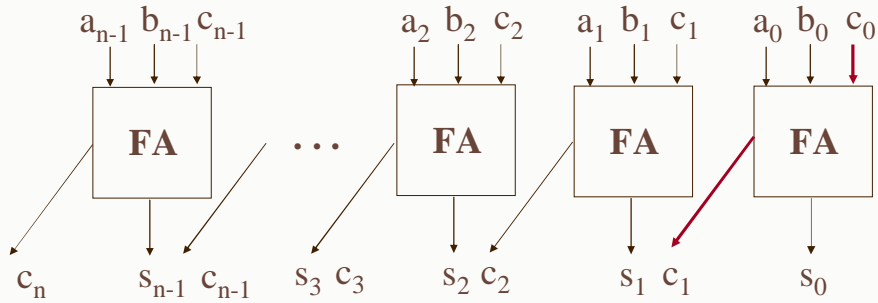


Ripple-Carry Carry Propagate Adder (CPA)



- Critical path is n full adders

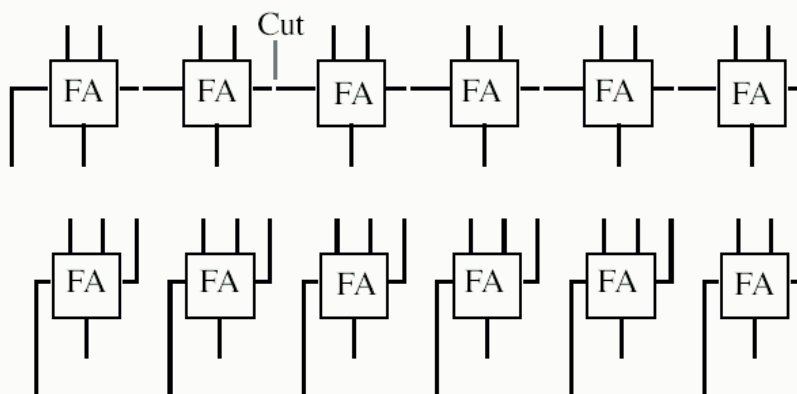
Carry Save Adder



- Critical path is 1 full adder, **regardless of the number of bits n !**

19

A Ripple-Carry vs. Carry-Save Adder



20

Operation of a Carry Save Adder (CSA)

Example

	2^4	2^3	2^2	2^1	2^0
x	0	1	0	1	0
y	1	1	0	1	1
z	1	0	1	1	1
s	0	0	1	1	0
c	1	1	0	1	1

$$x+y+z = s + c$$

21

Carry Propagate and Carry-Save Adders in Dot Notation

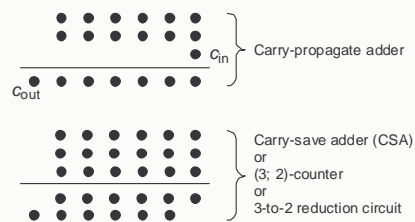


Fig. 8.7 Carry-propagate adder (CPA) and carry-save adder (CSA) functions in dot notation.

22

Specifying Full- and Half-Adder Blocks in Dot Notation

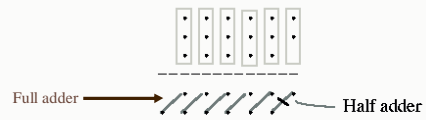


Fig. 8.8 Specifying full- and half-adder blocks, with their inputs and outputs, in dot notation.

Carry-Save Adder Trees

CoolRunner-II

High Performance

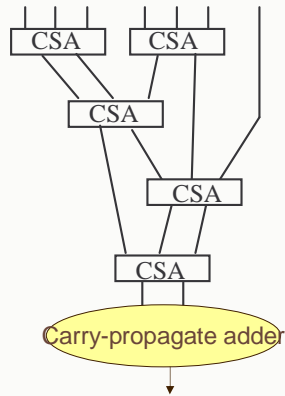
CoolClock

Low Power

DataGate

ECE 645 – Computer Arithmetic

Carry-Save Adder Tree



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

$$= O(\log n + \log k)$$

$$C_{\text{carry-save-multi-add}} = (n - 2)C_{\text{CSA}} + C_{\text{CPA}}$$

- CSA adder tree uses
 - **very roughly** $\log_2(n)$ stages of carry-save adders
 - This is why use "big O" notation
 - Each stage is of 1 latency unit (i.e. one full adder cell), so these comprise $\log_2(n)$ latency units
 - We will find exact number of stages in upcoming slides
 - One fast k-bit carry-propagate adder
 - Assuming you use a fast adder, this comprises approximately $\log_2(k)$ latency units

25

Carry-Save Adder for Four Operands

X_3	X_2	X_1	X_0
Y_3	Y_2	Y_1	Y_0
Z_3	Z_2	Z_1	Z_0
W_3	W_2	W_1	W_0

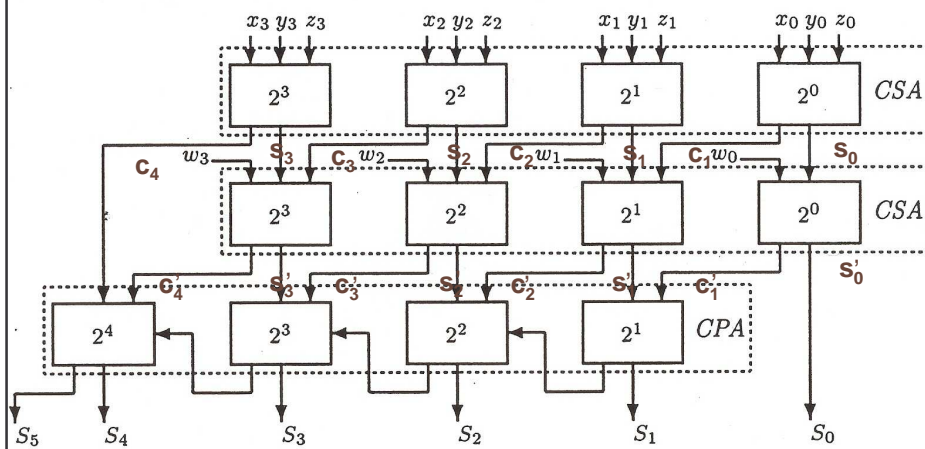
S_3	S_2	S_1	S_0
C_4	C_3	C_2	C_1
W_3	W_2	W_1	W_0

C_4	S'_3	S'_2	S'_1	S'_0
C'_4	C'_3	C'_2	C'_1	

S_5 S_4 S_3 S_2 S_1 S_0

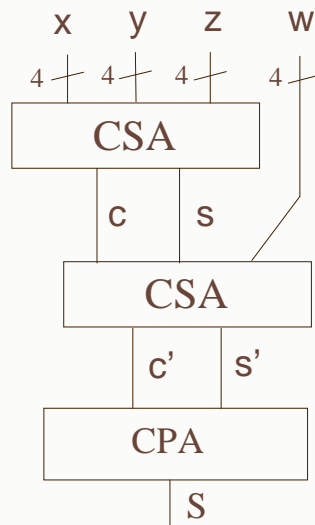
26

Carry-Save Adder for Four Operands



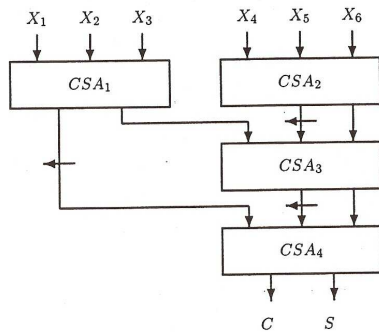
27

Carry-Save Adder for Four Operands

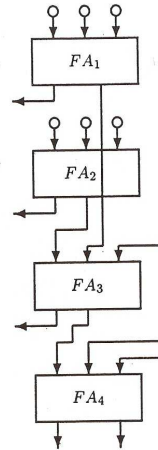


28

Carry-Save Adder for Six Operands



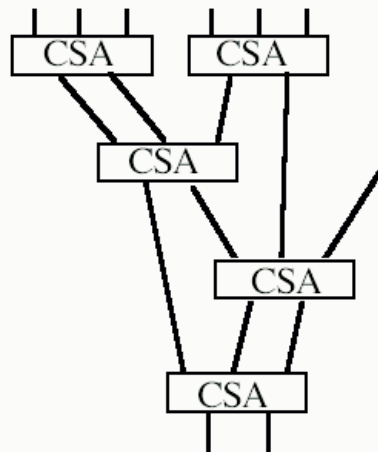
CSA tree



Implementation of
one-bit slice

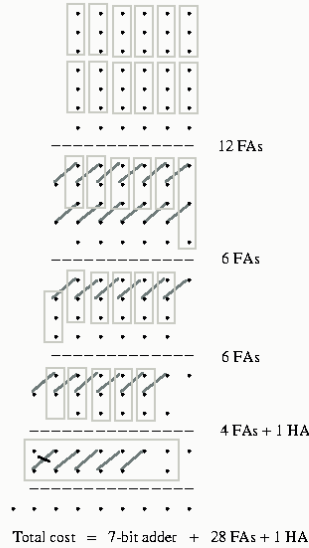
29

Tree of Carry-Save Adders Reducing Seven Numbers to Two



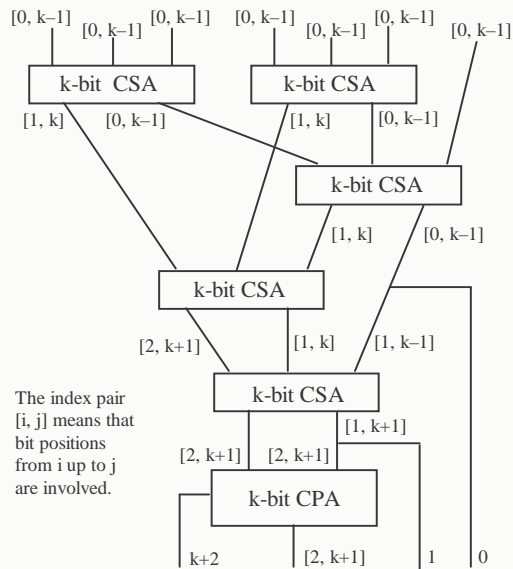
30

Addition of Seven Six-Bit Numbers in Dot Notation



31

Width of Adders in a CSA Tree



Due to the gradual retirement (dropping out) of some of the result bits, CSA widths do not vary much as we go down the tree levels

32

Serial Carry Save Adder

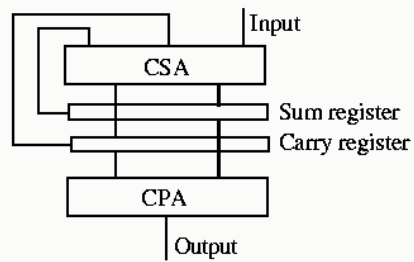


Fig. 8.13 Serial carry-save addition using a single CSA.

33

Parameters of Tree Carry-Save Adders

Latency

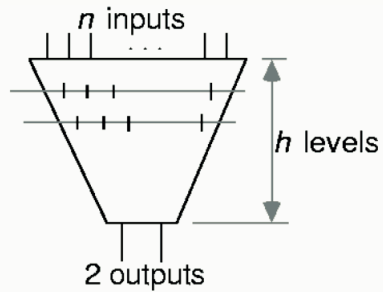
$$\text{Latency}_{\text{CSA}} = h(n) \cdot T_{\text{FA}} + \text{Latency}_{\text{CPA}}(k, n)$$

Tree height
for n operands

Component Adders	Widths
CSA	$k \dots k + \lceil \log_2 n \rceil$ typically close to k bits
CPA	$\leq k + \lceil \log_2 n \rceil$

34

Relationship Between Number of Inputs and Tree Height



35

Number of Inputs as Function of Tree Height

Maximum number of inputs that can be reduced to two by an h -level tree, $n(h)$

$$n(0) = 2$$

$$n(h) = \left\lfloor \frac{3}{2} n(h-1) \right\rfloor$$

$$n(1) = 3$$

$$n(2) = 4$$

$$n(3) = 6$$

$$n(4) = 9$$

$$n(5) = 13$$

$$n(6) = 19$$

$$2 \left(\frac{3}{2} \right)^{h-1} < n(h) \leq 2 \left(\frac{3}{2} \right)^h$$

36

Maximum number of input n(h)

Table 8.1 The maximum number n(h) of inputs for an h-level carry-save-adder 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

37

Tree Height as Function of Number of Inputs

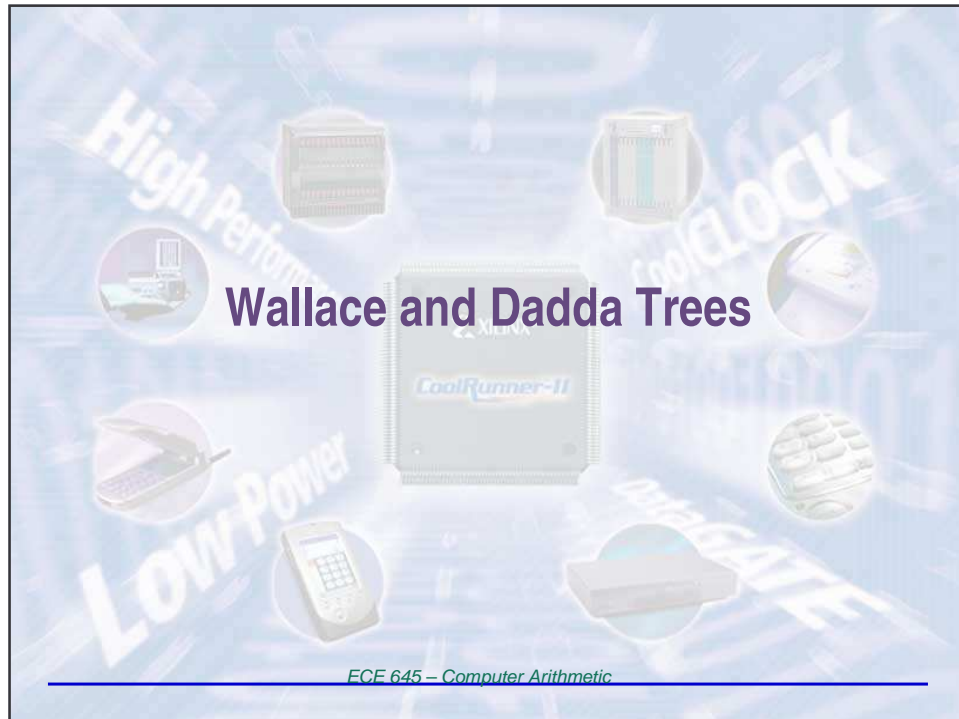
**Smallest height of the tree carry save adder
for n operands, h(n)**

$$h(n) = 1 + h\left(\left\lceil \frac{2}{3}n \right\rceil\right)$$

$$h(2) = 0$$

$$h(n) \geq \log_{\frac{3}{2}}\left(\frac{n}{2}\right)$$

38



Wallace Trees

- So far all examples of carry-save adder trees have been Wallace Trees
- Wallace Trees
 - Reduce the size of the final Carry Propagate Adder (CPA)
 - Generally optimum from the point of view of **speed**
- Dadda Trees
 - Reduce the cost of the carry save tree
 - Generally optimum (among the CSA trees) from the point of view of **area**

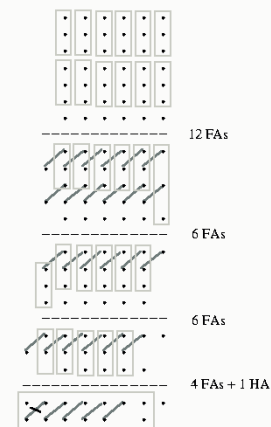
Wallace and Dadda Trees

- Wallace reduces number of operands at earliest opportunity
 - Goal of this is to have smallest number of bits for CPA adder
 - However, sometimes having a few bits longer CPA adder does not affect the propagation delay significantly (i.e. carry-lookahead)
- Dadda seeks to reduce the number of FA and HA units
 - May be at the cost of a slightly larger final CPA

41

Wallace Tree

In a Wallace tree, we reduce the number of operands at the earliest possible opportunity

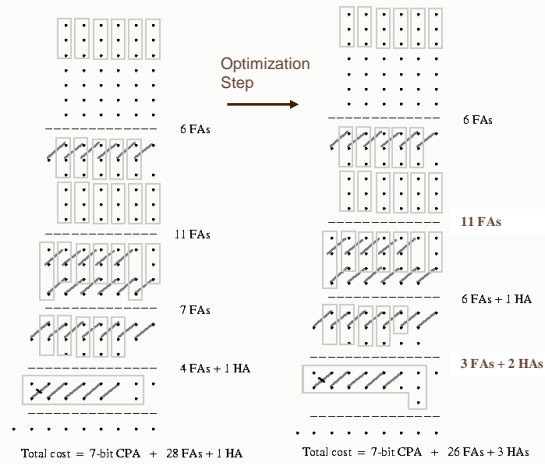


Total cost = 7-bit adder + 28 FAs + 1 HA

42

Dadda Tree

In a Dadda tree, we reduce the number of operands at the latest possible opportunity that leads to no added delay (target the next smaller number in Table 8.1)



5:3 Counter and Generalized Counters

High Performance

CoolClock

Low Power

DataRate

CoolRunner-II

ECE 645 - Computer Arithmetic

General Compressors

- So far we have seen a 3 input to 2 output reduction circuit. Often called:
 - Carry-save adder
 - 3:2 compressor
 - 3:2 counter
- This was implemented as a full adder
- Can also have other compression ratios
 - 5:3 is an example

45

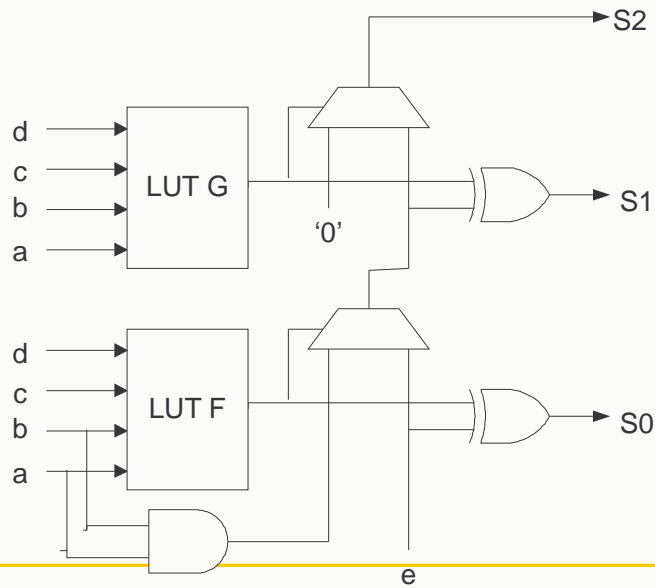
5-to-3 Parallel Counter (or Compressor)

	2^4	2^3	2^2	2^1	2^0
a	0	1	0	1	0
b	1	1	0	1	1
c	1	0	1	1	1
d	1	0	1	1	1
e	1	1	1	1	1
s0	0	1	1	1	0
s1	0	1	1	0	0
s2	1	0	0	1	1

$$a+b+c+d+e = s0+s1+s2$$

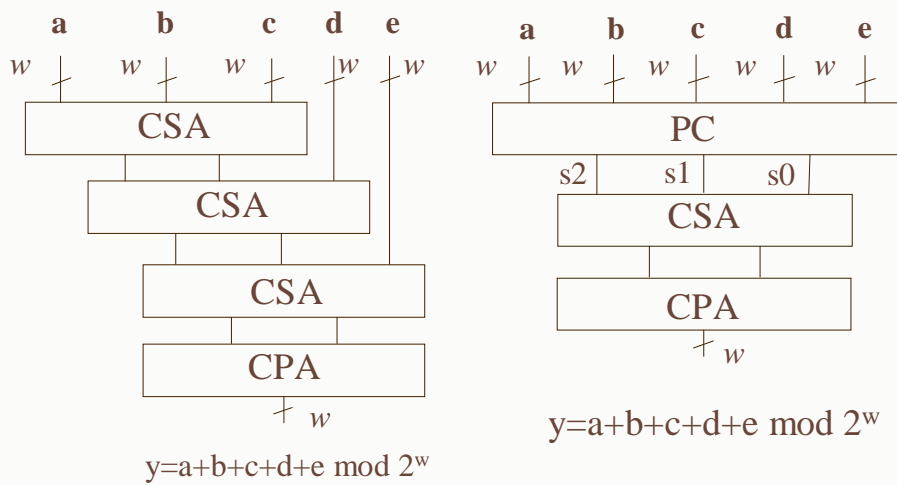
46

Implementation of 1-bit of 5-to-3 Parallel Counter using Single CLB Slice of a Virtex FPGA



47

Carry Save Adder vs. 5-to-3 Parallel Counter



48

Generalized Parallel Counters

