# UNIVERSITY OF MASSACHUSETTS
## Dept. of Electrical & Computer Engineering

## Digital Computer Arithmetic
## ECE 666

### Part 6b
### High-Speed Multiplication - II

Israel Koren

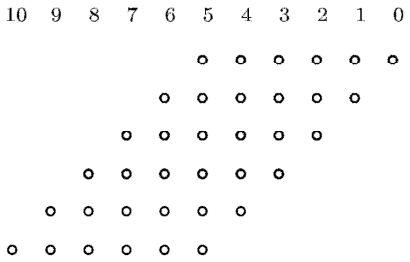



## Accumulating the Partial Products

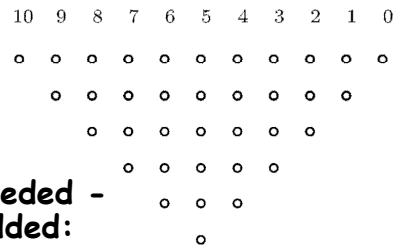
- After generating partial products either directly or using smaller multipliers
- Accumulate these to obtain final product
  - A fast multi-operand adder
- Should take advantage of particular form of partial products - reduce hardware complexity
- They have fewer bits than final product, and must be aligned before added
- Expect many columns that include fewer bits than total number of partial products - requiring simpler counters

# Example - Six Partial Products

- Generated when multiplying unsigned 6-bit operands using one-bit-at-a-time algorithm
- 6 operands can be added using 3-level carry-save tree



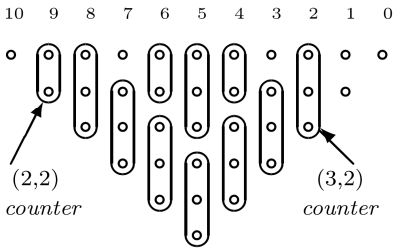

- Number of (3,2) counters can be substantially reduced by taking advantage of the fact that all columns but 1 contain fewer than 6 bits
- Deciding how many counters needed - redraw matrix of bits to be added:



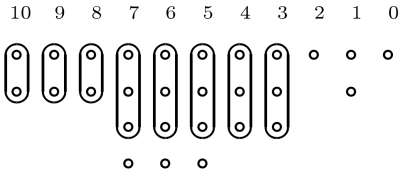




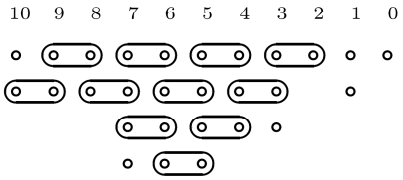
# Reduce Complexity - Use (2,2) Counters (HAs)



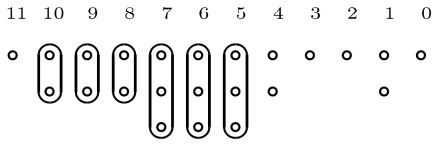

(a) Level 1 carry-save addition.




(c) Level 2 carry-save addition.




(b) Results of level 1.

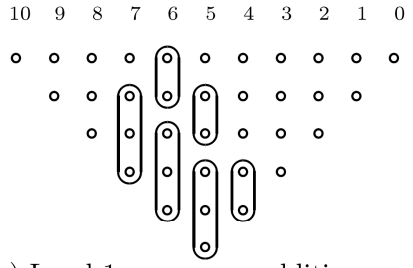



(d) Level 3 carry-save addition.

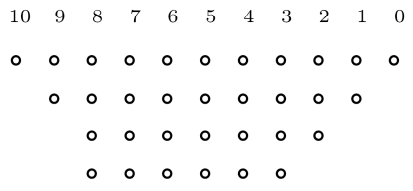
- Number of levels still 3, but fewer counters

## Further reduction in number of counters

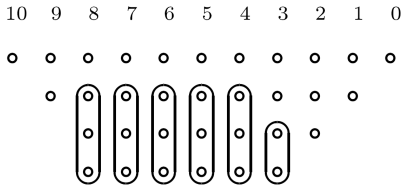
- Reduce # of bits to closest element of 3,4,6,9,13,19,…
- 15 (3,2) and 5 (2,2) vs. 16 (3,2) and 9 (2,2) counters



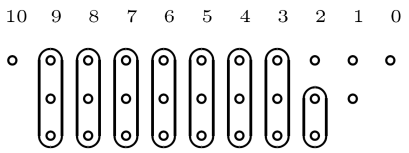

(a) Level 1 carry-save addition.




(b) Results of level 1.



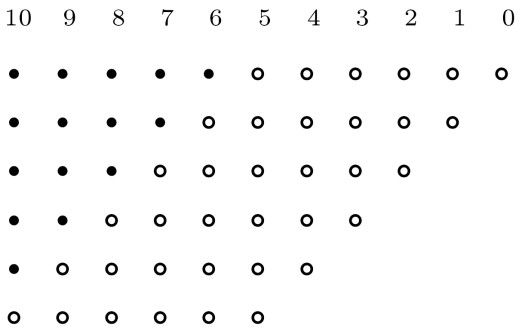

(c) Level 2 carry-save addition.




(d) Level 3 carry-save addition.

## Modified Matrix for Negative Numbers

- Sign bits must be properly extended
- In row 1: 11 instead of 6 bits, and so on
- Increases complexity of multi-operand adder
- If two's complement obtained through one's complement - matrix increased even further



## Reduce Complexity Increase

- Two's complement number
  **s s s s s s $z_4$ $z_3$ $z_2$ $z_1$ $z_0$**
  with value

$$-s\cdot 2^{10}+s\cdot 2^9+s\cdot 2^8+s\cdot 2^7+s\cdot 2^6+s\cdot 2^5+z_4\cdot 2^4+z_3\cdot 2^3+z_2\cdot 2^2+z_1\cdot 2^1+z_0$$

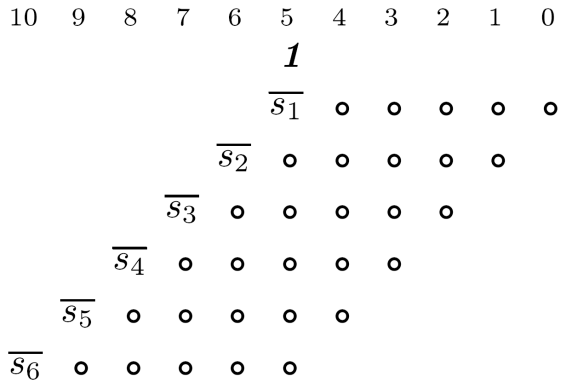
- Replaced by
  **0 0 0 0 0 (-s) $z_4$ $z_3$ $z_2$ $z_1$ $z_0$**
- since

$$-s\cdot 2^{10}+s\cdot(2^9+2^8+2^7+2^6+2^5)$$
$$=-s\cdot 2^{10}+s\cdot(2^{10}-2^5)=\quad -s\cdot 2^5$$



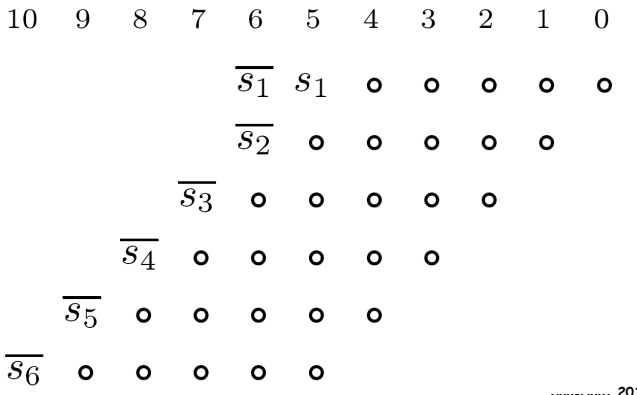

## New Bit Matrix

- To get -s in column 5 - complement original s to (1-s) and add 1
  - Carry of 1 into column 6 serves as the extra 1 needed for sign bit of second partial product
- New matrix has fewer bits but higher maximum height (7 instead of 6)



## Eliminating Extra 1 in Column 5

- Place two sign bits $s_1$ and $s_2$ in same column
- $(1-s_1)+(1-s_2) = 2 - s_1 - s_2$
- 2 is carry out to next column
- Achieved by first extending sign bit $s_1$

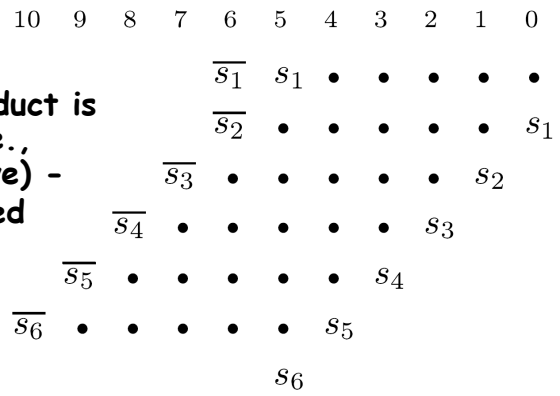






## Using One's Complement and Carry

- Add extra carries to matrix
- Full circles - complements of corresponding bits are taken whenever $s_i=1$
- Extra $s_6$ in column 5 increases maximum column height to 7
- If last partial product is always positive (i.e., multiplier is positive) - $s_6$ can be eliminated



## Example

- Recoded multiplier using canonical recoding

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | | | | | 0 | 1 | 0 | 1 | 1 | 0 | 22 |
| $X$ | | | $\times$ | | 0 | 0 | 1 | 0 | 1 | 1 | 11 |
| $Y$ | | | | | 0 | 1 | 0 | $\bar{1}$ | 0 | $\bar{1}$ | Recoded multiplier |
| **1** | **1** | **1** | **1** | **1** | **1** | 0 | 1 | 0 | 1 | 0 | |
| **0** | **0** | **0** | **0** | **0** | 0 | 0 | 0 | 0 | 0 | | |
| **1** | **1** | **1** | **1** | 0 | 1 | 0 | 1 | 0 | | | |
| **0** | **0** | **0** | 0 | 0 | 0 | 0 | 0 | | | | |
| **0** | **0** | 1 | 0 | 1 | 1 | 0 | | | | | |
| **0** | 0 | 0 | 0 | 0 | 0 | | | | | | |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | |




## Smaller Matrix for the Example

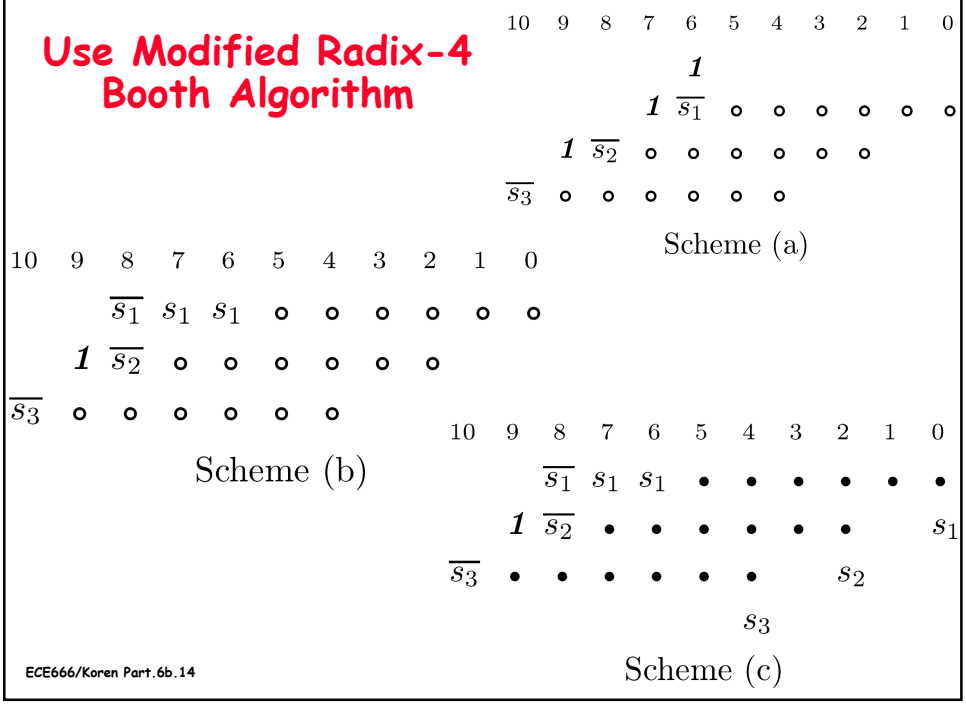
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | **0** | **1** | 0 | 1 | 0 | 1 | 0 |
| | | | | **1** | 0 | 0 | 0 | 0 | 0 | |
| | | | **0** | 0 | 1 | 0 | 1 | 0 | | |
| | | **1** | 0 | 0 | 0 | 0 | 0 | | | |
| | **1** | 1 | 0 | 1 | 1 | 0 | | | | |
| **1** | 0 | 0 | 0 | 0 | 0 | | | | | |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |

## Using One's Complement and Carry

| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | **0** | **1** | 0 | 1 | 0 | 0 | 1 |
| | | | | **1** | 0 | 0 | 0 | 0 | 0 | **1** |
| | | | **0** | 0 | 1 | 0 | 0 | 1 | **0** | |
| | | **1** | 0 | 0 | 0 | 0 | 0 | **1** | | |
| | **1** | 1 | 0 | 1 | 1 | 0 | **0** | | | |
| **1** | 0 | 0 | 0 | 0 | 0 | **0** | | | | |
| | | | | | **0** | | | | | |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |




## Use Modified Radix-4 Booth Algorithm



## Example 2: Using radix-4 modified Booth's

- Same recoded multiplier $01010\bar{1}0\bar{1}$

| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| | | **0** | **1** | **1** | 0 | 1 | 0 | 1 | 0 |
| | ***1*** | **0** | 0 | 1 | 0 | 1 | 0 | | |
| **1** | 1 | 0 | 1 | 1 | 0 | | | | |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |

| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| | | **0** | **1** | **1** | 0 | 1 | 0 | 0 | 1 |
| | ***1*** | **0** | 0 | 1 | 0 | 0 | 1 | | **1** |
| **1** | 1 | 0 | 1 | 1 | 0 | | **1** | | |
| | | | | | **0** | | | | |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |



## Alternative Techniques for Partial Product Accumulation

- Reducing number of levels in tree - speeding up accumulation
- Achieving more regular design
- Tree structures usually have irregular interconnects
  - Irregularity complicates implementation- area-inefficient layouts
- Number of tree levels can be lowered by using reduction rate higher than 3:2
- Achieve 2:1 reduction rate by using SD adders
  - SD adder also generates sum in constant time
  - Number of levels in SD adder tree is smaller
  - Tree produces a single result rather than two for CSA tree

## Final Result of SD Tree

- In most cases, conversion to two's complement needed
- Conversion done by forming two sequences:
- First - $Z^+$ - created by replacing each negative digit of SD number by 0
- Second - $Z^-$ - replaces each negative digit with its absolute value, and each positive digit by 0
- Difference $Z^+ - Z^-$ - found by adding two's complement of $Z^-$ to $Z^+$ using a CPA
- Final stage of a CPA needed as in CSA tree

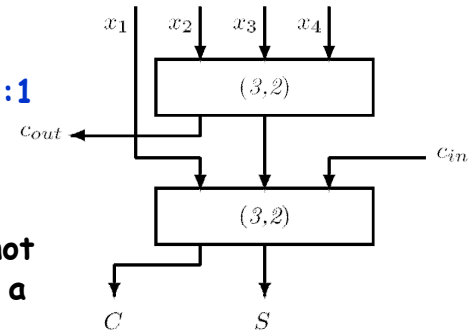



## SD Adder Tree vs. CSA Tree

- SD - no need for a sign bit extension when negative partial products - no separate sign bit
- Design of SD adder more complex - more gates and larger chip area - each signed digit requires two ordinary bits (or multiple-valued logic)
- Comparison between the two must be made for specific technology
- Example:
  - 32x32 Multiplier based on radix-4 modified Booth's algorithm - 16 partial products
  - CSA tree with 6 levels, SD adder tree with 4 levels
  - Sophisticated logic design techniques and layout schemes result in less area-consuming implementations

## (4;2) Compressors

- Same reduction rate of 2:1 achieved without SD representations by using (4;2) compressors
- Designed so that Cout is not a function of Cin to avoid a ripple-carry effect

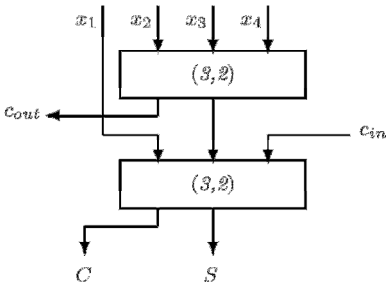


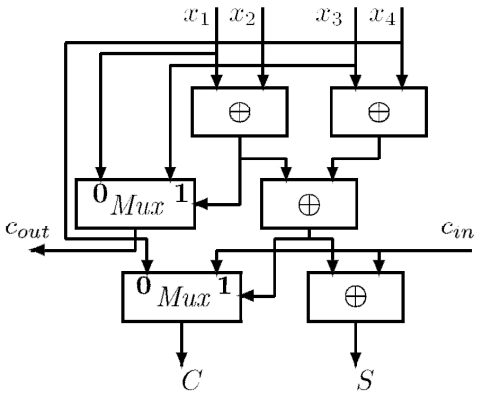

- (4;2) compressor may be implemented as a multi-level circuit with a smaller overall delay compared to implementation based on 2 (3,2) counters




## Example Implementation

- Delay of 3 exclusive-or gates to output S vs. delay of 4 exclusive-or gates
  - 25% lower delay



# Other Multi-Level Implementations of a (4;2) Compressor

- **All implementations must satisfy**

$$x_1 + x_2 + x_3 + x_4 + c_{in} = S + 2(C + c_{out})$$

- **Cout should not depend on Cin to avoid horizontal rippling of carries**
- **Truth table : (a,b,c,d,e,f - Boolean variables)**

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $c_{out}$ | $C$ | $S$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | $c_{in}$ |
| 0 | 0 | 0 | 1 | 0 | $c_{in}$ | $\overline{c_{in}}$ |
| 0 | 0 | 1 | 0 | 0 | $c_{in}$ | $\overline{c_{in}}$ |
| 0 | 0 | 1 | 1 | $a$ | $\bar{a}$ | $c_{in}$ |
| 0 | 1 | 0 | 0 | 0 | $c_{in}$ | $\overline{c_{in}}$ |
| 0 | 1 | 0 | 1 | $b$ | $\bar{b}$ | $c_{in}$ |
| 0 | 1 | 1 | 0 | $c$ | $\bar{c}$ | $c_{in}$ |
| 0 | 1 | 1 | 1 | 1 | $c_{in}$ | $\overline{c_{in}}$ |

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $c_{out}$ | $C$ | $S$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | $c_{in}$ | $\overline{c_{in}}$ |
| 1 | 0 | 0 | 1 | $d$ | $\bar{d}$ | $c_{in}$ |
| 1 | 0 | 1 | 0 | $e$ | $\bar{e}$ | $c_{in}$ |
| 1 | 0 | 1 | 1 | 1 | $c_{in}$ | $\overline{c_{in}}$ |
| 1 | 1 | 0 | 0 | $f$ | $\bar{f}$ | $c_{in}$ |
| 1 | 1 | 0 | 1 | 1 | $c_{in}$ | $\overline{c_{in}}$ |
| 1 | 1 | 1 | 0 | 1 | $c_{in}$ | $\overline{c_{in}}$ |
| 1 | 1 | 1 | 1 | 1 | 1 | $c_{in}$ |

- **Previous implementation - a=b=c=1, d=e=f=0**




# Comparing Delay of Trees

| Number of operands | Number of levels using (3,2) | Number of levels using (4;2) | Equivalent delay |
|---|---|---|---|
| 3 | 1 | 1 | 1.5 |
| 4 | 2 | 1 | 1.5 |
| 5 – 6 | 3 | 2 | 3 |
| 7 – 8 | 4 | 2 | 3 |
| 9 | 4 | 3 | 4.5 |
| 10 – 13 | 5 | 3 | 4.5 |
| 14 – 16 | 6 | 3 | 4.5 |
| 17 – 19 | 6 | 4 | 6 |
| 20 – 28 | 7 | 4 | 6 |
| 29 – 32 | 8 | 4 | 6 |
| 33 – 42 | 8 | 5 | 7.5 |

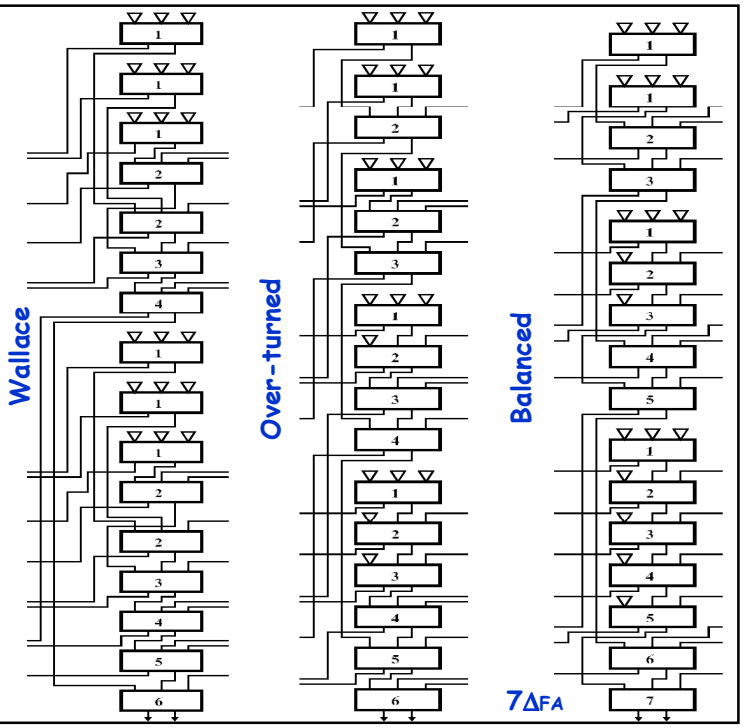
## Other Implementations

- Other counters and compressors can be used: e.g., (7,3) counters
- Other techniques suggested to modify CSA trees which use (3,2) counters to achieve a more regular and less area-consuming layout
- Such modified tree structures may require a somewhat larger number of CSA levels with a larger overall delay
- Two such techniques are:
  - Balanced delay trees
  - Overturned-stairs trees




## Bit-Slices for Three Techniques

* 18 operands
* radix-4 modified Booth
* Triangles - multiplexers
* Rectangles - (3,2) counters
* 15 outgoing & incoming carries aligned
* adjacent bit-slices abut

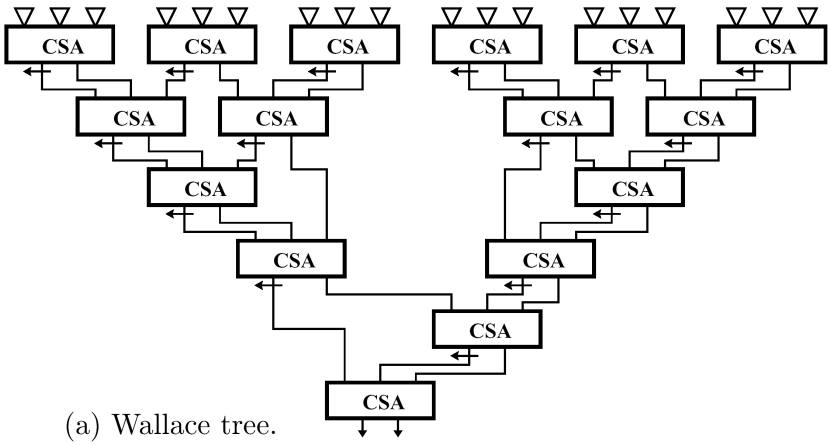


## Comparing the Three Trees

- Incoming carries routed so that all inputs to a counter are valid before or at necessary time
- Only for balanced tree - all 15 incoming carries generated exactly when required - all paths balanced
- In other 2 - there are counters for which not all incoming carries are generated simultaneously
  - For example, bottom counter in overturned-stairs - incoming carries with delays of $4\Delta_{FA}$ and $5\Delta_{FA}$
- Number of wiring tracks between adjacent bit-slices (affect layout area)
  - Wallace tree requires 6; overturned-stairs 3; balanced tree 2 tracks
- Tradeoff between size and speed
  - Wallace : lowest delay but highest number of wiring tracks
  - Balanced: smallest number of wiring tracks but highest delay



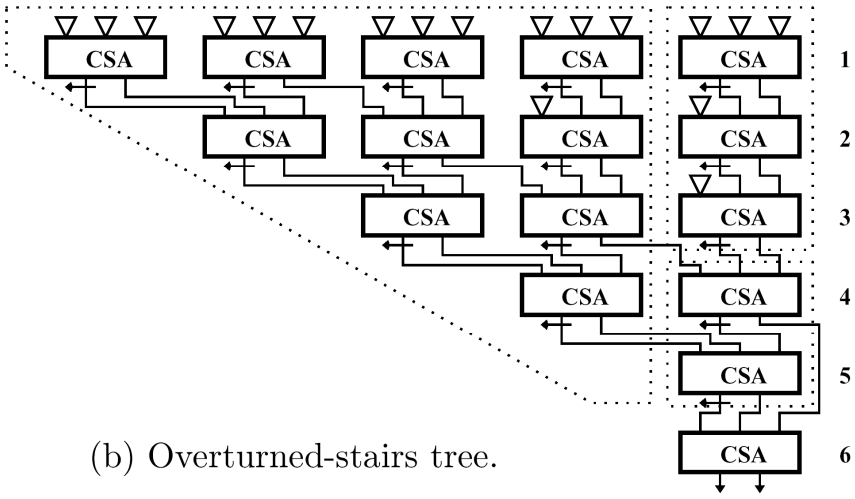

## Complete Structure of Wallace Tree




(a) Wallace tree.

- Balanced and overturned-stairs have regular structure - can be designed in a systematic way

## Complete Structure of Over-turned Tree

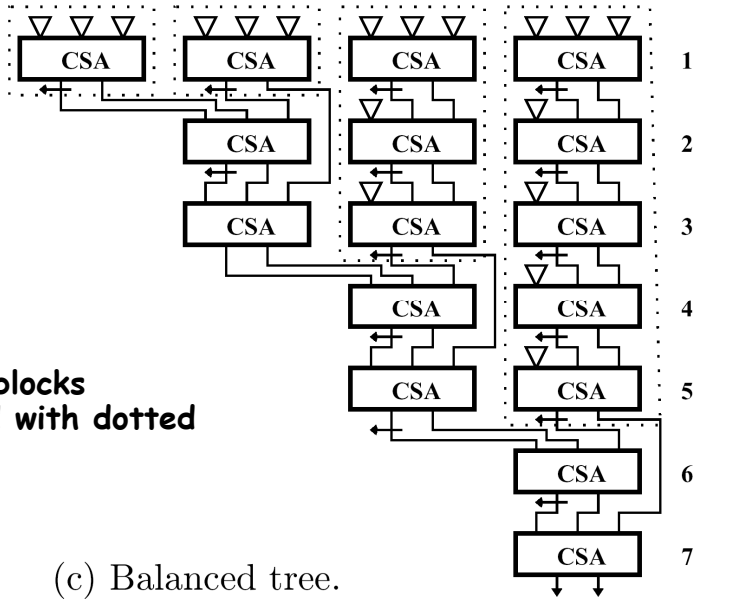



(b) Overturned-stairs tree.

- Building blocks indicated with dotted lines




## Complete Structure of Balanced Tree

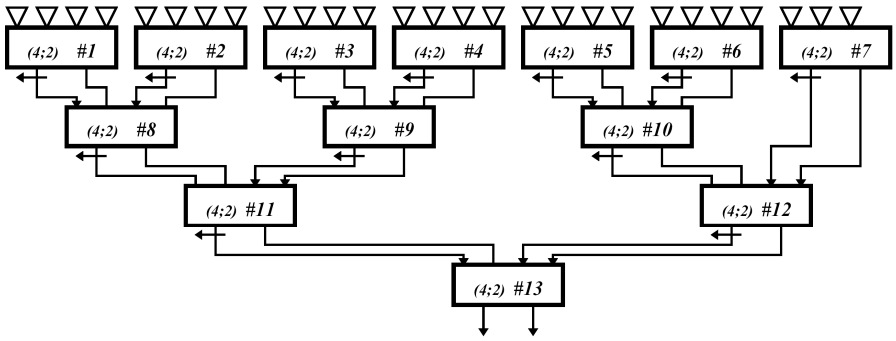



- Building blocks indicated with dotted lines

(c) Balanced tree.

## Layout of CSA Tree

- Wires connecting carry-save adders should have roughly same length for balanced paths
- CSA tree for 27 operands constructed of (4;2) compressors

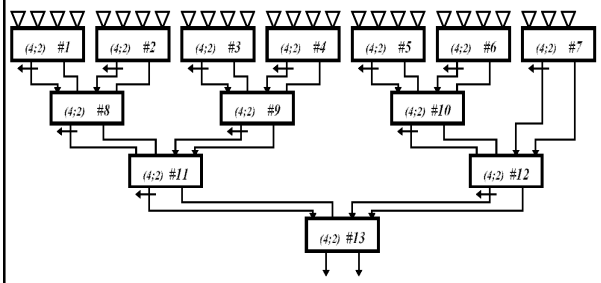


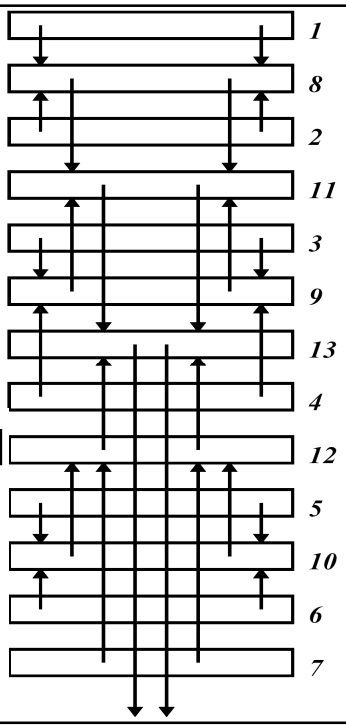




## Layout of CSA Tree

- Bottom compressor (#13) is located in middle so that compressors #11 and #12 are roughly at same distance from it
- Compressor #11 has equal length wires from #8 and #9



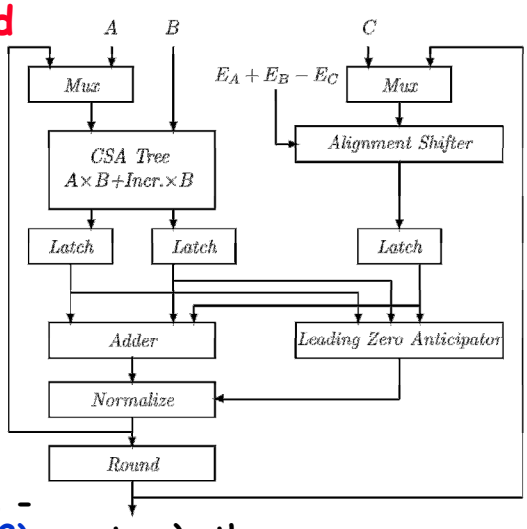



## Fused Multiply-Add Unit

- Performs A x B followed by adding C
  - A x B + C done as single and indivisible operation
- Multiply only: set C=0; add (subtract) only: set B=1
  - Can reduce overall execution time of chained multiply and then add/subtract operations
- Example: Evaluation of a polynomial $a_nx^n+a_{n-1}x^{n-1}+...+a_0$ through $[(a_nx+a_{n-1})x+a_{n-2}]x$ + ...
- Independent multiply and add operations can not be performed in parallel
- Another advantage for floating-point operations - rounding performed only once for A x B+C rather then twice for multiply and add
  - Rounding introduces computation errors - reducing number of roundings reduces overall error




## Implementing Fused Multiply-Add Unit




- A,B,C - significands; $E_A$,$E_B$,$E_C$ - exponents of operands
- CSA tree generates partial products and performs carry-save accumulation to produce 2 results which are added with properly aligned C
- Adder gets 3 operands - first reduces to 2 ((3,2) counters), then performs carry-propagate addition
- Post-normalization and rounding executed next

## Two Techniques to reduce Execution Time

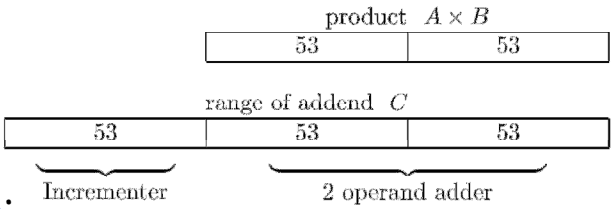
- First: leading zero anticipator circuit uses propagate and generate signals produced by adder to predict type of shift needed in post-normalization step
- It operates in parallel to addition so that the delay of normalization step is shorter
- Second (more important): alignment of significand $C$ in $E_A+E_B-E_C$ done in parallel to multiplication
- Normally, align significand of smaller operand (smaller exponent)
- Implying: if $A \times B$ smaller than $C$, have to shift product after generation - additional delay




## Instead - Always align $C$

- Even if larger than $A \times B$ - allow shift to be performed in parallel to multiplication
- Must allow $C$ to shift either to right (traditional) or left
- Direction - $E_A+E_B-E_C$ is positive or negative
- If $C$ shifted to left - must increase total number of bits in adder

## Example




- Long IEEE operands - possible range of $C$ relative to $AxB$:
- $53 \geq E_A+E_B-E_C \geq -53$
- If $E_A+E_B-E_C \geq 54$, bits of $C$ shifted further to right will be replaced by a sticky bit, and if $E_A+E_B-E_C \leq -54$, all bits of $AxB$ replaced by sticky bit
- Overall penalty - 50% increase in width of adder - increasing execution time
- Top 53 bits of adder need only be capable of incrementing if a carry propagates from lower 106 bits




## Additional Computation Paths

- Path from Round to multiplexer on right used for $(XxY+Z)+AxB$
- Path from Normalize to multiplexer on left used for $(X \times Y+Z) \times B + C$
- Rounding step for $(X \times Y+Z)$ is performed at same time as multiplication by B, by adding partial product Incr.x B to CSA tree