UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Digital Computer Arithmetic
ECE 666

Part 5c
Fast Addition - III

Israel Koren

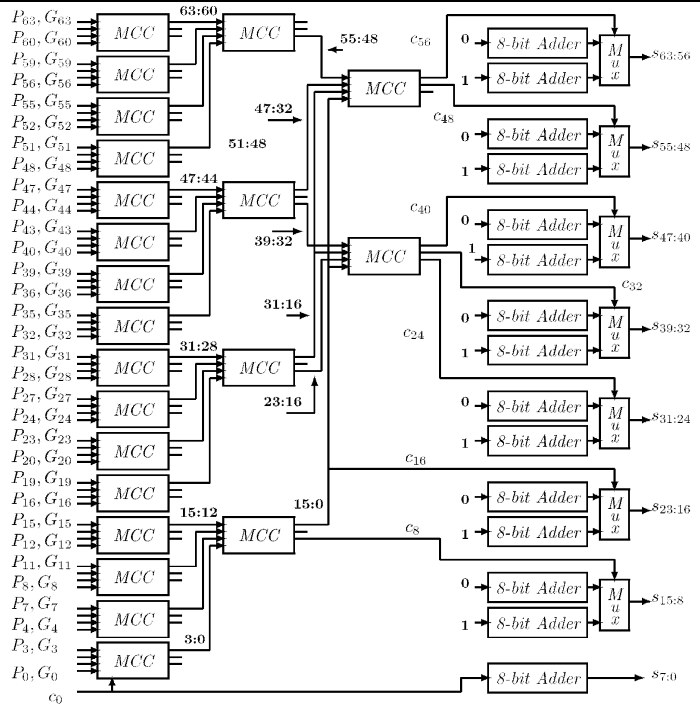
Hybrid Adders

- Combination of two or more addition methods
- Common approach: one method for carry - another for sum
- Two hybrid adders combining variation of a carry-select for sum and modified Manchester for carry
  - Both divide operands into equal groups - 8 bits each
  - First - uses carry-select for sum for each group of 8 bits separately
  - Second - uses a variant of conditional-sum
- Group carry-in signal that selects one out of two sets of sum bits not generated in ripple-carry
- Instead, carries into 8-bit groups generated by a carry-look-ahead tree
- 64-bit adder - carries are $c_8, c_{16}, c_{24}, c_{32}, c_{40}, c_{48}, c_{56}$

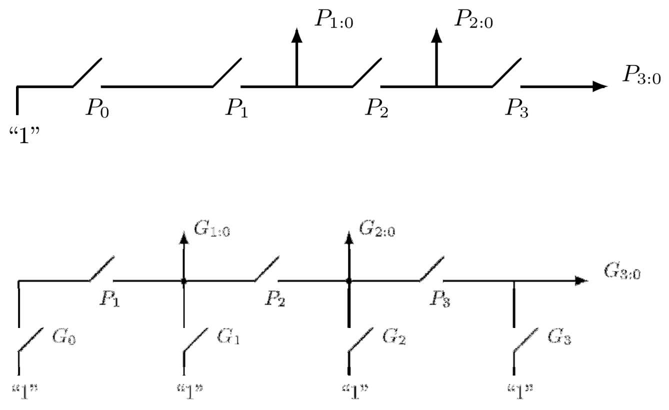
Blocking Factor in Carry Tree

- Structure of carry-look-ahead tree for generating carries similar to those seen before
- Differences - variations in blocking factor at each level and exact implementation of fundamental carry operator
- Restricting to a fixed blocking factor - natural choices include 2, 4 or 8
  - 2 - largest number of levels in tree, vs.
  - 8 - complex modules for fundamental carry operator with high delay
- Factor of 4 - a reasonable compromise
- A Manchester carry propagate/generate module (MCC) with a blocking factor of 4

64-bit Hybrid Adder

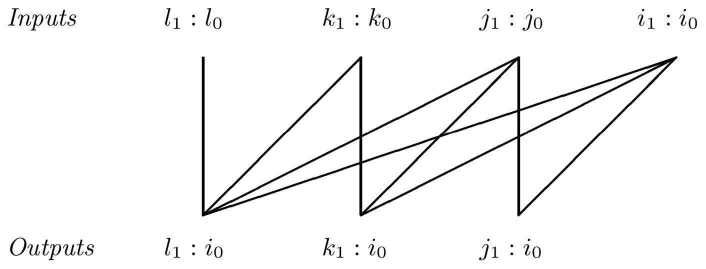


Manchester Carry Module



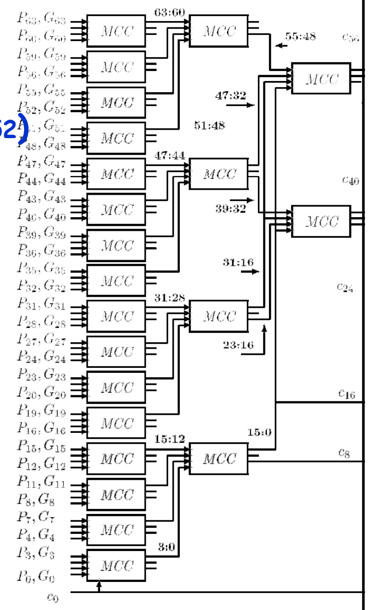
MCC - General Case

- MCC accepts 4 pairs of inputs:
- $(P_{i1:i0},G_{i1:i0}),(P_{j1:j0},G_{j1:j0}),(P_{k1:k0},G_{k1:k0}),(P_{l1:l0},G_{l1:l0})$
- where $i_1 \geq i_0,\ j_1 \geq j_0,\ k_1 \geq k_0,\ l_1 \geq l_0$
- Produces 3 pairs of outputs:
- $(P_{j1:i0},G_{j1:i0}),(P_{k1:i0},G_{k1:i0}),(P_{l1:i0},G_{l1:i0})$
- where $i_1 \geq j_0-1,\ j_1 \geq k_0-1,\ k_1 \geq l_0-1$
- Allows overlap among input subgroups



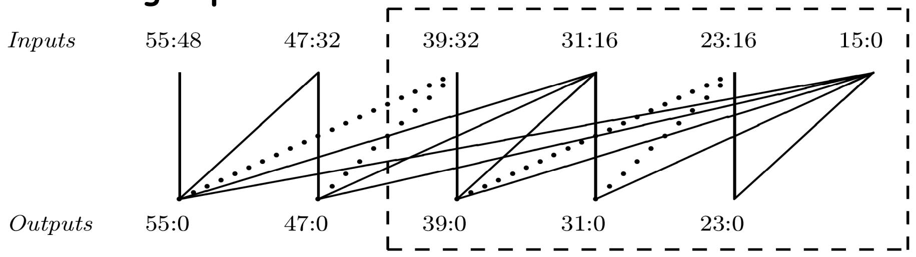
Carry Tree

- **First level** - 14 MCCs calculating $(P_{3:0},G_{3:0}),(P_{7:4},G_{7:4}),\ldots,(P_{55:52},G_{55:52})$
  - * only outputs $P_{3:0}$ and $G_{3:0}$ are utilized
- **Second level**: each MCC generates 2 pairs $(P_{3:0},\ G_{3:0}),(P_{1:0},\ G_{1:0})$
- Providing $(P_{7:0},G_{7:0}),(P_{15:0},G_{15:0}),(P_{23:16},G_{23:16}),(P_{31:16},G_{31:16}),(P_{39:32},G_{39:32}),(P_{47:32},G_{47:32}),(P_{55:48},G_{55:48})$
- Generates c_8 & c_{16} - $G_{7:0}$ & $G_{15:0}$
- c_0 is incorporated into $(P_{3:0},\ G_{3:0})$

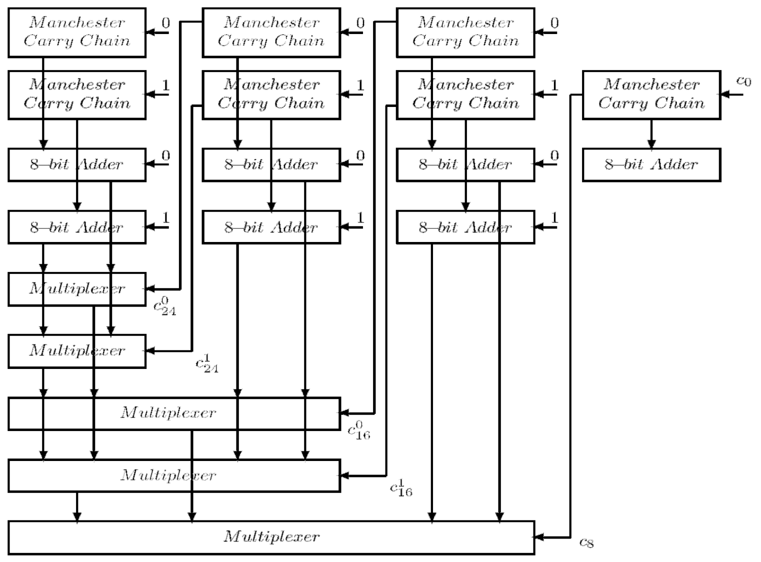


Third level - Two MCCs Sufficient

- One for dashed box generating c_{24}, c_{32} and c_{40}
- Second MCC for 2 remaining outputs with inputs 55:48, 47:32, 31:16 and 15:0 generating c_{48} and c_{56}
- MCC in dashed box must implement 2 dotted lines from 23:16 - required for generating 23:0
- Above implementation of adder not unique
  - * does not necessarily minimize overall execution time
- **Alternate implementations**: variable size of carry-select groups and of MCCs at different levels of tree



A Schematic Diagram of a 32-bit Hybrid Adder

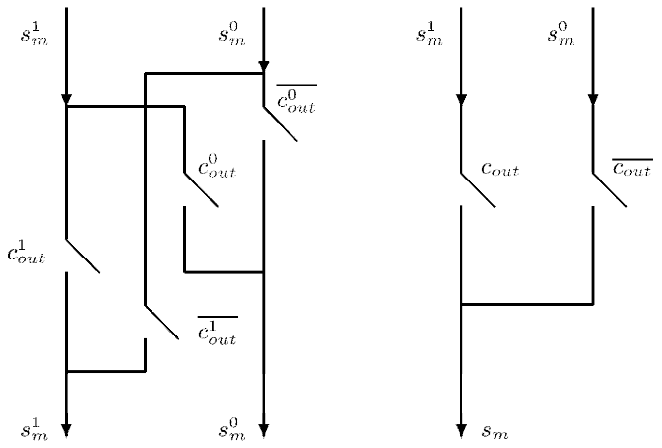


Grouping of Bits in a 64-bit Adder

- 64 bits divided into two sets of 32 bits, each set further divided into 4 groups of 8 bits
- For every group of 8 bits - 2 sets of conditional sum outputs generated separately
- Two most significant groups combined into group of size 16
- Further combined with next group of 8 to form group of 24 bits and so on
  - * principle of conditional-sum addition
  - * However, the way input carries for basic 8-bit groups are generated is different
- MCC generates P_m, G_m and K_m and C^0_{out} ,C^1_{out} for assumed incoming carries of 0 and 1
- Conditional carry-out signals control multiplexers

Dual and Regular Multiplexer

- Two sets of dual multiplexers (of size 8 and 16)
- Single regular multiplexer of size 24



High-Order Half of 64-bit Adder

- Similar structure but incoming carry c_{32} calculated by separate carry-look-ahead circuit
- Inputs are conditional carry-out signals generated by 4 MCCs
- Allows operation of high-order half to overlap operation of low-order half
- Summary: combines variants of 3 different techniques for fast addition: Manchester carry generation, carry-select, conditional-sum
- Other designs of hybrid adders exist - e.g., groups with unequal number of bits
- "Optimality" of hybrid adders depends on technology and delay parameters

Carry-Save Adders (CSAs)

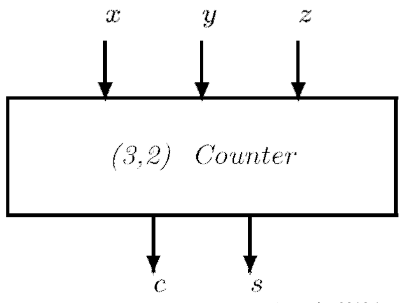
- 3 or more operands added simultaneously (e.g., in multiplication) using 2-operand adders
- Time-consuming carry-propagation must be repeated several times: k operands - k-1 propagations
- Techniques for lowering this penalty exist - most commonly used - carry-save addition
- Carry propagates only in last step - other steps generate partial sum and sequence of carries
- Basic CSA accepts 3 n-bit operands; generates 2 n-bit results: n-bit partial sum, n-bit carry
- Second CSA accepts the 2 sequences and another input operand, generates new partial sum and carry
- CSA reduces number of operands to be added from 3 to 2 without carry propagation

Implementing Carry Save Adders

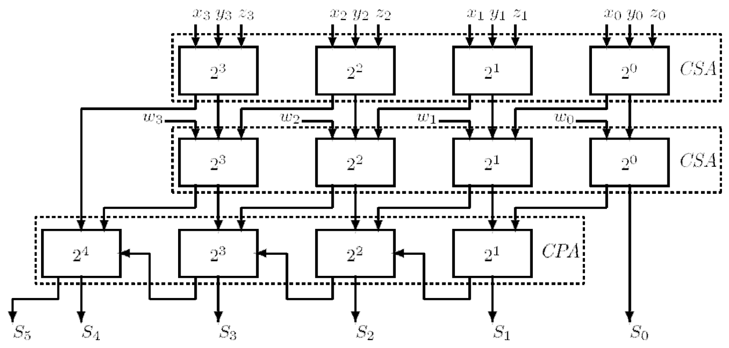
- Simplest implementation - full adder (FA) with 3 inputs x,y,z
- x+y+z=2c+s (s,c - sum and carry outputs)

$$s = (x + y + z) \bmod 2 \qquad \text{and} \qquad c = \frac{(x + y + z) - s}{2}.$$

- Outputs - weighted binary representation of number of 1's in inputs
- FA called a (3,2) counter
- n-bit CSA: n (3,2) counters in parallel with no carry links

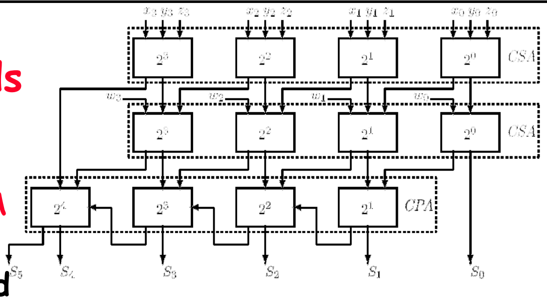


Carry-Save Adder for four 4-bit Operands



* Upper 2 levels - 4-bit CSAs
* 3rd level - 4-bit carry-propagating adder (CPA)
* Ripple-carry adder - can be replaced by a carry-look-ahead adder or any other fast CPA
* Partial sum bits and carry bits interconnected to guarantee that only bits having same weight are added by any (3,2) counter

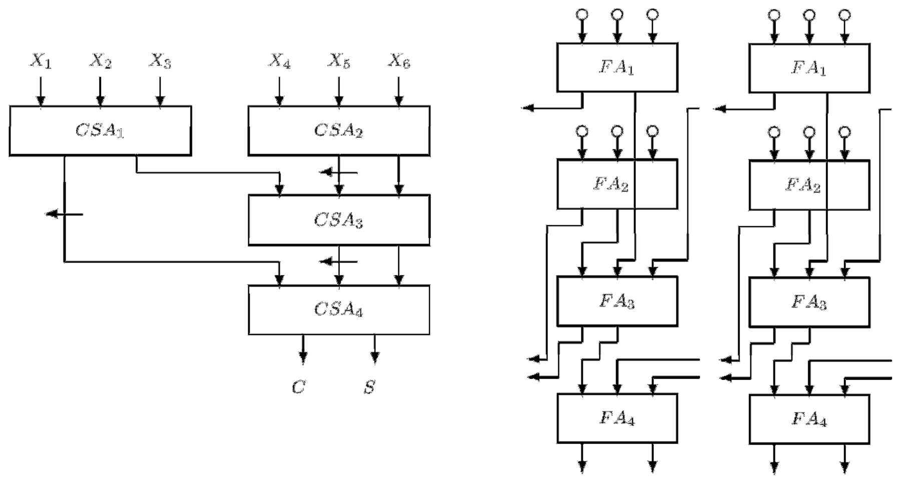
Adding k Operands



- (k-2) CSAs + one CPA
- If CSAs arranged in cascade - time to add k operands is $(k-2)T_{CSA} + T_{CPA}$
- T_{CPA} ; T_{CSA} - operation time of CPA ; CSA
- Δ_G ; Δ_{FA} delay of a single gate ; full adder
- $T_{CSA} = \Delta_{FA} \geq 2\ \Delta_G$
- Sum of k operands of size n bits each can be as large as $k(2^n-1)$
- Final addition result may reach a length of $n+\lceil \log_2 k \rceil$ bits

Six-operand Wallace Tree

- Better organization for CSAs - faster operation time



Number of Levels in Wallace Tree

- Number of operands reduced by a factor of 2/3 at each level - $k \cdot (\frac{2}{3})^l \leq 2$ (l - number of levels)
- Consequently, l = $\text{Number of levels} \approx \frac{log\ (k/2)}{log\ (3/2)}$
- Only an estimate of l - number of operands at each level must be an integer
- N_i - number of operands at level i
- N_{i+1} - at most $\lfloor 3/2\ N_i \rfloor$ ($\lfloor x \rfloor$ - largest integer smaller than or equal to x)
- Bottom level (0) has 2 - maximum at level 1 is 3 - maximum at level 2 is $\lfloor 9/2 \rfloor$ =4
- Resulting sequence: 2,3,4,6,9,13,19,28,…
- For 5 operands - still 3 levels

Number of Levels in a CSA Tree for k operands

Number of operands	Number of levels
3	1
4	2
$5 \leq k \leq 6$	3
$7 \leq k \leq 9$	4
$10 \leq k \leq 13$	5
$14 \leq k \leq 19$	6
$20 \leq k \leq 28$	7
$29 \leq k \leq 42$	8
$43 \leq k \leq 63$	9

- Example: k=12 - 5 levels - delay of $5T_{CSA}$ instead of $10T_{CSA}$ in a linear cascade of 10 CSAs

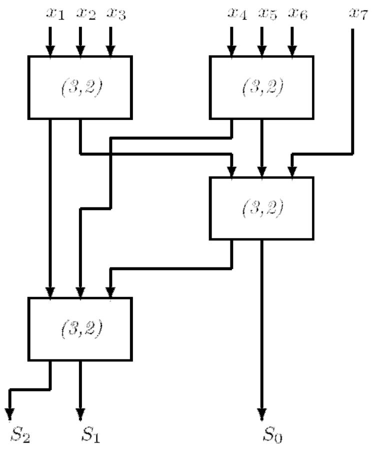
Most Economical Implementation (Fewer CSAs)

- Achieved when number of operands is element of 3,4,6,9,13,19,28,…
- If given number of operands, k, not in sequence - use only enough CSAs to reduce k to closest (smaller than k) element
- Example: k=27, use 8 CSAs (24 inputs) rather than 9, in top level - number of operands in next level is 8×2+3=19
- Remaining part of tree will follow the series

Number of operands	Number of levels
3	1
4	2
$5 \leq k \leq 6$	3
$7 \leq k \leq 9$	4
$10 \leq k \leq 13$	5
$14 \leq k \leq 19$	6
$20 \leq k \leq 28$	7
$29 \leq k \leq 42$	8
$43 \leq k \leq 63$	9

(7,3) and Other Counters

- (7,3) counter: 3 outputs - represent number of 1's in 7 inputs
- Another example: (15,4) counter
- In general: (k,m) counter - k and m satisfy $2^m-1 \geq k$ or $m \geq \lfloor \log_2(k+1) \rfloor$
- (7,3) counter using (3,2) counters:
- Requires 4 (3,2)'s in 3 levels - no speed-up



(7,3) Counters

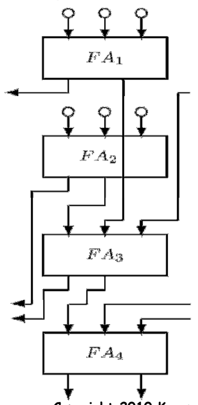
- (7,3) can be implemented as a multilevel circuit - may have smaller delay
- Number of interconnections affects silicon area - (7,3) preferrable to (3,2)
  - * (7,3) has 10 connections and removes 4 bits
  - * (3,2) has 5 connections and removes only 1 bit
- Another implementation of (7,3) - ROM of size 2^7 x 3= 128 x 3 bits
- Access time of ROM unlikely to be small enough
- Speed-up may be achieved for ROM implementation of (k,m) counter with higher values of k

Avoiding Second Level of Counters

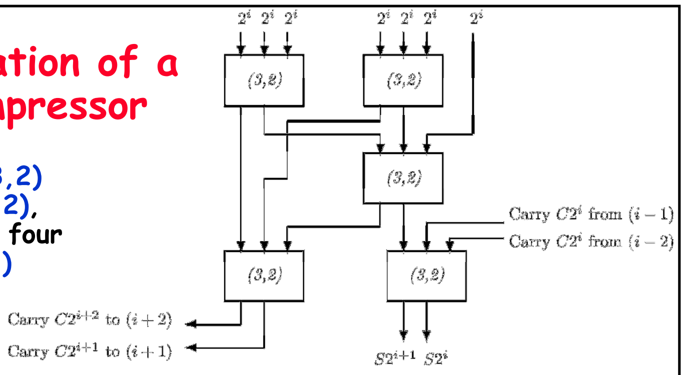
- Several (7,3) counters (in parallel) are used to add 7 operands - 3 results obtained
- Second level of (3,2) counters needed to reduce the 3 to 2 results (sum and carry) added by a CPA
- Similarly - when (15,4) or more complex counters are used - more than two results generated
- In some cases - additional level of counters can be combined with first level - more convenient implementation
- When combining a (7,3) counter with a (3,2) counter - combined counter called a (7;2) compressor

(k;m) Compressor

- Variant of a counter with k primary inputs, all of weight 2^i , and m primary outputs of weights 2^i ,2^{i+1} ,...,2^{i+m-1}
- Compressor has several incoming carries of weight 2^i from previous compressors, and several outgoing carries of weights 2^{i+1} and up
- Trivial example of a (6;2) compressor:
- All outgoing carries have weight 2^{i+1}
- Number of outgoing carries = number of incoming carries = k-3 (in general)



Implementation of a (7;2) Compressor



2^i 2^i 2^i
2^i 2^i 2^i
2^i
(3,2)
(3,2)
(3,2)
(3,2)
(3,2)
Carry $C2^i$ from $(i-1)$
Carry $C2^i$ from $(i-2)$
Carry $C2^{i+2}$ to $(i+2)$
Carry $C2^{i+1}$ to $(i+1)$
$S2^{i+1}$ $S2^i$

* Bottom right (3,2) - additional (3,2), while remaining four - ordinary (7,3) counter

* 7 primary inputs of weight 2^i and 2 carry inputs from columns i-1 and i-2
* 2 primary outputs, $S2^i$ and $S2^{i+1}$, and 2 outgoing carries $C2^{i+1}$, $C2^{i+2}$, to columns i+1 and i+2
* Input carries do not participate in generation of output carries - avoids slow carry-propagation
* Not a (9,4) counter - 2 outputs with same weight
* Above implementation does not offer any speedup
* Multilevel implementation may yield smaller delay as long as outgoing carries remain independent of incoming carries

Multiple-column counters

♦ Generalized parallel counter: add l input columns and produce m-bit output - $(k_{l-1},k_{l-2},\ldots,k_0,m)$

♦ k_i - number of input bits in i-th column with weight 2^i

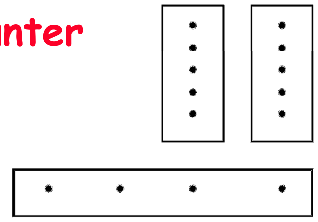
♦ (k,m) counter - a special case

♦ Number of outputs m must satisfy

$$2^m - 1 \geq \sum_{i=0}^{l-1} k_i 2^i$$

♦ If all l columns have same height k - $(k_0=k_1= \ldots =k_{l-1}=k)$ - $2^m - 1 \geq k(2^l - 1)$

Example - (5,5,4) Counter



- k=5, ℓ=2, m=4
- $2^m - 1 = k(2^\ell - 1)$ - all 16 combinations of output bits are useful
- (5,5,4) counters can be used to reduce 5 operands (of any length) to 2 results that can then be added with one CPA
- Length of operands determines number of (5,5,4) counters in parallel
- Reasonable implementation - using ROMs
- For (5,5,4) - 2^{5+5}x4 (=1024x4) ROM

Number of Results of General Counters

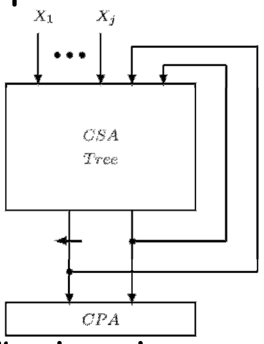
- String of (k,k,…,k,m) counters may generate more than 2 intermediate results
  - requiring additional reduction before CPA
- Number of intermediate results:
- A set of (k,k,…,k,m) counters, with ℓ columns each, produces m-bit outputs at intervals of ℓ bits
- Any column has at most $\lceil m/\ell \rceil$ output bits
- k operands can be reduced to $s = \lceil m/\ell \rceil$ operands
  - If s=2 - a single CPA can generate final sum
  - Otherwise, reduction from s to 2 needed

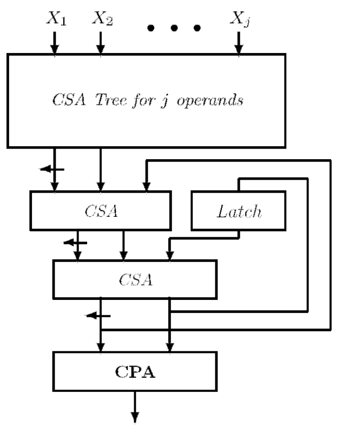
Example

- Number of bits per column in a 2-column counter (k,k,m) is increased beyond 5 - $m \geq 5$ and $s = \lceil m/2 \rceil > 2$
- For k=7, $2^m - 1 \geq 7 \times 3 = 21 \Rightarrow m=5$
- (7,7,5) counters generate s=3 operands - another set of (3,2) counters is needed to reduce number of operands to 2

Reducing Hardware Complexity of CSA Tree

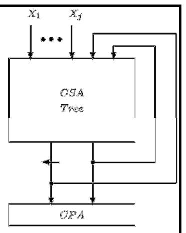
- Design a smaller carry-save tree - use it iteratively
- n operands divided into $\lceil n/j \rceil$ groups of j operands - design a tree for j+2 operands and a CPA



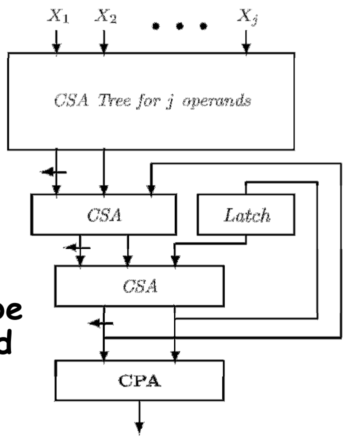


- Feedback paths - must complete first pass through CSA tree before second set of j operands is applied
- Execution slowed down - pipelining not possible

Partial Tree



- Reduced hardware complexity of CSA tree - partial tree
- Two feedback connections prevent pipelining
- Modification - intermediate results of CSA tree connected to bottom level of tree
- Smaller tree with j inputs, 2 separate CSAs, and a set of latches at the bottom
- CSAs and latches form a pipeline stage
- Top CSA tree for j operands can be pipelined too - overall time reduced


X_1
X_2
X_j
CSA Tree for j operands
CSA
Latch
CSA
CPA