UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Digital Computer Arithmetic
ECE 666

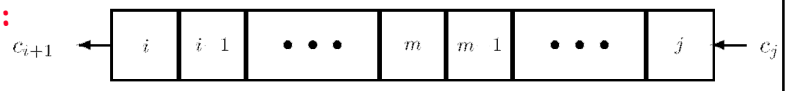
Part 5b
Fast Addition - II

Israel Koren

Copyright 2010 Koren

Carry-Look-Ahead Addition Revisited

- Generalizing equations for fast adders - carry-look-ahead, carry-select and carry-skip
- Notation:



  - $P_{i:j}$ - group-propagated carry
  - $G_{i:j}$ - group-generated carry

  for group of bit positions $i, i-1, \ldots, j$ ($i \geq j$)
- $P_{i:j}=1$ when incoming carry into least significant position j, c_j, is allowed to propagate through all $i-j+1$ positions
- $G_{i:j}=1$ when carry is generated in at least one of positions j to i and propagates to $i+1$, ($c_{i+1} = 1$)
  - Generalization of previous equations
  - Special case - single bit-position functions P_i and G_i

Copyright 2010 Koren

Group-Carry Functions

- Boolean equations

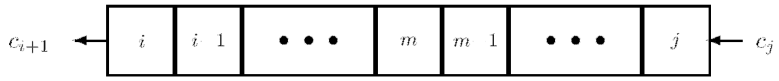
$$P_{i:j} = \begin{cases} P_i & \text{if } i = j \\ P_i \cdot P_{i-1:j} & \text{if } i > j \end{cases}$$

$$G_{i:j} = \begin{cases} G_i & \text{if } i = j \\ G_i + P_i \cdot G_{i-1:j} & \text{if } i > j \end{cases}$$

- Pi:i ≡ Pi ; Gi:i ≡ Gi
- Recursive equations can be generalized (i ≥ m ≥ j+1)

$$\begin{aligned} P_{i:j} &= P_{i:m} \cdot P_{m-1:j}, \\ G_{i:j} &= G_{i:m} + P_{i:m} \cdot G_{m-1:j} \end{aligned}$$

- Same generalization used for deriving section-carry propagate and generate functions - P** and G**
- Proof - induction on m



Copyright 2010 Koren

Fundamental Carry Operator

- Boolean operator - fundamental carry operator - o

$$(P,G) \circ (\widetilde{P},\widetilde{G}) = (P \cdot \widetilde{P}, G + P \cdot \widetilde{G})$$

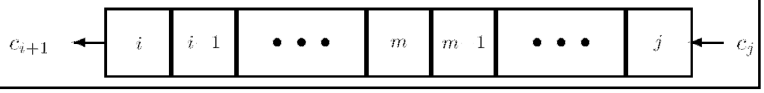
- Using the operator o
- (Pi:j,Gi:j)=(Pi:m,Gi:m) o (Pm-1:j,Gm-1:j) (i≥m≥j+1)
- Operation is associative

$$((P_{i:m},G_{i:m}) \circ (P_{m-1:v},G_{m-1:v})) \circ (P_{v-1:j},G_{v-1:j})$$
$$= (P_{i:m},G_{i:m}) \circ ((P_{m-1:v},G_{m-1:v}) \circ (P_{v-1:j},G_{v-1:j}))$$

- Operation is idempotent

$$(P,G) \circ (P,G) = (P \cdot P, G + P \cdot G) = (P,G)$$

- Therefore
  (Pi:j,Gi:j)=(Pi:m,Gi:m) o (Pv:j,Gv:j) i≥m ; v≥j ; v≥m-1



Individual Bit Carry & Sum

- Group carries $P_{i:j}$ and $G_{i:j}$ calculated from subgroup carries - subgroups are of arbitrary size and may even overlap
- Group and subgroup carries used to calculate individual bit carries c_{i+1}, c_i,…, c_{j+1}, and sum outputs s_i, s_{i-1},…, s_j
- Must take into account "external" carry c_j
- For the mth bit position, $i \geq m \geq j$ -

$$c_m = G_{m-1:j} + P_{m-1:j} \cdot c_j$$

- rewritten as

$$(P_{m-1:j}, G_{m-1:j}) \circ (1, c_j)$$

- If $P_m = x_m \oplus y_m$ then $s_m = c_m \oplus P_m$
- If $P_m = x_m + y_m$ then $s_m = c_m \oplus (x_m \oplus y_m)$

Copyright 2010 Koren

Various Adder Implementations

- Equations can be used to derive various implementations of adders - ripple-carry, carry-look-ahead, carry-select, carry-skip, etc.
- 5-bit ripple-carry adder: All subgroups consist of a single bit position ; computation starts at position 0, proceeds to position 1 and so on

$$(P_4, G_4) \circ \{(P_3, G_3) \circ ((P_2, G_2) \circ [(P_1, G_1) \circ \{(P_0, G_0) \circ (1, c_0)\}])\}$$

- 16-bit carry-look-ahead adder: 4 groups of size 4; ripple-carry among groups

$$(P_{15:12}, G_{15:12}) \circ \{(P_{11:8}, G_{11:8}) \circ [(P_{7:4}, G_{7:4}) \circ \{(P_{3:0}, G_{3:0}) \circ (1, c_0)\}]\}$$

Copyright 2010 Koren

Brent-Kung Adder

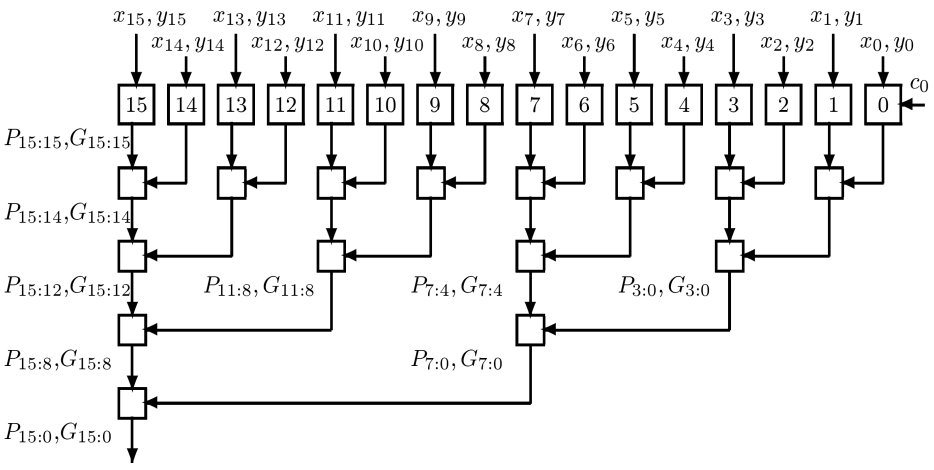
- Variant of carry-look-ahead adder - blocking factor of 2 → very regular layout tree with $\log_2 n$ levels - total area ≈ $n \log_2 n$
- Consider c_{16} - incoming carry at stage 16 in a 17-bit (or more) adder and suppose $G_0 = x_0 y_0 + P_0 c_0$
- The part that generates $(P_{7:0}, G_{7:0})$ corresponds to

$$\begin{aligned}(P_{7:0}, G_{7:0}) &= (P_{7:4}, G_{7:4}) \circ (P_{3:0}, G_{3:0}) \\ &= \{(P_{7:6}, G_{7:6}) \circ (P_{5:4}, G_{5:4})\} \circ \{(P_{3:2}, G_{3:2}) \circ (P_{1:0}, G_{1:0})\} \\ &= \{[(P_7, G_7) \circ (P_6, G_6)] \circ [(P_5, G_5) \circ (P_4, G_4)]\} \\ &\quad \circ \{[(P_3, G_3) \circ (P_2, G_2)] \circ [(P_1, G_1) \circ (P_0, G_0)]\}\end{aligned}$$

- Each line, except c_0, represents two signals - either x_m, y_m or $P_{v:m}, G_{v:m}$

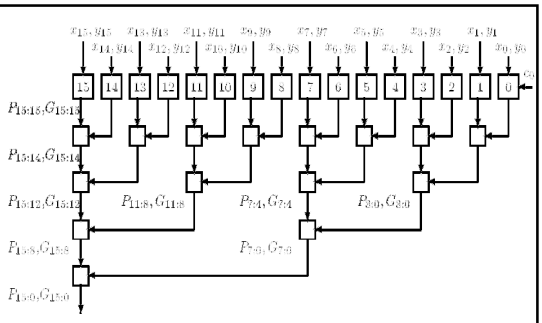
Copyright 2010 Koren

Tree Structure for Calculating C_{16}



Copyright 2010 Koren

Carry Calculation



- Circuits in levels 2 to 5 implement fundamental carry op
- $c_{16}=G_{15:0}$; $P_m=x_m \oplus y_m$ sum: $s_{16}=c_{16} \oplus P_{16}$
- Tree structure also generates carries c_2, c_4 and c_8
- Carry bits for remaining positions can be calculated through extra subtrees that can be added
- Once all carries are known - corresponding sum bits can be computed
- Above - blocking factor = 2
  - Different factors for different levels may lead to more efficient use of space and/or shorter interconnections

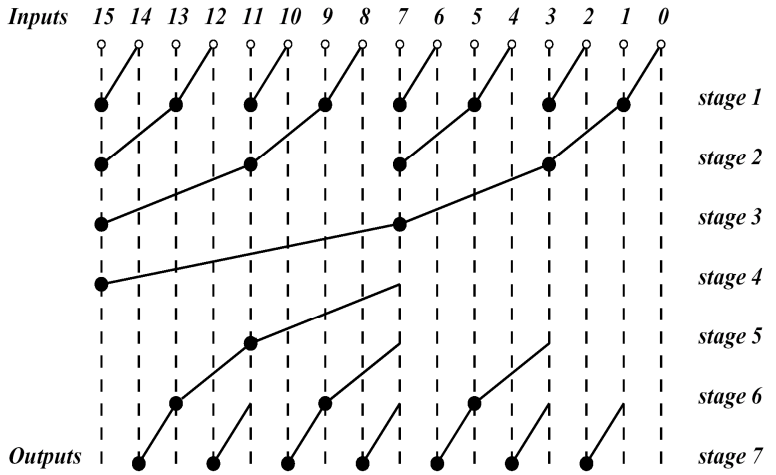
Copyright 2010 Koren

Prefix Adders

- The BK adder is a parallel prefix circuit - a combinational circuit with n inputs $x_1,x_2,\ldots,x_n$ producing outputs $x_1, x_2 \circ x_1, \ldots, x_n \circ x_{n-1} \circ \ldots \circ x_1$
- $\circ$ is an associative binary operation
- First stage of adder generates individual P_i and G_i
- Remaining stages constitute the parallel prefix circuit with fundamental carry operation serving as the $\circ$ associative binary operation
- This part of tree can be designed in different ways

Copyright 2010 Koren

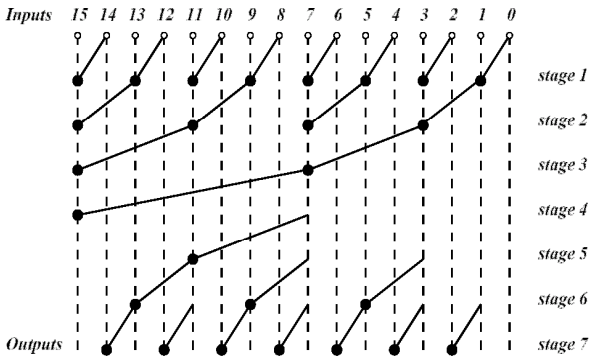
Implementation of the 16-bit Brent-Kung Adder



Copyright 2010 Koren

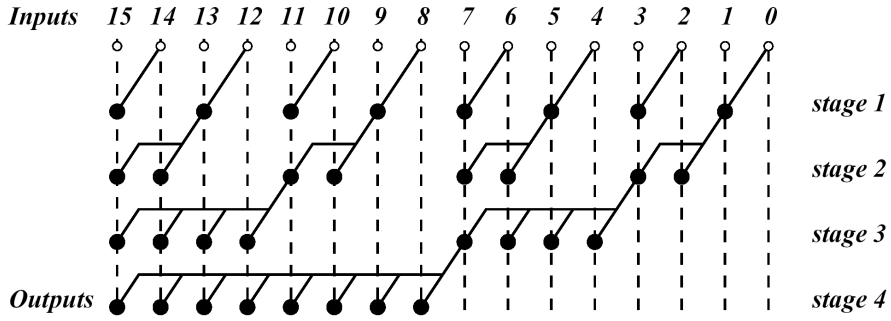
Brent-Kung Parallel Prefix Graph

- Bullets implement the fundamental carry operation - empty circles generate individual Pi and Gi
- Number of stages and total delay - can be reduced by modifying structure of parallel prefix graph
- Min # of stages = $\log_2 n$
  - 4 for n=16
  - For BK parallel prefix graph = $2\log_2 n - 1$



Ladner-Fischer Parallel Prefix Adder

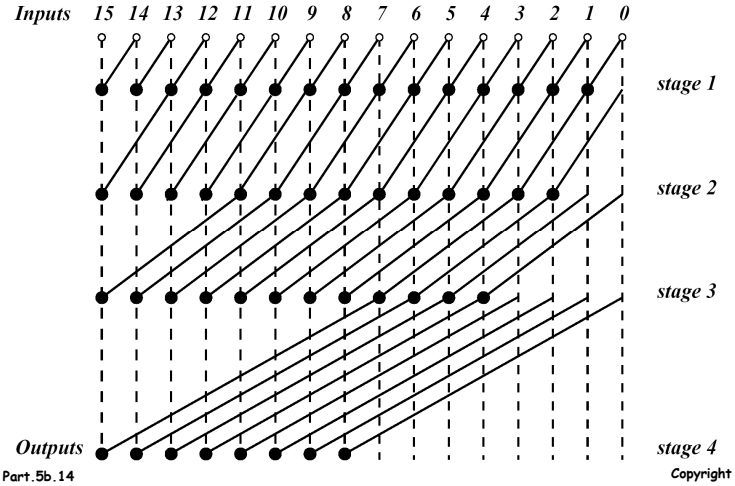
- Implementing a 4-stage parallel prefix graph
- Unlike BK, LF adder employs fundamental carry operators with a fan-out ≥ 2 - blocking factor varies from 2 to n/2
- Fan-out ≤ n/2 requiring buffers - adding to overall delay



Copyright 2010 Koren

Kogge-Stone Parallel Prefix Adder

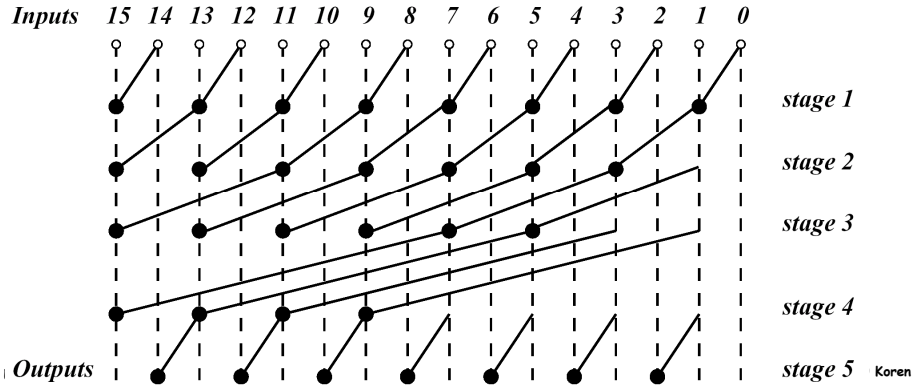
- $\log_2 n$ stages - but lower fan-out
- More lateral wires with long span than BK - requires buffering causing additional delay



Copyright 2010 Koren

Han-Carlson Parallel Prefix Adder

- Other variants - small delay in exchange for high overall area and/or power
  - Compromises between wiring simplicity and overall delay
- A hybrid design combining stages from BK and KS
  - 5 stages - middle 3 resembling KS - wires with shorter span than KS



Koren

Ling Adders

- Variation of carry-look-ahead - simpler version of group-generated carry signal - reduced delay
- Example: A carry-look-ahead adder - groups of size 2 - produces signals $G_{1:0}, P_{1:0}, G_{3:2}, P_{3:2}, \ldots$
- Outgoing carry for position 3 - $c_4 = G_{3:0} = G_{3:2} + P_{3:2}\, G_{1:0}$
- where $G_{3:2} = G_3 + P_3\, G_2$; $G_{1:0} = G_1 + P_1\, G_0$; $P_{3:2} = P_3\, P_2$
- Either assume $c_0 = 0$ or set $G_0 = x_0\, y_0 + P_0\, c_0$
  - Also $P_i = X_i + Y_i$
- $G_{3:0} = G_3 + P_3\, G_2 + P_3\, P_2\, (G_1 + P_1\, G_0)$
- since $G_3 = G_3\, P_3$ - $G_{3:0} = P_3\, H_{3:0}$
  - where $H_{3:0} = H_{3:2} + P_{2:1}\, H_{1:0}$; $H_{3:2} = G_3 + G_2$; $H_{1:0} = G_1 + G_0$
- Note: $P_{2:1}$ used instead of $P_{3:2}$ before

Copyright 2010 Koren

Ling Adders - Cont.

- H - alternative to carry generate G
  - Similar recursive calculation
  - No simple interpretation like G
  - Simpler to calculate
- Example: $H_{3:0} = G_3 + G_2 + P_2P_1 (G_1+G_0)$
- Simplified - $H_{3:0} = G_3 + G_2 + P_2G_1 + P_2P_1G_0$
- While - $G_{3:0} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$
- Smaller maximum fan-in → simpler/faster circuits
- Variations of G have corresponding variations for H
- $G_{3:0} = G_3 + P_3G_{2:0}$ -
  $H_{3:0} = G_3 + T_2H_{2:0}$ where $T_2 = X_2 + Y_2$
- General expression for H -
  $H_{i:0} = G_i + T_{i-1}H_{i-1:0}$ where $T_{i-1} = X_{i-1} + Y_{i-1}$

Copyright 2010 Koren

Calculation of Sum Bits in Ling Adder

- Slightly more involved than for carry-look-ahead
- Example:

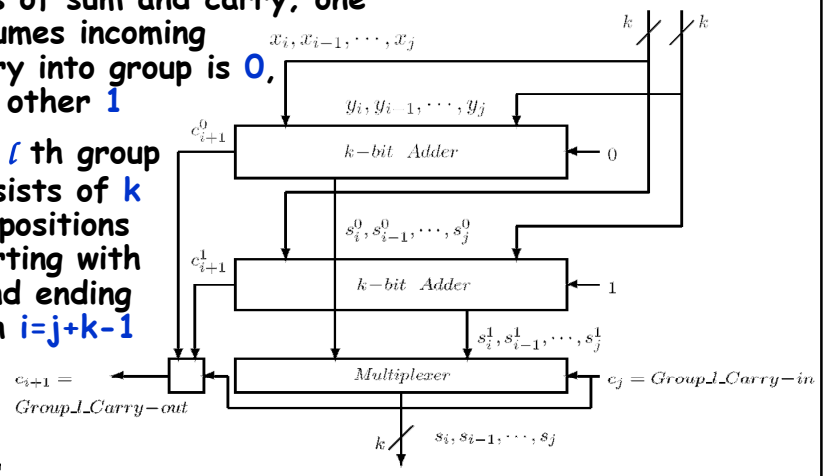
$$\begin{aligned} s_3 &= c_3 \oplus (x_3 \oplus y_3) = (P_2 H_{2:0}) \oplus (x_3 \oplus y_3) \\ &= \overline{H_{2:0}}(x_3 \oplus y_3) + H_{2:0}(P_2 \oplus (x_3 \oplus y_3)) \end{aligned}$$

- Calculation of $H_{2:0}$ faster than c_3 - delay reduced
- Other variations of carry-look-ahead and implementations of Ling adders appear in literature

Copyright 2010 Koren

Carry-Select Adders

- n bits divided into non-overlapping groups of possibly different lengths - similar to conditional-sum adder
- Each group generates two sets of sum and carry; one assumes incoming carry into group is 0, the other 1
- the i th group consists of k bit positions starting with j and ending with i=j+k-1



Carry-Select Adder - Equations

- Outputs of group: sum bits S_i, S_{i-1}, … , S_j and group outgoing carry C_{i+1}

$$s_m = s^0_m \cdot \overline{c_j} + s^1_m \cdot c_j \; ; \qquad m = j, j+1, \cdots, i$$

$$c_{i+1} = c^0_{i+1} \cdot \overline{c_j} + c^1_{i+1} \cdot c_j$$

- Same notation as for conditional-sum adder
- Two sets of outputs can be calculated in a ripple-carry manner

Copyright 2010 Koren

Detailed Expressions

- For bit m - calculate carries from $G^0_{m-1:j}$; $G^1_{m-1:j}$

$$(P_{m-1:j}, G^0_{m-1:j}) = (P_{m-1}, G_{m-1}) \circ (P_{m-2}, G_{m-2}) \circ \cdots \circ (P_j, G_j)$$

$$(P_{m-1:j}, G^1_{m-1:j}) = (P_{m-1:j}, G^0_{m-1:j}) \circ (1,1) = (P_{m-1:j}, G^0_{m-1:j} + P_{m-1:j})$$

- $P_{m-1:j}$ has no superscript - independent of incoming carry
- Once individual carries are calculated - corresponding sum bits are

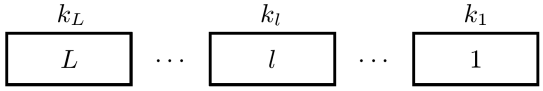
$$s^0_m = c^0_m \oplus P_m \quad \text{and} \quad s^1_m = c^1_m \oplus P_m$$

- Since c^0_{i+1} implies c^1_{i+1} - $c_{i+1} = c^0_{i+1} + c^1_{i+1} \cdot c_j$
- Group sizes can be either different or all equal to k, with possibly one group smaller

Copyright 2010 Koren

Different Group Sizes

- Notations:
  - Size of group l - k_l
  - L - number of groups
  - ΔG - delay of a single gate



- k_l chosen so that delay of ripple-carry within group and delay of carry-select chain from group 1 to l are equal
- Actual delays depend on technology and implementation
- Example: Two-level gate implementation of MUX
  - Delay of carry-select chain through preceding l-1 groups - $(l-1)2\Delta G$
  - Delay of ripple-carry in l th group - $k_l 2\Delta G$
- Equalizing the two - $k_l = l - 1$ with $k_l \geq 1$; $l = 1, 2, \ldots, L$

Copyright 2010 Koren

Different Group Sizes - Cont.

- Resulting group sizes - 1, 1, 2, 3, ...
- Sum of group sizes $\geq n$
- $1+L(L-1)/2 \geq n \rightarrow L(L-1) \geq 2(n-1)$
- Size of largest group and execution time of carry-select adder are of the order of $\sqrt{n}$
- Example: $n=32$, 9 groups required - one possible choice for sizes: 1, 1, 2, 3, 4, 5, 6, 7 & 3
- Total carry propagation time is $18\Delta_G$, instead of $62\Delta_G$ for ripple-carry adder
- If sizes of L groups are equal - carry-select chain (i.e., generating Group Carry-Out from Group Carry-In) not necessarily ripple-carry type
- Single or multiple-level carry-look-ahead can be used

Copyright 2010 Koren

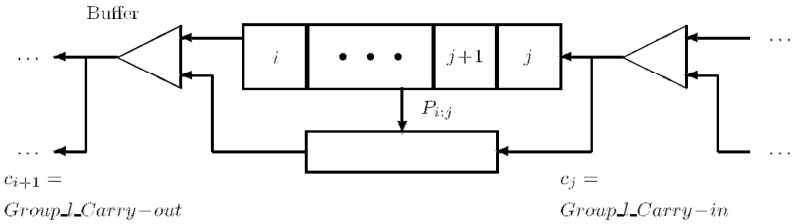
Carry-Skip Adders

- Reduces time needed to propagate carry by skipping over groups of consecutive adder stages
- Generalizes idea behind Manchester Adder
- Illustrates dependence of "optimal" algorithm for addition on available technology
  - Known for many years, only recently became popular
- In VLSI - speed comparable to carry look-ahead (for commonly used word lengths - not asymptotically)
- Requires less chip area and consumes less power
- Based on following observation:
- Carry propagation process can skip any adder stage for which $x_m \neq y_m$ (or, $P_m = x_m \oplus y_m = 1$)
- Several consecutive stages can be skipped if all satisfy $x_m \neq y_m$

Copyright 2010 Koren

Carry-Skip Adder - Structure

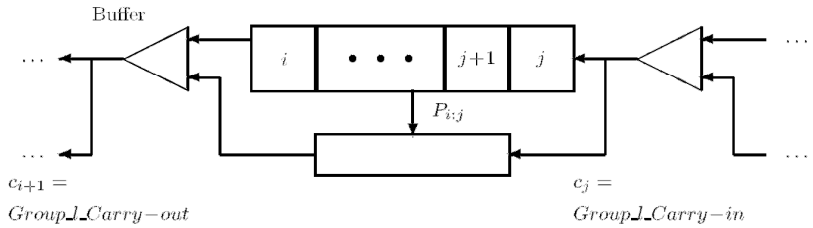
- n stages divided into groups of consecutive stages with simple ripple-carry used in each group
- Group generates a group-carry-propagate signal that equals 1 if for all internal stages $P_m=1$
- Signal allows an incoming carry into group to "skip" all stages within group and generate a group-carry-out
- Group ℓ consists of k bit positions j,j+1,…,j+k-1(=i)



Copyright 2010 Koren

Structure - Cont.

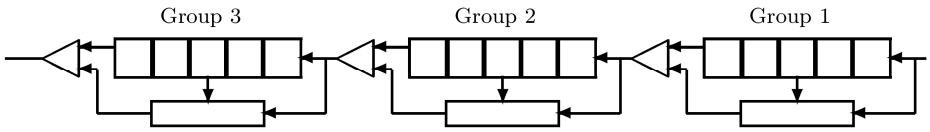
- Group_ℓ_Carry-out = $G_{i:j}$ + $P_{i:j}$ Group_ℓ_Carry-in
- $G_{i:j}$ = 1 when a carry is generated internal to group and allowed to propagate through all remaining bit positions including i
- $P_{i:j}$ = 1 when k=i-j+1 bit positions allow incoming carry cj to propagate to next position i+1
- Buffers realize the OR operation



Copyright 2010 Koren

Example - 15-bit carry-skip adder

- Consisting of 3 groups of size 5 each
- $P_{i:j}$ for all groups can be generated simultaneously allowing a fast skip of groups which satisfy $P_{i:j}=1$



Copyright 2010 Koren

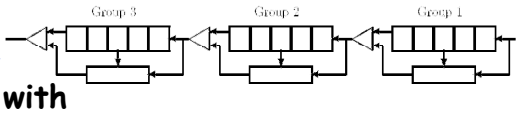
Determining Optimal Group Size k

- Assumption: Groups have equal size k - n/k integer
- k selected to minimize time for longest carry-propagation chain
- Notations:
  - tr - carry-ripple time through a single stage
  - ts(k) - time to skip a group of size k (for most implementations - independent of k)
  - tb - delay of buffer (implements OR) between two groups
  - Tcarry - overall carry-propagation time - occurs when a carry is generated in stage 0 and propagates to stage n-1
- Carry will ripple through stages 1,2, … ,k-1 within group 1, skip groups 2,3, … , (n/k-1), then ripple through group n/k

Copyright 2010 Koren

Determining Optimal k - Cont.

- $T_{carry}=(k-1)t_r+t_b+(n/k-2)(t_s+t_b)+(k-1)t_r$
- Example - two-level gate implementation used for ripple-carry and carry-skip circuits
  - $t_r = t_s+t_b=2\Delta_G$
  - $T_{carry}=(4k+2n/k-7)\ \Delta_G$



- Differentiating T_{carry} with respect to k and equating to 0 -

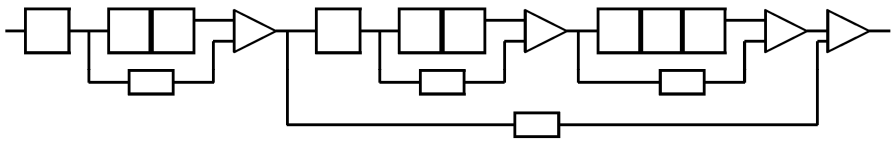
$$k_{opt} = \sqrt{n/2}$$

- Group size and carry propagation time proportional to $\sqrt{n}$ - same as for carry-select adder
- Example: n=32, 8 groups of size k_{opt} = 4 is best
- $T_{opt}=25\Delta_G$ instead of $62\Delta_G$ for ripple-carry adder

Copyright 2010 Koren

Further Speedup

- Size of first and last groups smaller than fixed size k - ripple-carry delay through these is reduced
- Size of center groups increased - since skip time is usually independent of group size
- Another approach: add second level to allow skipping two or more groups in one step (more levels possible)
- Algorithms exist for deriving optimal group sizes for different technologies and implementations (i.e., different values of ratio $(t_s+t_b)/t_r$)



Copyright 2010 Koren

Variable-Size Groups

- Unlike equal-sized group case - cannot restrict to analysis of worst case for carry propagation
- This may lead to trivial conclusion: first and last groups consisting of a single stage - remaining n-2 stages constituting a single center group
- Carry generated at the beginning of center group may ripple through all other n-3 stages - becoming the worst case
- Must consider all possible carry chains starting at arbitrary bit position a (with $x_a=y_a$) and stopping at b ($x_b=y_b$) where a new carry chain (independent of previous) may start

Copyright 2010 Koren

Optimizing Different Size Groups

- $k_1, k_2, \ldots, k_L$ - sizes of L groups $\sum_{i=1}^{L} k_i = n$
- General case: Chain starts within group u, ends within group v, skips groups u+1, u+2, … ,v-1
- Worst case - carry generated in first position within u and stops in last position within v
- Overall carry-propagation time is

$$T_{carry}(u,v) = (k_u - 1)\cdot t_r + t_b + \sum_{l=u+1}^{v-1} (t_s(k_l) + t_b) + (k_v - 1)\cdot t_r$$

- Number of groups L and sizes $k_1, k_2, \ldots, k_L$ selected so that longest carry-propagation chain is minimized - $minimize\left[\max_{1\le u\le v\le L} T_{carry}(u,v)\right]$
- Solution algorithms developed - geometrical interpretations or dynamic programming

Copyright 2010 Koren

Optimization - Example

- 32-bit adder with single level carry-skip
- $t_s+t_b=t_r$
- Optimal organization - L=10 groups with sizes $k_1,k_2,\ldots,k_{10} = 1,2,3,4,5,6,5,3,2,1$
- Resulting in $T_{carry} \leq 9\ t_r$
- If $t_r=2\ \Delta_G$ - $T_{carry} \leq 18\ \Delta_G$ instead of $25\ \Delta_G$ in equal-size group case
- Exercise: Show that any two bit positions in any two groups u and v ($1 \leq u \leq v \leq 10$) satisfy $T_{carry}(u,v) \leq 9\ t_r$

Copyright 2010 Koren

Carry-skip vs. Carry-select Adder

- Strategies behind two schemes sound different
- Equations relating group-carry-out with group-carry-in are variations of same basic equation
- Both have execution time proportional to $\sqrt{n}$
- Only details of implementation vary, in particular calculation of sum bits
- Even this difference is reduced when the multiplexing circuitry is merged into summation logic

Copyright 2010 Koren