Chapter 6 Carry-Lookahead Adders

- Unrolling the Carry Recurrence
- Carry-Lookahead Adder Design
- Ling Adder and related Designs
- Carry Determination as Prefix Computation
- Alternative Parallel Prefix Networks
- VLSI Implementation Aspects

Unrolling the Carry Recurrence

$$c_1 = g_0 + p_0 c_0$$

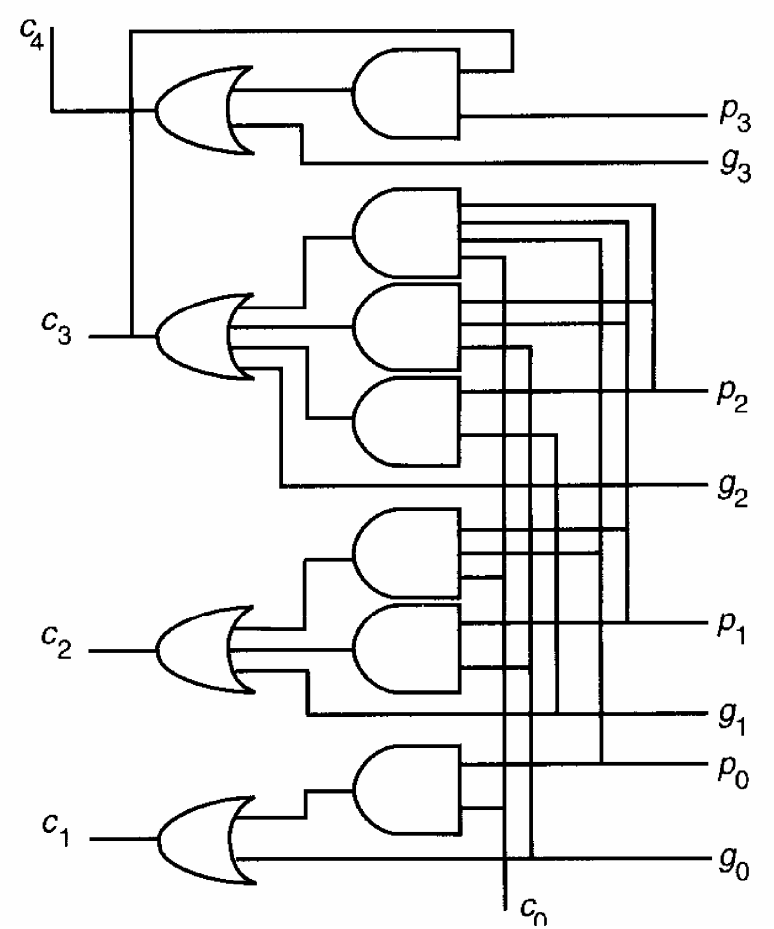
$$c_2 = g_1 + p_1 c_1 = g_1 + p_1(g_0 + p_0 c_0) = g_1 + p_1 g_0 + p_1 p_0 c_0$$

$$c_3 = g_2 + p_2 c_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

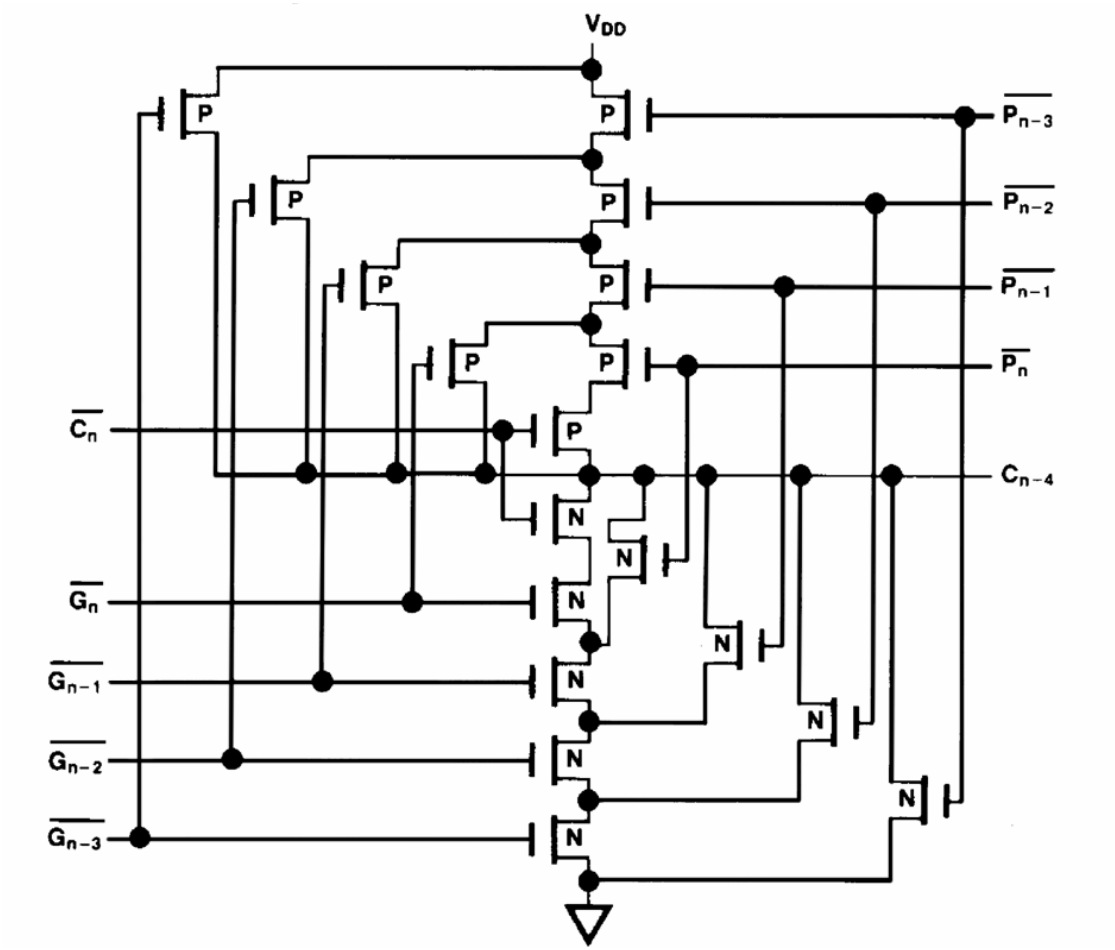
$$c_4 = g_3 + p_3 c_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$

$$\vdots$$

4-bit Full Carry Lookahead



HP Carry Lookahead Circuit



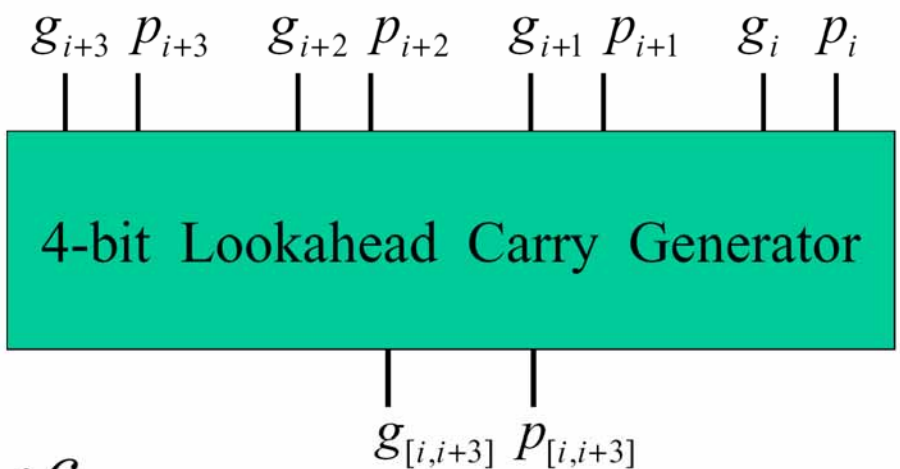
Alternatives to Full Carry Lookahead

- Full carry lookahead is impractical for wide addition
- Tree Networks
  - less circuitry than full lookahead at the expense of increased latency
- Kinds of Tree Networks
  - High-radix addition (radix must be power of 2)
  - Multi-level carry lookahead (technique most used in practice)

4-bit Propagate & Generate

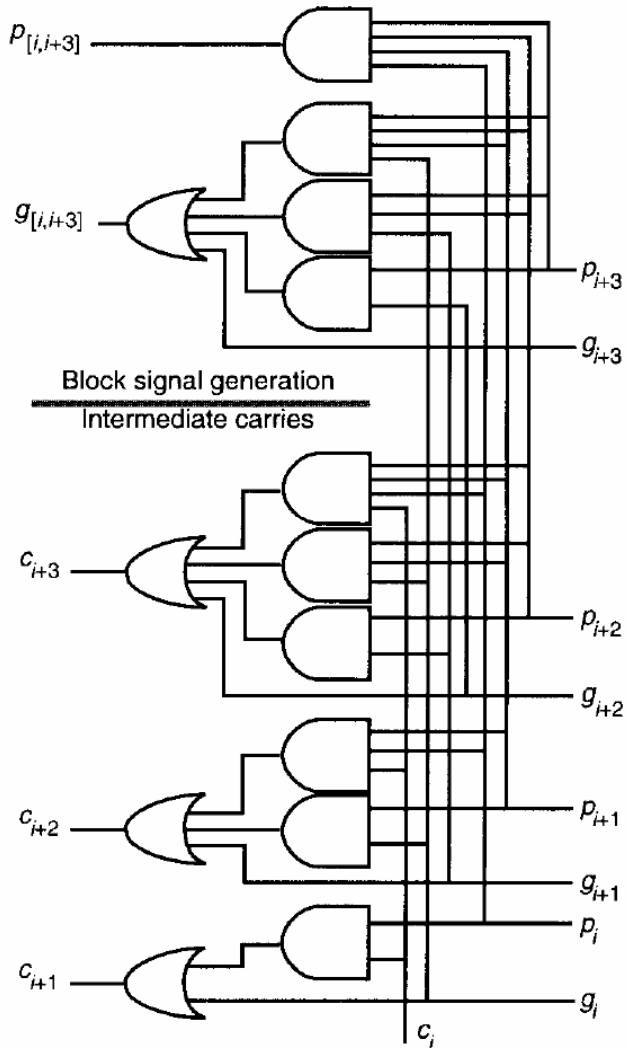
$$g_{[i,i+3]} = g_{i+3} + g_{i+2}p_{i+3} + g_{i+1}p_{i+2}p_{i+3} + g_i p_{i+1}p_{i+2}p_{i+3}$$

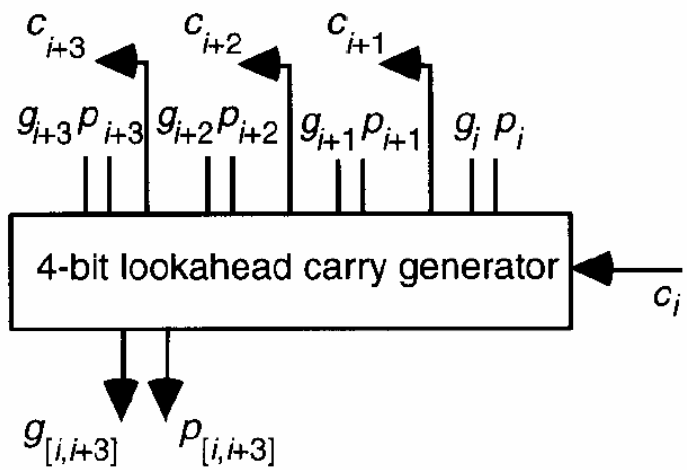
$$p_{[i,i+3]} = p_i p_{i+1} p_{i+2} p_{i+3}$$



$$c_{i+4} = g_{[i,i+3]} + p_{[i,i+3]} \cdot c_i$$

4-bit Lookahead Carry Generator



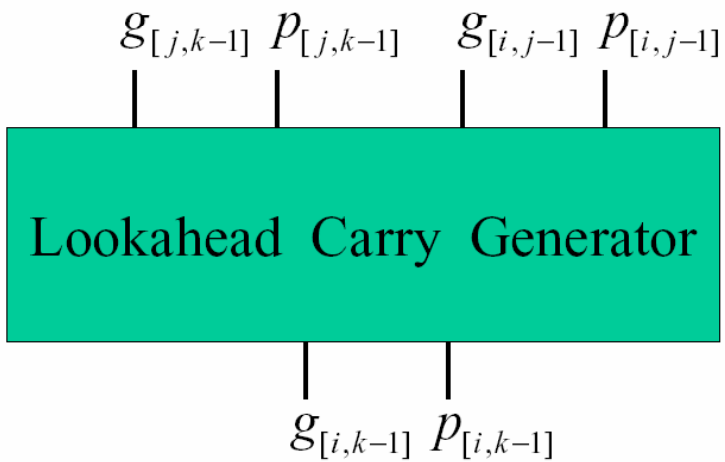


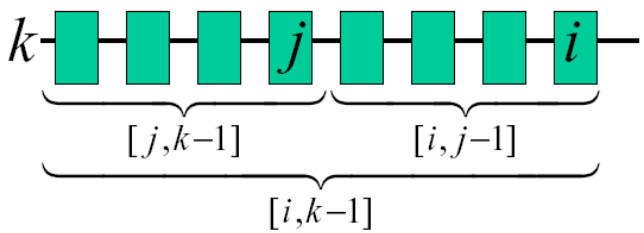
General Propagate & Generate

$$i < j < k$$

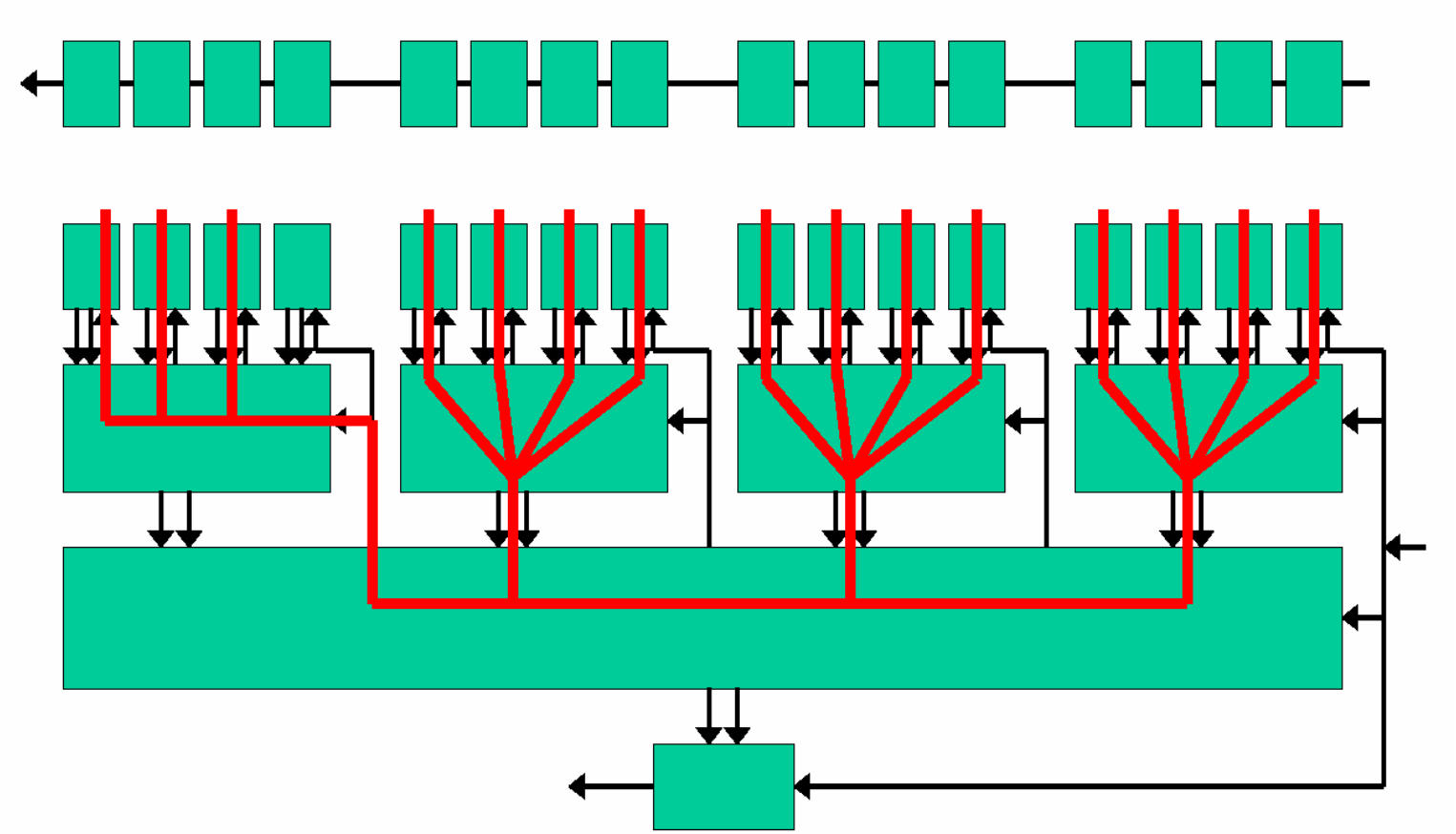
$$g_{[i,k-1]} = g_{[j,k-1]} + g_{[i,j-1]} \cdot p_{[j,k-1]}$$

$$p_{[i,k-1]} = p_{[i,j-1]} \cdot p_{[j,k-1]}$$





16-bit Carry Chain with 2-level Carry Lookahead



What is the worst case delay path ?

Worst Case Latency

- Producing the *g* and *p* for individual bit positions (1 gate delay)
- Producing the *g* and *p* signals for 4-bit blocks (2 gate delays)
- Predicting the carry-in signal *c*4, *c*8, *c*12 for the blocks (2 gate delays)
- Predicting the internal carries within each 4-bit block (2 gate delays)

Worst Case Latency

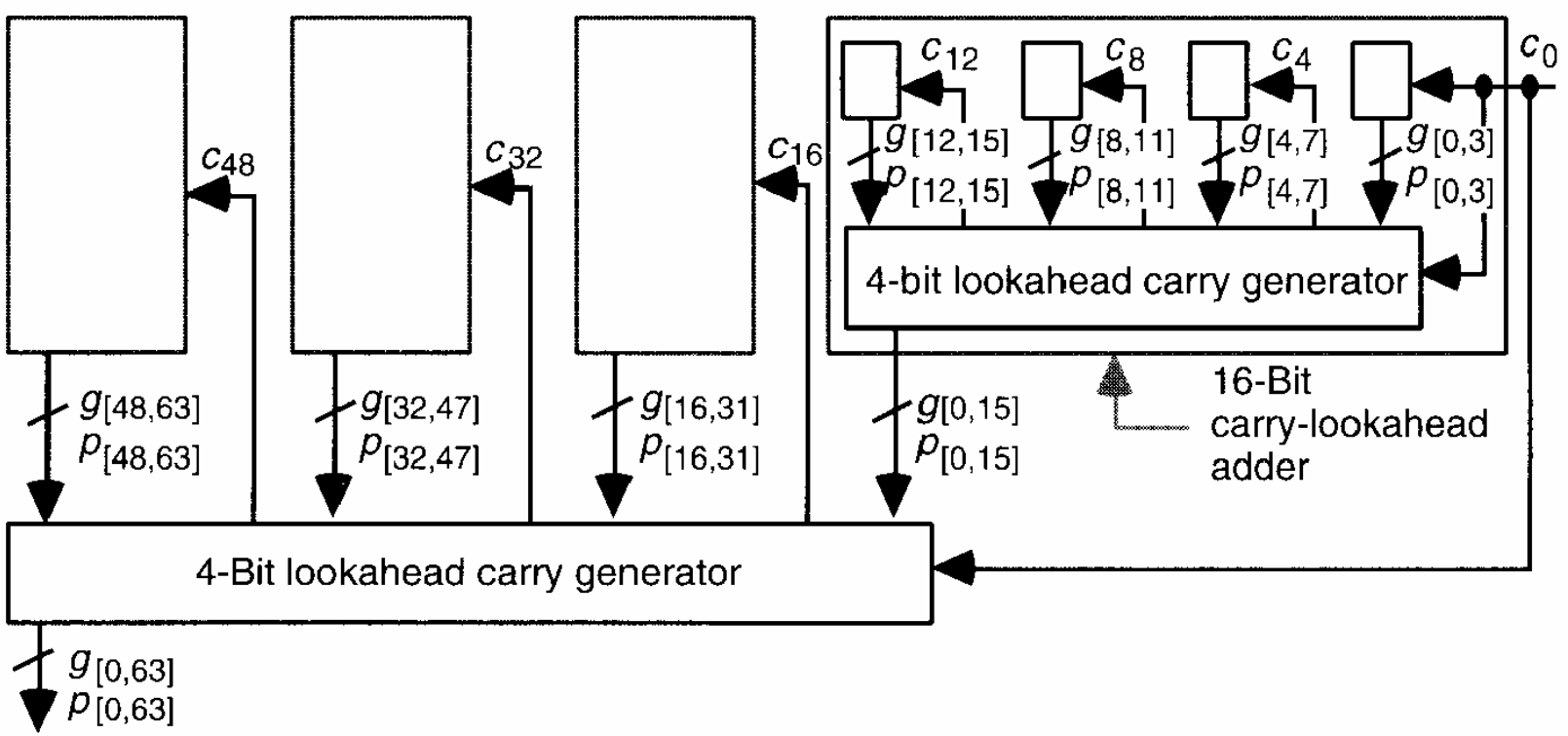
- Computing the sum bits (2 gate delays)
- The delay of a *k*-bit carry-lookahead adder based on 4-bit lookahead blocks is:

$$Time = 4 \log_4 k + 1 \text{ gate delays}$$

Final $c_{\text{out}} = c_k$

- Last carry is not used to compute any sums
- Needed in many situations
  - Overflow computation, for example
- Three ways to compute it:
  - $c_k = g_{[0,k-1]} + c_0 p_{[0,k-1]}$
  - $c_k = g_{k-1} + c_{k-1} \underline{p}_{k-1}$
  - $c_k = x_{k-1} y_{k-1} + \overline{s}_{k-1} (x_{k-1} + y_{k-1})$

64-bit Carry Lookahead Adder



Ling Adder [1981]

$$c_i = g_{i-1} + c_{i-1}p_{i-1} = g_{i-1} + c_{i-1}t_{i-1} =$$
$$= g_{i-1} + g_{i-2}t_{i-1} + g_{i-3}t_{i-2}t_{i-1} + g_{i-4}t_{i-3}t_{i-2}t_{i-1} + c_{i-4}t_{i-4}t_{i-3}t_{i-2}t_{i-1}$$

Ling's idea was to propagate $h_i = c_i + c_{i-1}$ instead of c_i

$$h_i = g_{i-1} + g_{i-2} + g_{i-3}t_{i-2} + g_{i-4}t_{i-3}t_{i-2} + h_{i-4}t_{i-4}t_{i-3}t_{i-2}$$

The carry chain is somewhat simpler, however, the sum equation is slightly more complex:

$$s_i = (t_i \oplus h_{i+1}) + h_i g_i t_{i-1}$$

Parallel Prefix Computations

The *parallel prefix* problem is:

Given:

1. Inputs: $x_0, x_1, x_2 \ldots, x_{k-1}$, and
2. An associative (but not necessarily commutative) operator : +

Compute:

$$x_0$$
$$x_0 + x_1$$
$$x_0 + x_1 + x_2$$
$$\vdots$$
$$x_0 + x_1 + x_2 + \cdots + x_{k-1}$$

Carry Computation is a Parallel Prefix Computation

$$\text{Inputs}: (g_0, p_0), (g_1, p_1), (g_2, p_2), \cdots, (g_{k-1}, p_{k-1})$$

$$\text{Operator}: \quad \notcirc$$

$$(g, p) = (g', p') \notcirc (g'', p'') = (g'' + g' \cdot p'', p' \cdot p'')$$

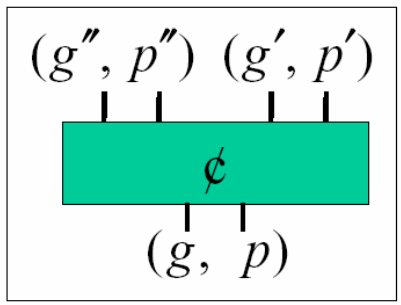
Compute:

$$(g_{[0,0]}, p_{[0,0]}) = (g_0, p_0)$$

$$(g_{[0,1]}, p_{[0,1]}) = (g_1, p_1) \notcirc (g_0, p_0)$$

$$\vdots$$

$$(g_{[0,k-1]}, p_{[0,k-1]}) = (g_{k-1}, p_{k-1}) \notcirc \cdots \notcirc (g_1, p_1) \notcirc (g_0, p_0)$$



Overlapping of Operands in ⭘ expression

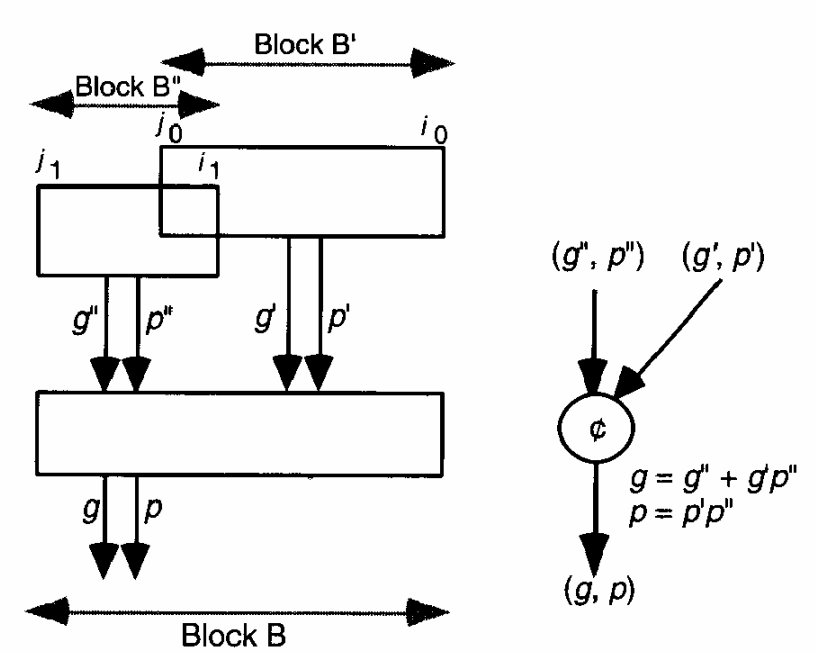
- Overlapping of operands is allowed.
  - Assume $i < j < k < q$
  - $(p_{[0,q]}, g_{[0,q]})$

    $= (p_{[0,j]}, g_{[0,j]}) \circ (p_{[j,q]}, g_{[j,q]})$

    $= (p_{[0,k]}, g_{[0,k]}) \circ (p_{[j,q]}, g_{[j,q]})$

    $= (p_{[0,k]}, g_{[0,k]}) \circ (p_{[j,k]}, g_{[j,k]}) \circ (p_{[j,q]}, g_{[j,q]})$

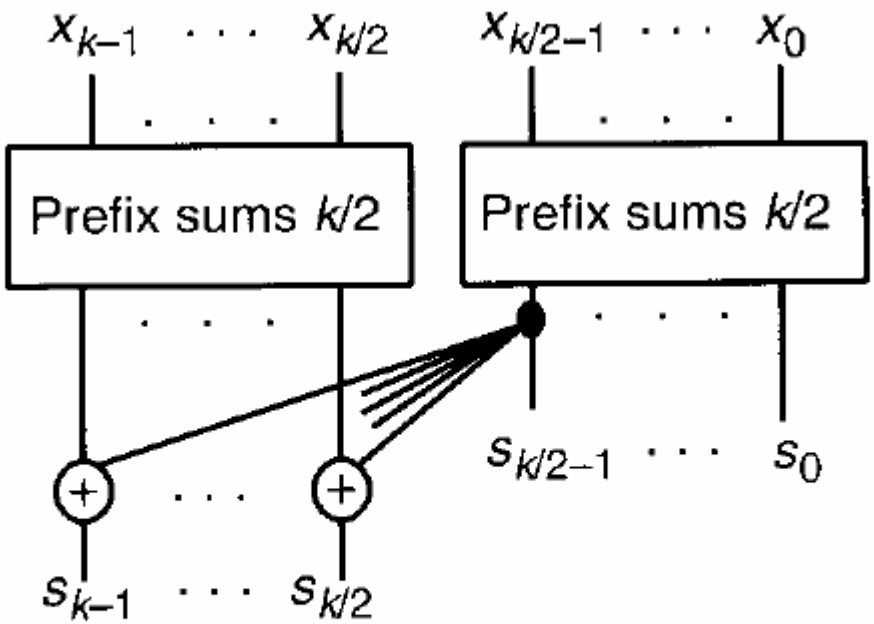
Combining (g, p) of Overlapping Blocks



(g, p) Networks

- Any design for a parallel prefix problem can be adapted to a carry computation network.
- Pairs of inputs can be combined in any way (re-associated according to the associative property) to compute block (g, p) signals.
- (g, p) signals have additional flexibility: overlapping blocks can be combined.

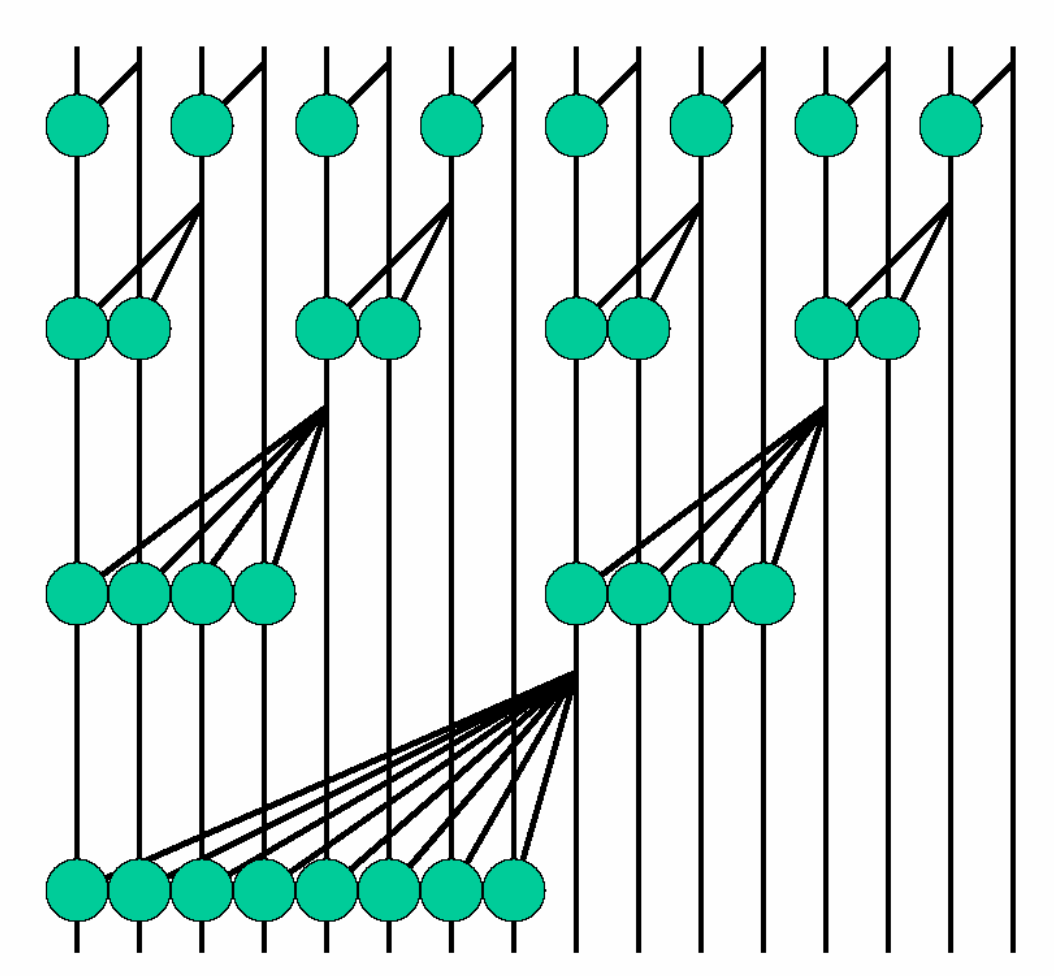
Recursive Prefix Sum Network



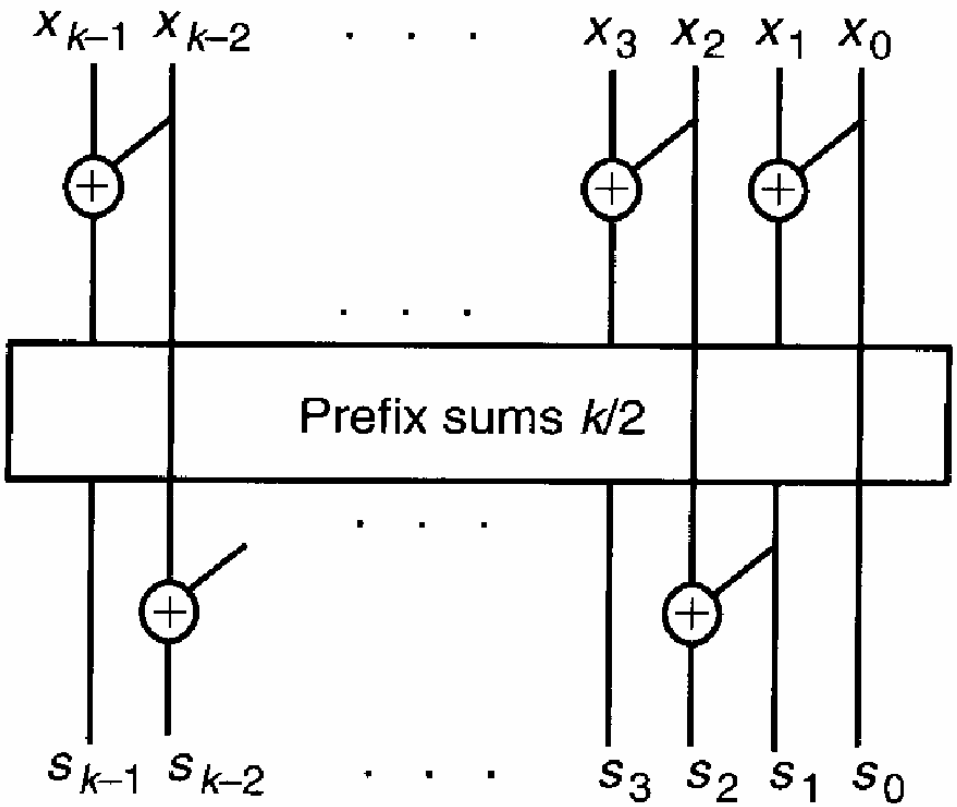
Delay recurrence: $D(k) = D(k/2) + 1 = \log_2 k$

Cost recurrence: $C(k) = 2C(k/2) + k/2 = (k/2)\ \log_2 k$

Divide and Conquer I



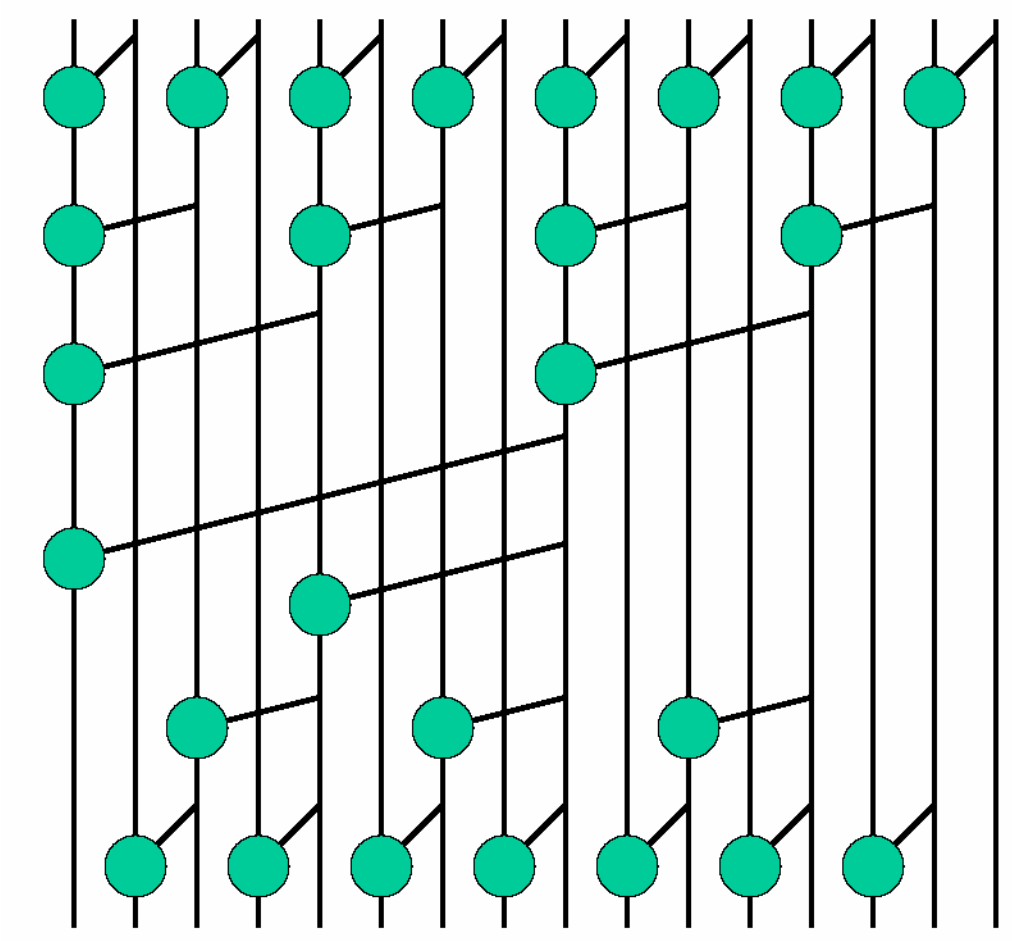
Brent-Kung Parallel Prefix Network



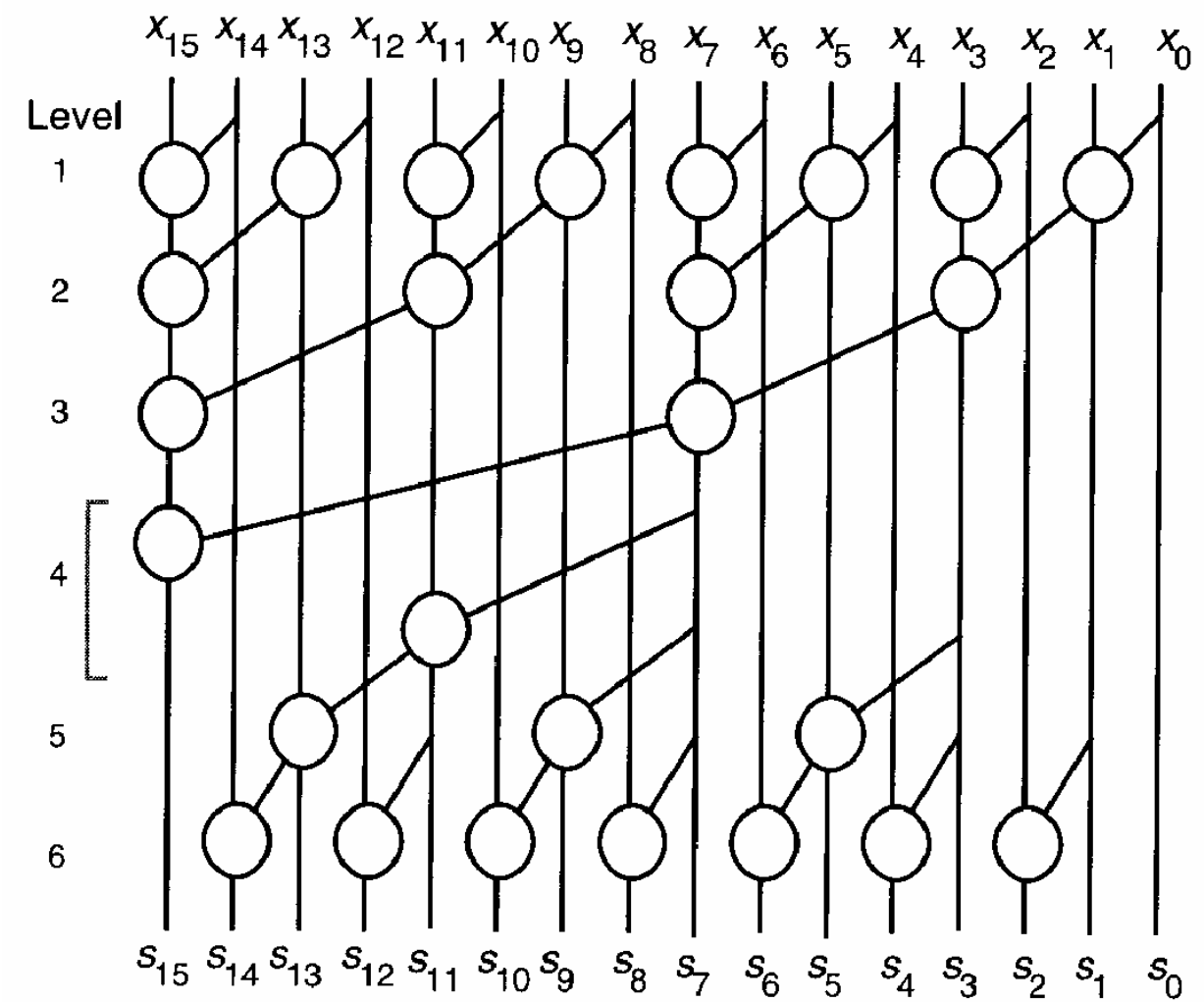
Delay recurrence: $D(k) = D(k/2) + 2 = 2 \log_2 k - 1$

Cost recurrence: $C(k) = C(k/2) + k - 1 = 2k - 2 - \log_2 k$

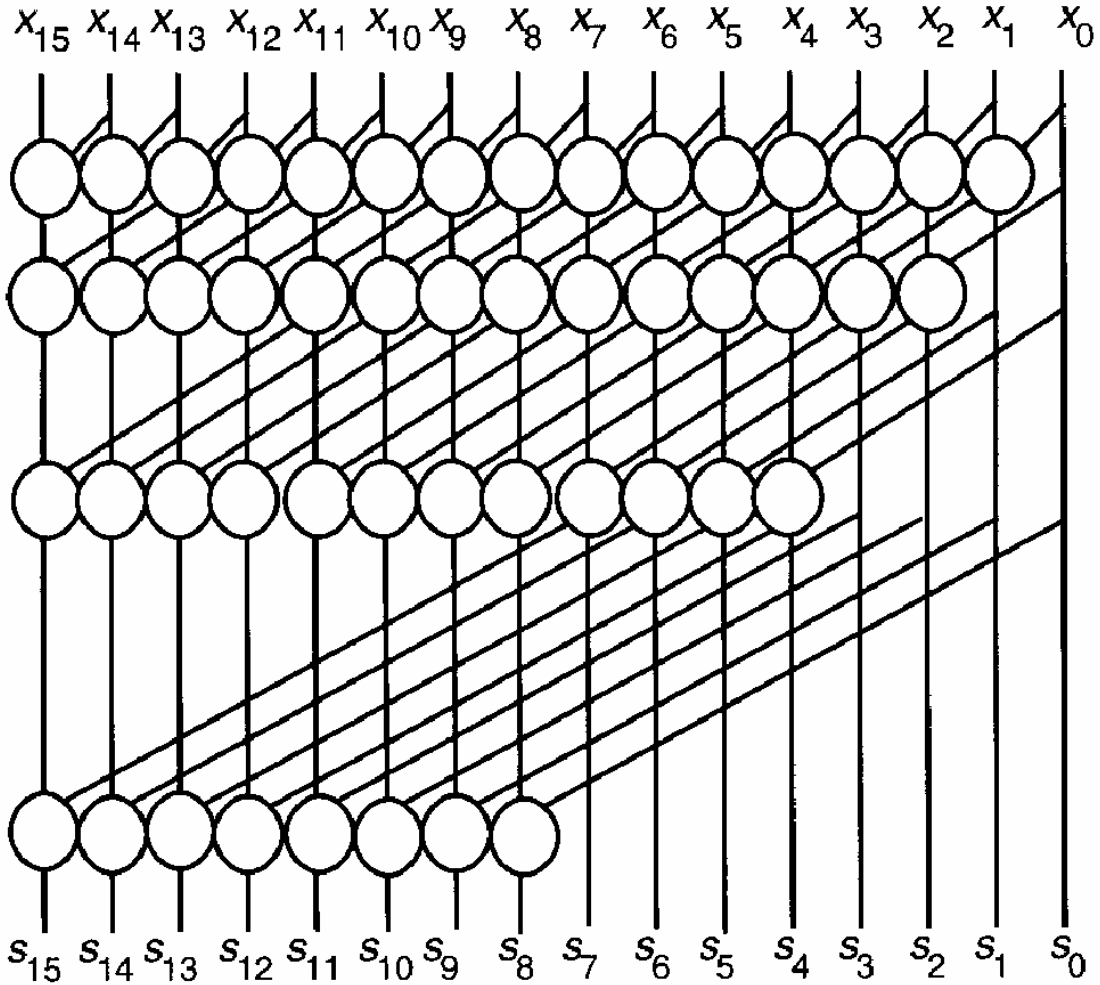
Divide and Conquer II
Brent-Kung Parallel Prefix



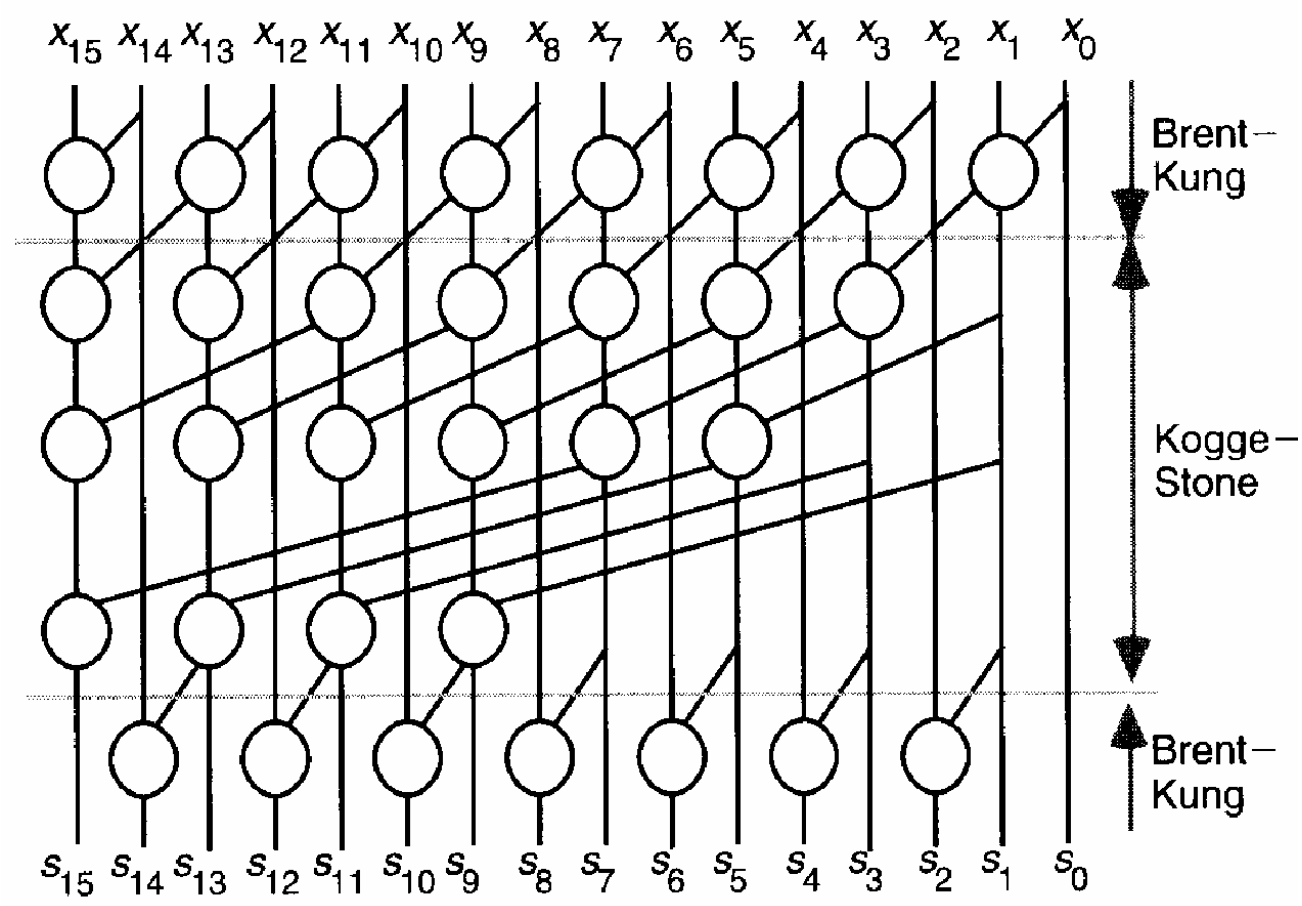
Brent-Kung Parallel Prefix Network



Kogge-Stone Parallel Prefix Network



Hybrid Brent-Kung / Kogge-Stone



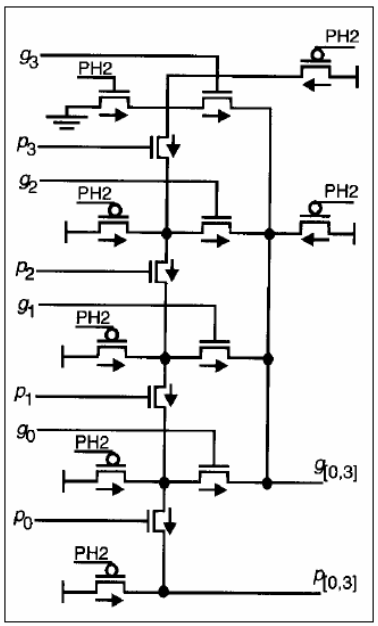
Network Comparisons

Network	Max Delay	Cost	Fan Out
Divide & Conquer I	$\text{Log}_2\, k$	$(k/2)\,\text{Log}_2\, k$	High
Brent - Kung	$2\,\text{Log}_2\, k - 1$	$2k - 2 - \text{Log}_2\, k$	Low
Kogge - Stone	$\text{Log}_2\, k$	$k\,\text{Log}_2\, k - k + 1$	Low
Hybrid B-K / K-S	$\text{Log}_2\, k + 1$	$(k/2)\,\text{Log}_2\, k$	Low

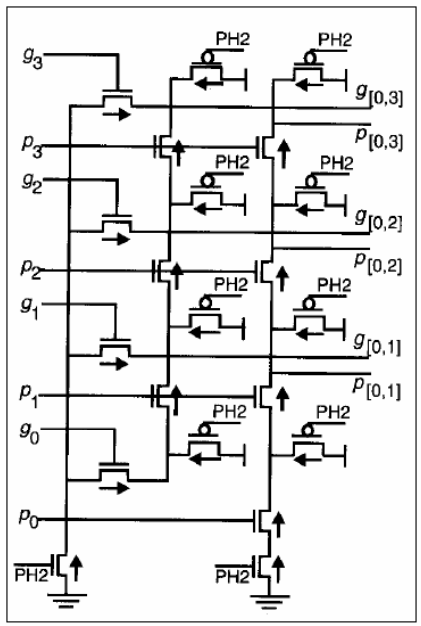
Cost is not a good estimate of Si area for these networks.
Regularity and interconnect are large factors.

MCC on Am29050

Lookahead for 64-bit, radix-256 addition



Level 1 MCC (not shown on block diagram)



Level 2,3 MCC