Parallel Prefix Adder Design with Matrix Representation

Youngmoon Choi
Sun Microsystems, Inc.
5300 Riata Park Court
Austin, TX 78727, USA
Youngmoon.Choi@sun.com

Earl E. Swartzlander. Jr.
Electrical & Computer Engineering
The University of Texas at Austin
Austin, TX 78712, USA

Abstract

This paper presents a one-shot batch process that generates a wide range of designs for a group of parallel prefix adders. The prefix adders are represented by two two-dimensional matrixes and two vectors. This matrix representation makes it possible to compose two functions for gate sizing which calculate the delay and the total transistor width of the carry propagation graph of adders. After gate sizing, the critical path net-lists of the carry propagation graph are generated from the matrix representation for spice delay calculation. The process is illustrated by generating sets of delay and total transistor width pairs for 32-bit and 64-bit cases.

1 Introduction

For parallel prefix graph addition, there are three basic schemes, Brent-Kung [2], Kogge-Stone [7] and Ladner-Fischer [8], and each has certain attractive characteristics. To achieve minimum depth, the Ladner-Fischer scheme allows infinite fan out while the Kogge-Stone scheme allows an infinite number of parallel connections. To alleviate the heavy capacitive load problem of these two schemes, the Brent-Kung scheme increases the number of levels which has the unfortunate effect of reducing the speed. Many hybrid schemes have been presented by using tradeoffs among these three schemes. Among them, Knowles [6] presented complete classes of regular fan-out prefix adders which are bounded at either end by the Ladner-Fischer and Kogge-Stone graphs. In the group, each adder has unique fan-out pattern.

In the Kogge-Stone approach, because the fan-out of the carry generation graph is uniform, the performance analysis of the adder is quite simple. However, with the Ladner-Fischer approach, the performance of an adder is strong function of the gate sizing due to its irregular fan-out. Therefore, for performance analysis of the Ladner-Fischer adders, a methodology to perform gate sizing is needed.

In this work, a one-shot process for analyzing the complete set of Knowles prefix adders is proposed. First, a matrix representation for gate level design of the Knowles adders is proposed which is composed of two two-dimensional matrixes and two vectors. In [15], one-dimensional vectors were used to represent adders in prefix graph level but in this work, two-dimensional matrixes are used to represent adders in gate level. Second, delay and transistor width calculation functions for gate sizing are constructed from the matrixes. Because the matrix representation contains enough information to construct its original adder, it is also possible to generate spice net-lists from the matrixes. These produce a one-shot batch process that generates the complete set of characteristic curves of the adders in a group. In the proposed process, prefix cell valency is fixed at 2 while the gate size domain is open. This is opposed to the previous work [1] in which gate sizes are fixed while prefix cell valency is open.

2 Background

2.1 Prefix Adder

Let $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$ be n-bit binary numbers with sum $S = s_{n-1}s_{n-2}\ldots s_0$. The Least Significant Bit (LSB) is bit 0 and the Most Significant Bit (MSB) is bit n-1. If c_0 is carry in signal of the summation, the following equations can be used to compute the sum:

$$g_i = a_i b_i \quad (1)$$

$$k_i = \overline{a_i + b_i} \quad (2)$$

$$p_i = a_i \oplus b_i \quad (3)$$

$$c_{i+1} = g_i + \bar{k_i} c_i \quad (4)$$

$$s_i = p_i \oplus c_i \ \ (i = 0,\ 1, \ldots, n-1) \quad (5)$$

where, g, k and p mean generate, kill and propagate, respectively.

The recurrence relation in Equation (4) can be applied repeatedly to obtain the following set of carry equations in terms of g_i, k_i and c_0.

$$c_{i+1} = g_i + (\sum_{j=0}^{i-1}(\prod_{k=j}^{i} \bar{k_k})g_j) + (\prod_{k=0}^{i} \bar{k_k})c_0, \quad where\ i = 0,\ 1, \ldots, n-1 \tag{6}$$

If binary operator $\circ$ is defined on ordered pair $(\bar{k}, g)$ by

$$(\bar{k_i}, g_i) \circ (\bar{k_j}, g_j) \stackrel{def}{=} (\bar{k_i}\bar{k_j},\ g_j + \bar{k_j}g_i) \tag{7}$$

then,

$$(1, c_0) \circ (\bar{k_0}, g_0) \circ \ldots \circ (\bar{k_i}, g_i) = (\bar{k_0}\bar{k_1} \ldots \bar{k_i}, c_{i+1}) \tag{8}$$

The most important properties of the $\circ$ operator are its associativity and idempotency. The associativity enables Equation (6) to be evaluated in parallel and the idempotency allows the sub-trees to overlap which provides some useful flexibility in the parallelization. All the prefix adders utilize this parallelism to calculate the result quickly.

Knowles [6] presented complete classes of regular fan-out prefix adders which are bounded at the extremes by the Kogge-Stone [7] and Ladner-Fischer [8] graphs. Knowles adders of 8-bit width are presented in Figure 1. In the figure, (1,1,1) is the Kogge-Stone adder and (1,2,4) is the Ladner-Fisher adder. The following rules are used for the construction of Knowles adders:

- Lateral wires at the j^{th} level span 2^j bits ($0 \leq level \leq \log_2 n - 1$).
- The lateral fan-out at the j^{th} level is a power of 2 between 1 and 2^j inclusive.
- The lateral fan-out at the j^{th} level cannot exceed that at the $(j+1)^{th}$ level.

Because of these construction rules, Knowles circuits are very regular and the numbers of circuits can be counted according to operand widths as shown in Table 1. The vector beneath each figure in Figure 1 represents the lateral fan-out of each level.

From now on, the vector is called the fan-out vector and expressed as follows:

$$fo = (fo(0), \ldots, fo(\log_2 n - 1)), \quad where\ n = operand\ width. \tag{9}$$

Each fan-out vector represents a circuit in the group as shown in Figure 1.

2.2 Gate Level Design of Knowles Adders

In this section, gate level designs of the Knowles adders are analyzed. For this purpose, gate level designs of the Kogge-Stone adder (Figure 2) and the Ladner-Fischer adder (Figure 3) are constructed first. In the figures, XNOR gates or equivalent logic gates before level 0 for generating $\bar{p}$ signals are omitted for simplicity. For the input gate (level 0), NAND and NOR gates are used and for the sum, XOR gates or equivalent logic gates are used. For the rest of the levels, there are four types of cells as shown in Figure 4. In the Ladner-Fischer adder, all the repeated inverters are removed from the carry trees to reduce the total transistor area except the output inverters of the multiple lateral fan-out gates which can reduce the total output capacitance of the gates.

The major difference between the two adders is their fan-out patterns. The Kogge-Stone adder has uniform fan-out throughout all the cells, but the Ladner-Fisher adder has a nonuniform fan-out and the fan-out increases as the level increases. There are two approaches to solve the large fan-out problem. One is buffer insertion ([1, 6]) and the other is increasing gradually the size of the gates along the fan-out path (instead of increasing the size of the gate, multiple gates can be placed in parallel [5]). The problem of buffer insertion is that it adds an additional logic level to the adder which increases the overall delay of the adder. Thus, in this work, the second method is used and a frame work is presented as a solution for this gate sizing problem.

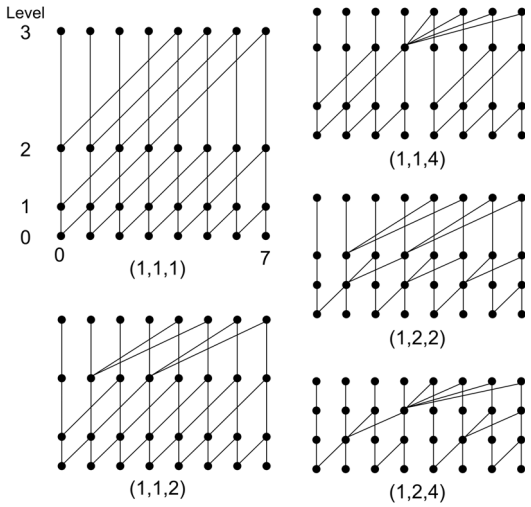

Figure 1. 8-bit Knowles adders.

Table 1. Number of Knowles adders.

Operand Width (bits)	Number of Knowles Adders
4	2
8	5
16	14
32	42
64	132
128	429
256	1430

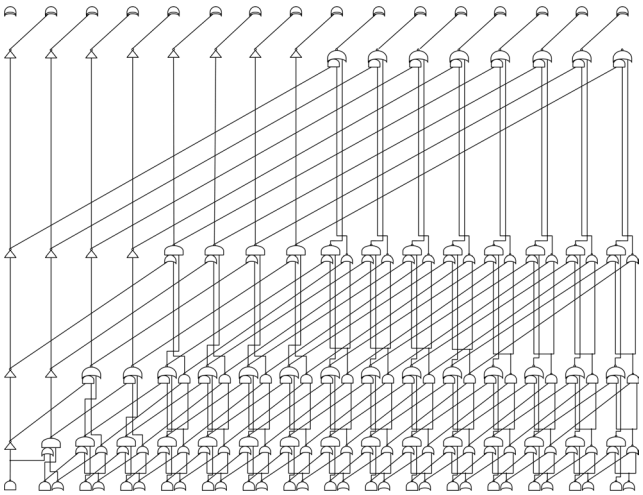

Figure 2. Gate level design of 16-bit Kogge-Stone adder.

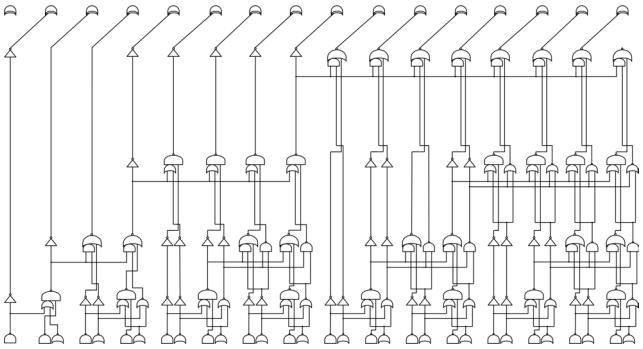

Figure 3. Gate level design of 16-bit Ladner-Fischer adder.

Through comparison of the two adders, a generalized gate level layout of the Knowles adders is presented in Figure 5.

In the generalized layout, wire connections are not drawn except for the wires for the last carry signal that is the same for all the adders in the group. In the figure, only the sub-sections marked by rectangles differ among adders in the group. Among the cells in the rectangular part, those which do not have any lateral input are pairs of inverters (type 2) and the others are type 4. This can be determined by the fan-out vector of the adder. If the fan-out of the i^{th} level is 2^i, it blocks the lateral connection of the cells in the lower level like the Ladner-Fischer adder. In the i^{th} level, first 2^{i-1} cells are type 1, cells from $2^{i-1}+1$ to 2^i are type 3, cells from $n-2^{i-1}+1$ to n are type 4 and the rest of the cells are of type 2 or type 4.

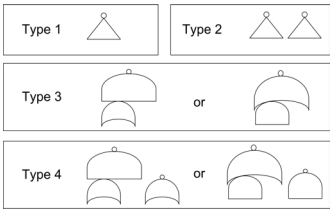

Figure 4. Types of cells for the carry propagation graph.

3 Matrix Representation of Knowles Adders

In this section, a matrix representation of the Knowles adders is proposed. The matrix representation is composed of two two-dimensional matrixes, the type matrix (T) and the size matrix (S), and two vectors, the fan-out vector (fo) and the size vector (s). The two dimensional representation is very natural for the prefix adders because the two dimensional index can identify uniquely a cell of the adders. The T matrix represents the types of the cells among four types shown in Figure 4. The S matrix marks cells which have variable size. Because level 0 is fixed for all the adders with the same operand width, the index of the matrixes starts from 1. To match the row index of the matrixes with the graphical representation of the adders in this work, the row index starts from bottom. These two dimensional matrixes are constructed from a given fan-out vector fo using the analysis results in Section 2.2.

The size vector s represents the multiplication factors for the variable size cells as follows:

$$s = (s(1), \ldots, s(\log_2 n - 1)), \quad where\ n = operand\ width. \tag{10}$$

All the gates in the last level are assumed to be of minimum size and therefore $s(\log_2 n)$ is excluded from the size vector. If they become non-minimum for any reason, the size of the other gates is simply increased at the same rate as the last level.

3.1 T matrix

The value of $T(i,j)$ represents the type of the cell in Figure 4 as follows:

$$\text{Type k} : T(i,j) = k,\ where\ k = 1,2,3,4. \tag{11}$$

Figure 6 is an algorithm for constructing the T matrix. As explained Section 2.2, in the i^{th} level, the first 2^{i-1} cells are set to type 1, cells from $2^{i-1}+1$ to 2^i are set to type 3, cells from $n-2^{i-1}+1$ to n are set to type 4 and the rest of the cells are checked by the innermost if-statement (line

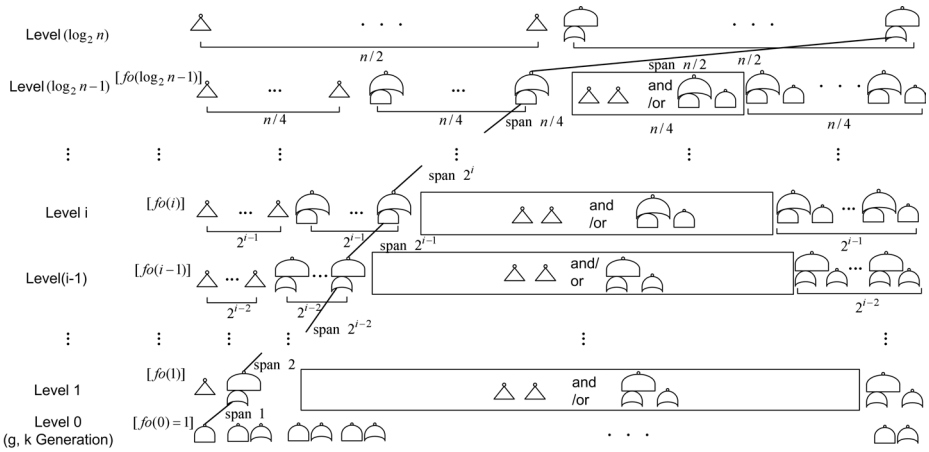

Figure 5. Generalized gate level layout of the carry graph of the Knowles adders (fo: fan-out vector).

Algorithm T_matrix
Input: $fo = (fo(0), \ldots, fo(\log_2 n - 1))$
Output: $T(i,j)$ matrix
begin
1. set $T(i,j) = 4$ for all $i = 1, 2, \ldots, \log_2 n$ and $j = 1, 2, \ldots, n$
2. **for** $i = 1 : \log_2 n$
3. set $T(i,j) = 1$ for $j = 1, 2, 3, \ldots, 2^{i-1}$
4. set $T(i,j) = 3$ for $j = 2^{i-1} + 1, \ldots, 2^i$
5. **for** $k = (2^i / fo(i-1)) + 1 : (n - 2^{i-1})/fo(i-1)$
6. **for** $m = i : \log_2 n - 1$
7. **if**$\{(fo(m) == 2^m) \wedge (\lceil k \cdot fo(i-1)/2^m \rceil \neq \lceil (k \cdot fo(i-1) - 2^{i-1})/2^m \rceil)\}$
set $T(i, fo(i-1) \cdot (k-1) + j) = 2$
for $j = 1, 2, \ldots, fo(i-1)$
end

Figure 6. Algorithm for generating the T matrix.

Algorithm S_matrix
Input: T, $fo = (fo(0), \ldots, fo(\log_2 n - 1))$
Output: $S(i,j)$ matrix
begin
1. set $S(i,j) = 0$ for all $i = 1, 2, \ldots, \log_2 n$ and $j = 1, 2, \ldots, n$
2. set $i = \log_2 n$
3. **for** $k = (2^{i-1}/fo(i-1)) + 1 : (n/fo(i-1))$
4. **if** $(fo(i-1) > 1)$
$S(i-1,\ fo(i-1) \cdot k - 2^{i-1}) = i - 1$
5. **for** $i = \log_2 n - 1 : 2 : -1$
6. **for** $k = (2^{i-1}/fo(i-1)) + 1 : (n/fo(i-1))$
7. **if** $\{fo(i-1) > 1 \wedge (T(i, fo(i-1) \cdot k) == 3\ or\ 4)\}$
$S(i-1,\ fo(i-1) \cdot k - 2^{i-1}) = i - 1$
8. **for** $j = 1 : n$
9. **if** $(S(i,j) == i)$
10. $S(i-1, j) = i - 1$
11. **if** $(j - 2^{i-1} > 0)$
$S(i-1, j - 2^{i-1}) = i - 1$
12. **Remove repeated inverters**
end

Figure 7. Algorithm for generating the S matrix.

7 in Figure 6) which sets a cell to type 2 if the cell doesn't have any lateral input.

3.2 S matrix

There is one assumption for the S matrix :
All the variable size cells in the same row have the same multiplication factor for their sizes.

The fan-out of the Kogge-Stone adder is already evenly distributed through out the carry graph. On the contrary, the fan-out of the Ladner-fischer adder is highly concentrated on the several high fan-out cells. To reduce the overall delay of the Ladner-Fisher adder, the high fan-out should be distributed into the overall carry graph. This fan-out spreading can be performed by increasing the size of the variable size cells of each level altogether. In other words, the critical path of the Ladner-Fischer adder is concentrated in just one path with all minimum-size gate implementation, but as the multiplication factors ($s_i \in s$) are increased, the critical path is moving and the other paths also become comparable to the critical path.

The convention for the values of $S(i,j)$ is:

$$\begin{aligned} \text{Empty cell} &: S(i,j) = -1 \\ \text{Minimum size cell} &: S(i,j) = 0 \\ \text{Variable size cell} &: S(i,j) = i \end{aligned}$$

Figure 7 is an algorithm for constructing the S matrix. The algorithm for the S matrix is composed of two steps. In the first step, it finds variable size cells. It searches the

rows of the T matrix from the top to the bottom. During the iteration, it first finds multiple lateral fan-out cells (lines 3-4 and 6-7 in Figure 7) and after that, it also finds the cells connected with variable size cells of upper level (lines 8-11 in Figure 7) because variable size cells may need large drivers.

In the second step, it removes unnecessary repeated inverters except for output inverters of lateral fan-out gates which can reduce the total output capacitance of the gates.

Figure 8 shows examples of T and S matrixes. In the figure, the index of bottom row is 1 to match the row index of the matrixes with the graphical representation of the adders.

In summary, the proposed matrix representation of the Knowles adders is a mathematical representation of a gate level design of the adders. The T matrix defines the types of the gates, the S matrix and the s vector define the size of the gates and the fo vector defines how to connect the gates.

4 Characterization of Knowles Adders

One of the advantages of the matrix representation of the Knowles adders is that it can be used to make functions for adder characterization. In the matrix representation, T, S and fo are constant but the size vector s is variable according to a design choice. Thus, s will be the input variable for the functions. In this work, two functions, one for delay and the other for total transistor width, will be constructed. The total transistor width is selected as a performance measure because both the dynamic power and the static power are directly dependant on it [11]. The functions are used for gate sizing through mathematical optimization.

Another advantage of the matrix representation is that spice net-lists can easily be generated from it because it is a mathematical equivalent of the gate level design of the adders. Thus, a batch process generating the complete set of characteristic curves of the adders is proposed using 6 characterization points on the delay and transistor width plane.

4.1 Functions for Gate Sizing

The delay model using logical and electrical efforts [9] is simple and many previous works [3, 4, 5, 14] have used it for the delay estimation or both gate sizing and delay estimation. In this work, it is only used for gate sizing because the adder delay will be more accurately calculated through spice simulation during the proposed batch characterization process.

The delay equation for the effort model is

$$d = \tau(gh + p). \tag{12}$$

(where τ is a constant for unit conversion, g is the logical effort, h is the electrical effort and p is the parasitic delay of the gate). In the ideal case, g and p are independent of the size of the gate, and the only factor that is affected by gate sizing is the electrical effort h which is the ratio of the capacitive load driven by the gate to the input capacitance at the corresponding input pin. However, simulation results show that g and p also vary with the input transition time [14] and the size of the gate and as a result of this, the delay estimation with the effort model is inaccurate. Nevertheless, the effort model can be used for gate size optimization because approximate optimal size vectors are enough for constructing characteristic curves.

In this work, three paths are selected for critical path delay evaluation.

Path-1: (0, 2) $\rightarrow$ (1, 2) $\rightarrow$ (2, 4) $\rightarrow$ (3, 8) $\rightarrow \cdots \rightarrow$ ($log_2(n-1)$,n/2) $\rightarrow$ ($log_2(n)$,n)

Path-2: (0, 4) $\rightarrow$ (1, 4) $\rightarrow$ (2, 4) $\rightarrow$ (3, 8) $\rightarrow \cdots \rightarrow$ ($log_2(n-1)$, n/2) $\rightarrow$ ($log_2(n)$, n)

Path-3: (0, n/2) $\rightarrow$ (1, n/2) $\rightarrow$ (2, n/2) $\rightarrow$ (3, n/2) $\rightarrow \cdots \rightarrow$ ($log_2(n-1)$, n/2) $\rightarrow$ ($log_2(n)$, n)

If the lateral fan-outs are big, the critical path is Path-1 which is mainly a chain of OAI and AOI gates. As the size of gates increases, the critical path moves to Path-2 and eventually the critical path becomes Path-3 which is mainly a chain of NAND and NOR gates in the center of the carry graph. If the critical path is located somewhere between Path-2 and Path-3, this means that the fan-out is already distributed in the carry graph in a certain degree and the delays of OAI-AOI and NAND-NOR gates become comparable. Thus, the actual critical path delay is not much bigger then that of Path-2 and Path-3.

Figure 9 shows a pseudo code for the delay calculation function of the carry propagation graph. The Path-1, 2 and 3 for each matrix representation are constructed using the effort model and the largest delay is the output of the function. The delay of the last level (level $\log_2 n$) is excluded because it is independent of the size vector. $size_{in}$ is the size multiplication factor of the input gates which is introduced to control the overall delay of the carry propagation graph.

Making a function for calculating the total transistor width of the carry propagation graph is quite straightforward compared to the delay function. w represents the total transistor width of the minimum size gate. Because NOR and NAND gates alternate in the graph, average value of their total transistor width used for simplicity. 'NDNR' denotes the average value.

Figure 10 shows a pseudo code for the total transistor width calculation function of the carry propagation graph. Unlike the delay function, this includes both size vector

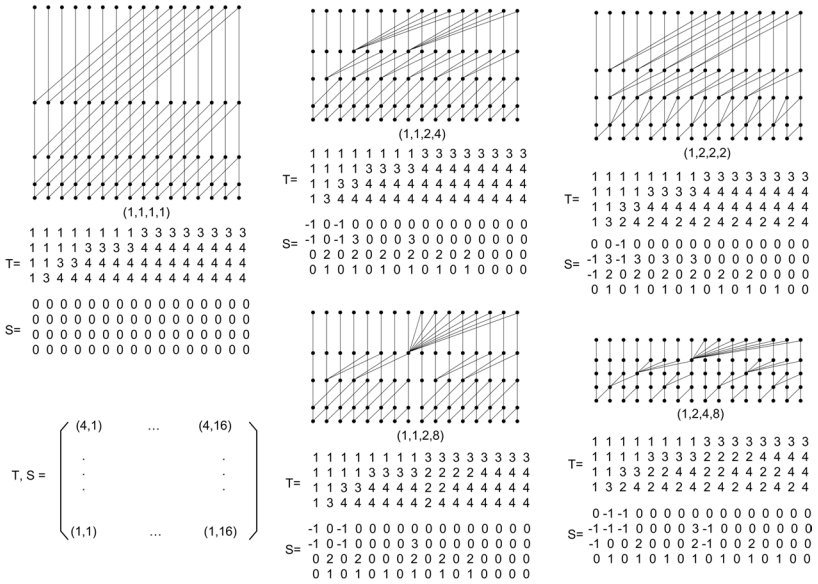

Figure 8. T and S matrix examples for the 16-bit case.

Function Delay
Input: $s = (s(1), \ldots, s(\log_2 n - 1))$
Output: D (delay from level 0 to level $(\log_2 n - 1)$)
Parameter: $fo,\ T,\ S,\ size_{in}$,
effort models of INV, NOR, NAND, AOI and OAI gates
begin
1. Construct Path-1
2. Construct Path-2
3. Construct Path-3
4. $D =$
$max\{Delay(Path-1), Delay(Path-2), Delay(Path-3)\}$
end

Figure 9. Function for delay calculation.

Function Width
Input: $s = (s(1), \ldots, s(\log_2 n - 1))$
Output: W (total transistor width of carry propagation tree)
Parameter: $fo,\ T,\ S,\ w_{inv},\ w_{aoai},\ w_{ndnr},\ size_{in}$
begin
1. $s = (s(1), \ldots, s(\log_2 n - 1), 1)$
2. $w = (w_{inv}, 2 \cdot w_{inv}, w_{aoai}, w_{aoai} + w_{ndnr})$
3. $W = w_{ndnr} \cdot (2n - 1) \cdot size_{in}$
4. **for** $i = 1 : \log_2 n$
5. **for** $j = 1 : n$
6. **switch** $S(i, j)$
case -1: %do nothing %
case 0: $W = W + w(T(i, j))$
otherwise : $W = W + s(i) \cdot w(T(i, j))$
end

Figure 10. Function for transistor width calculation.

dependant terms and size vector independent terms and represents the real total transistor width of the carry propagation graph.

4.2 Characteristic Curve

Using the proposed delay and transistor width functions, optimal gate sizing is performed with two costs, delay only and delay-width product. The gate sizing is performed by selecting the size vector which minimizes a given cost using any appropriate algorithm.

To refine the characteristic graph, three intermediate points are inserted, one between the minimum size point and the minimum delay-width product point and the other two between the minimum delay-width product point and the minimum delay point. Thus, six characteristic points are proposed as follows.

- Minimum size
- Intermediate point 1
- Minimum Delay-Width product
- Intermediate point 2
- Intermediate point 3
- Minimum delay

After gate sizing, spice simulation is performed to obtain a realistic delay for the carry propagation graph. Because the proposed matrix representation contains enough information for gate level implementation of the adders, Path1, 2 and 3 can easily be generated from it. The maximum delay among 3 paths is selected as the critical path delay of the prefix graph. Unlike the delay function, the delay of the last level (level $\log_2 n$) is included during the spice simulation to obtain the whole delay of the carry propagation graph. If

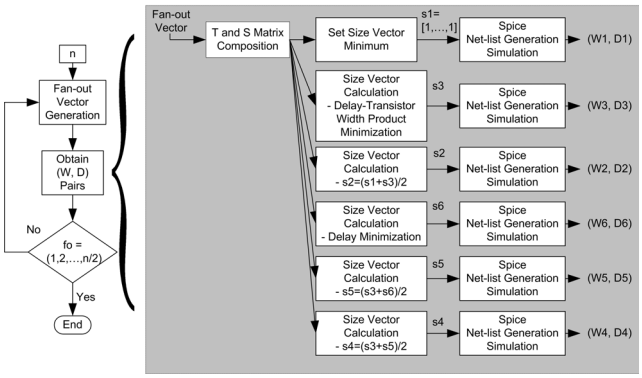

Figure 11. Block diagram for the proposed batch characterizing process.

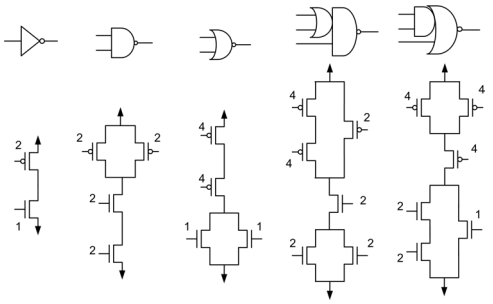

Figure 12. Transistor level design of library Cells $(L = 0.18\mu, W = 0.27\mu \times multiplication\ number\ of\ the\ transistor)$ **.**

Table 2. The Effort Models of the Library Cells.

	Inv	Nand	Nor	OAI($slow$)
$\tau g[ps]$	14.43	18.51	21.74	24.91
$\tau p[ps]$	29.97	39.71	55.15	59.80
	OAI($fast$)	AOI($slow$)	AOI($fast$)	
$\tau g[ps]$	19.01	24.66	22.62	
$\tau p[ps]$	39.76	60.10	49.52	

a static timing analyzer such as PathMill are available, the proposed 3-path spice simulation scheme can be replaced with the analyzer (in this case, a net-list of the whole carry graph will be used instead of the 3 paths). However, the proposed process uses HSpice because it's the more basic and popular circuit simulator and PathMill is used to cross-check the validity of the 3-path spice simulation scheme.

Figure 11 shows the proposed process. In the figure, Fan-out vectors are generated from $(1, 1, \ldots, 1)$ to $(1, 2, \ldots, n/2)$ and six pairs of (W, D) are generated at each iteration. W is calculated from the total transistor width generating function and D is calculated with spice simulation.

If adequate layout information is available, wire delay can be inserted into the delay function [13] and the spice net-lists to make the proposed process able to deal with the effects of wire delay. In this case, $S(i, j)$ of Kogge-Stone adder which has lateral fan-out will be also changed from 0 to i to mark it as a variable size cell to drive wire load.

5 Simulation

The proposed batch process is applied to the TSMC 0.18μ technology [12] whose minimum width (w_{min}) is 0.27μ. For simplicity, the sizes of all transistors are rounded to multiples of w_{min} and transistor width is expressed in number of w_{min}. All the spice simulations are performed at room temperature and a 1.8V power supply voltage.

For the first step of the proposed process, static library cells are composed base on P/N ratio 2. The results are shown in Figure 12. The effort model of each library cell is simulated under the conditions that the fan-out of the input signal is 3 (the input line is connected to two additional dummy gates together with the target gate) and the size of the gate is 3. For all the gates, the worst case input pins (the furthest from the output node) are selected for the simulation. The fast pins of the AOI and OAI gates are separately modeled and they denoted by AOI($fast$) and OAI($fast$), respectively. τg and τp are calculated using least square method from the data which is generated by three h's, 1, 3 and 6. The results are shown in Table 2. $size_{in}$ is selected as 2 in this simulation but it can be used as a tuning parameter which controls the ratio of the adder delay to the overall delay of the total critical path of a given design. Table 3 summarizes the parameters for the width function of Figure 10. The results of 32-bit case are shown in Table 4. The output capacitance of the gates of the last level (level $\log_2 n$) for spice simulation is assumed to be of $1.5 \times Cin_{aoai}$. In the table, the approximate delay model provides an upper bound for the size vector and the actual minimum delay size vector differ from that in some of the cases. For those cases, the minimum delay size vector can be located using the intermediate points of the process.

The 64-bit case has also been simulated and Figures 13 and 14 show the W-D curves of some Knowles carry

Table 3. Parameters for the Width Function ($[number\ of\ w_{min}]$).

w_{inv}	w_{ndnr}	w_{aoai}	$size_{in}$
3	9	16.5	2

Table 4. 32-bit case simulation result ($W[number\ of\ w_{min}], D[ps]$).

Fan-out	Min. Size (W1,D1)	Inter. 1 s2 (W2,D2)	Opt. Pro. s3 (W3,D3)	Inter. 2 s4 (W4,D4)	Inter. 3 s5 (W5,D5)	Min. Delay s6 (W6,D6)
(1,1,1,1,1)	(4237,495)	(1, 1, 1, 1) (4237,495)	(1, 1, 1, 1) (4237,495)	(1, 1, 1, 1) (4237, 495)	(1, 1, 1, 1) (4237, 495)	(1, 1, 1, 1) (4237, 495)
(1,1,1,1,2)	(4237,526)	(1, 1, 1, 1) (4237,526)	(1, 1, 1, 1) (4237,526)	(1, 1, 1, 1) (4237, 526)	(2, 2, 2, 2) (4824, 501)	(2, 2, 2, 2) (4824, 501)
(1,1,1,1,4)	(4237,584)	(1, 1, 1, 2) (4276,550)	(1, 1, 1, 2) (4276,550)	(1, 1, 1, 2) (4276, 550)	(2, 2, 2, 2) (4635, 534)	(2, 3, 3, 3) (4837, 536)
(1,1,1,1,8)	(4237,695)	(1, 2, 2, 3) (4411,597)	(1, 2, 2, 4) (4431,588)	(1, 2, 2, 4) (4431, 588)	(2, 3, 3, 5) (4780, 563)	(2, 4, 4, 6) (4935, 576)
(1,1,1,1,16)	(3945,852)	(1, 2, 2, 4) (4129,595)	(1, 2, 3, 7) (4221,568)	(1, 2, 3, 7) (4221, 568)	(2, 3, 4, 8) (4567, 557)	(2, 4, 4, 9) (4677, 567)
(1,1,1,2,2)	(4213,561)	(1, 1, 1, 1) (4213,561)	(1, 1, 1, 1) (4213,561)	(1, 1, 1, 1) (4213, 561)	(2, 2, 2, 2) (5106, 546)	(2, 2, 2, 2) (5106, 546)
(1,1,1,2,4)	(4213,620)	(1, 1, 1, 2) (4252,584)	(1, 1, 1, 2) (4252,584)	(1, 1, 1, 2) (4252, 584)	(2, 2, 2, 3) (5106, 570)	(2, 2, 2, 3) (5106, 570)
(1,1,1,2,8)	(4213,733)	(1, 1, 1, 2) (4233,649)	(1, 1, 1, 2) (4233,649)	(1, 1, 1, 2) (4233, 649)	(2, 2, 2, 3) (5067, 616)	(2, 3, 3, 4) (5604, 627)
(1,1,1,2,16)	(3921,885)	(1, 1, 2, 3) (4081,653)	(1, 1, 2, 5) (4114,622)	(1, 1, 2, 6) (4131, 620)	(2, 2, 3, 7) (4801, 604)	(2, 3, 4, 8) (5175, 606)
(1,1,1,4,4)	(4201,691)	(1, 2, 2, 2) (4506,630)	(1, 2, 2, 3) (4545,618)	(1, 2, 2, 3) (4545, 618)	(2, 3, 3, 3) (5107, 591)	(2, 4, 3, 3) (5251, 599)
(1,1,1,4,8)	(4201,806)	(1, 2, 2, 2) (4486,695)	(1, 2, 2, 3) (4506,666)	(1, 3, 2, 3) (4650, 690)	(2, 4, 3, 4) (5232, 632)	(3, 5, 4, 5) (5814, 623)
(1,1,1,4,16)	(3909,952)	(1, 2, 2, 4) (4179,654)	(1, 2, 3, 6) (4288,620)	(1, 3, 4, 7) (4525, 616)	(2, 4, 5, 9) (5076, 591)	(2, 5, 7, 11) (5406, 594)
(1,1,1,8,8)	(3786,884)	(1, 2, 3, 3) (4104,643)	(1, 2, 4, 4) (4191,627)	(1, 3, 4, 4) (4335, 643)	(2, 4, 5, 5) (4863, 602)	(2, 5, 6, 6) (5094, 609)
(1,1,1,8,16)	(3630,1089)	(1, 2, 3, 4) (3958,668)	(1, 2, 4, 7) (4075,622)	(1, 3, 5, 8) (4303, 615)	(2, 4, 6, 9) (4828, 590)	(2, 5, 7, 11) (5073, 594)
(1,1,2,2,2)	(4213,586)	(1, 1, 1, 1) (4213,586)	(1, 1, 1, 1) (4213,586)	(1, 1, 1, 1) (4213, 586)	(2, 2, 2, 2) (5208, 560)	(2, 2, 2, 2) (5208, 560)
(1,1,2,2,4)	(4213,645)	(1, 1, 1, 2) (4252,609)	(1, 1, 1, 2) (4252,609)	(1, 1, 1, 2) (4252, 609)	(2, 2, 2, 3) (5208, 584)	(2, 3, 3, 3) (5776, 604)
(1,1,2,2,8)	(4213,758)	(1, 1, 1, 2) (4233,675)	(1, 1, 1, 2) (4233,675)	(1, 2, 2, 3) (4821, 669)	(2, 3, 3, 4) (5757, 636)	(3, 4, 4, 5) (6693, 638)
(1,1,2,2,16)	(3897,914)	(1, 1, 2, 3) (4057,681)	(1, 1, 2, 5) (4090,650)	(1, 2, 3, 6) (4540, 637)	(2, 3, 4, 7) (5338, 617)	(3, 4, 5, 9) (6153, 621)
(1,1,2,4,4)	(4201,717)	(1, 1, 1, 2) (4240,675)	(1, 1, 1, 2) (4240,675)	(1, 1, 1, 2) (4240, 675)	(2, 2, 2, 2) (5035, 630)	(2, 3, 3, 3) (5521, 629)
(1,1,2,4,8)	(4201,832)	(1, 1, 2, 3) (4362,708)	(1, 1, 2, 4) (4381,698)	(1, 2, 2, 4) (4707, 696)	(2, 3, 3, 4) (5502, 661)	(3, 4, 4, 5) (6316, 657)
(1,1,2,4,16)	(3885,981)	(1, 1, 2, 3) (3994,716)	(1, 1, 2, 5) (4027,684)	(1, 2, 3, 6) (4401, 659)	(2, 3, 4, 8) (5139, 633)	(3, 4, 6, 10) (5953, 633)
(1,1,2,8,8)	(3786,907)	(1, 2, 3, 3) (4150,644)	(1, 2, 4, 4) (4237,627)	(2, 3, 5, 5) (4863, 596)	(2, 4, 6, 6) (5140, 598)	(3, 6, 8, 7) (6024, 605)
(1,1,2,8,16)	(3606,1118)	(1, 2, 3, 4) (3981,683)	(1, 2, 4, 7) (4098,638)	(2, 3, 5, 8) (4720, 607)	(2, 4, 7, 10) (5079, 592)	(3, 6, 9, 13) (5992, 598)
(1,1,4,4,4)	(3792,727)	(1, 2, 2, 2) (4122,631)	(1, 2, 2, 2) (4122,631)	(1, 3, 2, 2) (4291, 645)	(2, 4, 3, 3) (4969, 592)	(3, 6, 4, 4) (5817, 593)
(1,1,4,4,8)	(3792,842)	(1, 2, 2, 2) (4102,696)	(1, 2, 2, 3) (4122,667)	(1, 3, 2, 3) (4291, 681)	(2, 4, 3, 4) (4950, 625)	(3, 6, 4, 5) (5778, 622)
(1,1,4,4,16)	(3522,1004)	(1, 2, 2, 4) (3817,662)	(1, 2, 3, 6) (3927,628)	(1, 3, 4, 7) (4189, 609)	(2, 4, 5, 9) (4816, 587)	(2, 6, 7, 11) (5341, 608)
(1,1,4,8,8)	(3552,931)	(1, 2, 3, 3) (3895,643)	(1, 2, 4, 4) (3982,627)	(1, 3, 5, 5) (4239, 636)	(2, 4, 6, 6) (4843, 598)	(2, 5, 7, 8) (5119, 607)
(1,1,4,8,16)	(3360,1142)	(1, 2, 3, 4) (3714,676)	(1, 2, 4, 7) (3831,631)	(1, 3, 5, 8) (4084, 609)	(2, 4, 7, 10) (4770, 585)	(3, 6, 9, 12) (5625, 598)
(1,2,2,2,2)	(3915,597)	(1, 1, 1, 1) (3915,597)	(1, 1, 1, 1) (3915,597)	(1, 1, 1, 1) (3915, 597)	(2, 2, 2, 2) (4935, 559)	(2, 2, 2, 2) (4935, 559)
(1,2,2,2,4)	(3915,656)	(1, 1, 1, 2) (3954,620)	(1, 1, 1, 2) (3954,620)	(1, 1, 1, 2) (3954, 620)	(2, 2, 2, 2) (4896, 593)	(3, 3, 3, 3) (5877, 591)
(1,2,2,2,8)	(3915,769)	(1, 1, 1, 2) (3934,686)	(1, 1, 1, 2) (3934,686)	(1, 2, 2, 3) (4522, 679)	(2, 3, 3, 4) (5484, 635)	(3, 4, 4, 5) (6445, 632)
(1,2,2,2,16)	(3618,923)	(1, 1, 2, 3) (3778,691)	(1, 1, 2, 5) (3811,660)	(1, 2, 3, 6) (4261, 645)	(2, 3, 4, 7) (5085, 615)	(3, 4, 5, 9) (5925, 615)
(1,2,2,4,4)	(3903,728)	(1, 1, 1, 2) (3942,686)	(1, 1, 1, 2) (3942,686)	(1, 2, 2, 2) (4389, 680)	(2, 3, 3, 3) (5248, 627)	(3, 4, 4, 3) (6069, 628)
(1,2,2,4,8)	(3903,843)	(1, 1, 2, 3) (4063,719)	(1, 1, 2, 4) (4083,709)	(1, 2, 3, 5) (4549, 706)	(2, 3, 4, 5) (5370, 662)	(3, 4, 5, 6) (6210, 655)
(1,2,2,4,16)	(3606,991)	(1, 1, 2, 3) (3715,726)	(1, 1, 2, 5) (3748,694)	(1, 2, 3, 6) (4122, 668)	(2, 3, 4, 8) (4885, 631)	(3, 4, 6, 10) (5725, 628)
(1,2,2,8,8)	(3510,916)	(1, 2, 3, 3) (3874,653)	(1, 2, 4, 4) (3961,628)	(2, 3, 5, 5) (4612, 596)	(2, 4, 7, 6) (4957, 603)	(3, 6, 9, 7) (5866, 609)
(1,2,2,8,16)	(3342,1128)	(1, 2, 3, 4) (3717,692)	(1, 2, 4, 7) (3834,646)	(2, 3, 5, 8) (4482, 606)	(2, 4, 7, 10) (4840, 590)	(3, 6, 10,13) (5847, 604)
(1,2,4,4,4)	(3630,727)	(1, 2, 2, 2) (3960,631)	(1, 2, 2, 2) (3960,631)	(2, 3, 2, 2) (4503, 605)	(2, 4, 3, 3) (4833, 592)	(3, 6, 4, 4) (5706, 593)
(1,2,4,4,8)	(3630,842)	(1, 2, 2, 2) (3940,696)	(1, 2, 2, 3) (3960,667)	(2, 3, 3, 4) (4644, 622)	(2, 4, 3, 4) (4813, 625)	(3, 6, 4, 5) (5667, 622)
(1,2,4,4,16)	(3360,1019)	(1, 2, 2, 4) (3655,675)	(1, 2, 3, 6) (3765,641)	(2, 3, 4, 7) (4401, 600)	(2, 5, 6, 9) (4926, 590)	(3, 7, 8, 11) (5824, 602)
(1,2,4,8,8)	(3354,945)	(1, 2, 3, 3) (3697,650)	(1, 2, 4, 4) (3784,627)	(2, 3, 5, 5) (4414, 596)	(2, 4, 6, 6) (4671, 598)	(3, 6, 8, 7) (5538, 605)
(1,2,4,8,16)	(3174,1157)	(1, 2, 3, 4) (3528,727)	(1, 2, 4, 7) (3645,644)	(1, 3, 5, 8) (3898, 619)	(2, 4, 7, 10) (4609, 585)	(3, 6, 9, 12) (5490, 598)

propagation graphs from 32-bit case and 64-bit case, respectively. Both graphs show that the proposed gate sizing scheme well decides the size vectors. Some of the intermediate points show bumps because they are selected by arithmetic averaging. As expected, Kogge-Stone $(1, 1, \ldots, 1)$ has the best delay, but the delay advantage will be decreased if wire delay is included. If both fast delay and moderate wire load are needed, $(1, \ldots, 1, n/2)$ is a suitable choice because they are divided into two n/2 graphs. The figures also show clearly which adder is the best for a given design criteria when only minimum size gates are available.

To crosscheck the validity of the proposed critical path search scheme, the delays of the proposed scheme are compared with the critical path reports of the PathMill static timing analyzer whose inputs are spice net-lists of complete carry graphs. The delay is the average of rising and falling transition times. Table 5 shows the comparison results. They are well matched within 4% error in average except for large fan-out with minimum size cases (case 2, 13 and 18) which are practically uninteresting. Moreover, the trends of two simulations are well matched.

6 Conclusions

A matrix representation for the gate level design of Knowles adders is proposed which is composed of two two-dimensional matrixes and two vectors. The matrix representation is successfully applied to gate sizing of the adders.

A fast characterization process for Knowles adders is proposed which can be used for selecting an adder during the early design phase. It is easily programmable through CAD tools such as MATLAB. It is a one-shot process which generates all characteristic curves of a group of adders automatically. For accurate delay estimation, the process

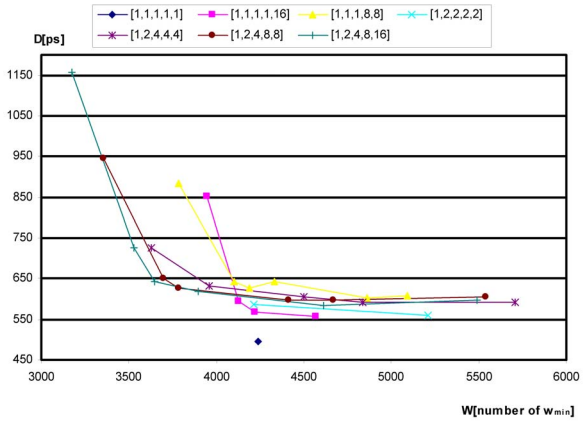

Figure 13. W-D Graphs for the selected elements of Table 4 (32-bit).

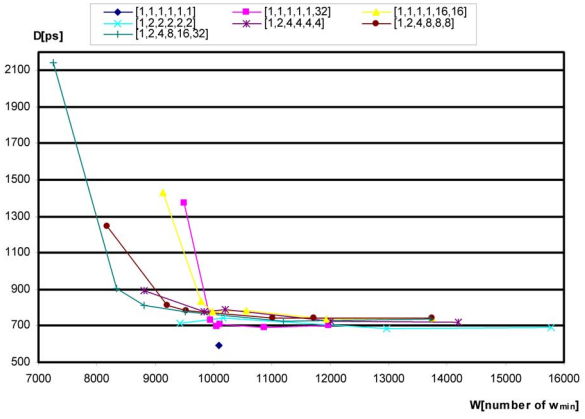

Figure 14. W-D Graphs for the selected elements of 64-bit case.

uses spice simulation. The set of characteristic curves provides a designer with useful information such as lower bounds for the delay, the total transistor width and the W-D slope.

References

[1] A. Beaumont-Smith and C. Lim. Parallel prefix adder design. *Proc. 15th IEEE Symposium on Computer Arithmetic*, pages 218–225, 2001.

[2] R. P. Brent and H. T. Kung. A regular layout for parallel adders. *IEEE Trans. Computers*, C-31:260–264, 1982.

[3] H. Dao and V. G. Oklobdzija. Application of logical effort on delay analysis of 64-bit static carry-lookahead adder. *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, pages 1322–1324, 2001.

Table 5. Critical path delay comparison between PathMill and proposed scheme ($[ps]$).

	Fan-out	Size	Proposed	PathMill	Error
1	(1,1,1,1,1)	(1,1,1,1)	495	497	-0.49%
2	(1,2,4,8,16)	(1,1,1,1)	1157	1039	10.18%
3	(1,2,4,8,16)	(1,3,5,8)	619	626	-1.05%
4	(1,2,4,8,16)	(3,6,9,12)	598	593	0.82%
5	(1,2,2,2,2)	(1,1,1,1)	597	564	5.46%
6	(1,2,2,2,2)	(2,2,2,2)	559	546	2.39%
7	(1,2,4,4,4)	(1,1,1,1)	727	691	4.93%
8	(1,1,1,4,8)	(1,3,2,3)	690	650	5.82%
9	(1,1,1,4,8)	(3,5,4,5)	623	607	2.54%
10	(1,1,1,8,8)	(1,1,1,1)	884	822	6.97%
11	(1,1,1,8,8)	(2,5,6,6)	609	590	3.20%
12	(1,1,1,1,1,1)	(1,1,1,1,1)	590	594	-0.79%
13	(1,2,4,8,16,32)	(1,1,1,1,1)	2143	1827	14.7%
14	(1,2,4,8,16,32)	(1,3,5,10,15)	774	771	0.43%
15	(1,2,4,8,16,32)	(2,4,7,12,18)	725	725	-0.06%
16	(1,2,4,8,16,32)	(3,7,9,15,22)	737	743	-0.81%
17	(1,2,2,2,2,2)	(1,1,1,1,1)	698	683	2.18%
18	(1,1,1,1,1,32)	(1,1,1,1,1)	1373	1233	10.17%
19	(1,1,1,1,1,32)	(1,2,2,4,10)	698	717	-2.74%
20	(1,1,1,1,1,32)	(2,3,3,6,13)	688	693	-0.65%

[4] H. Dao and V. G. Oklobdzija. Application of logical effort techniques for speed optimization and analysis of representative adders. *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, pages 1666–1669, 2001.

[5] D. Harris and I. Sutherland. Logical effort of carry propagate adders. *Proc. 37th Asilomar Conf. Signals, Systems, and Computers*, pages 873–878, 2003.

[6] S. Knowles. A family of adders. *Proc. 15th IEEE Symposium on Computer Arithmetic*, pages 277–281, 2001.

[7] P. M. Kogge and H. S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Computers*, C-22:786–793, 1973.

[8] R. Ladner and M. Fischer. Parallel prefix computation. *JACM*, 27(4):831–838, 1980.

[9] I. E. Sutherland and R. F. Sproull. *Designing for Speed on the Back of an Envelope in Advanced Research in VLSI*. Univ. of Calif., CA, 1991.

[10] T. Han and D. A. Carlson. Fast area-efficient vlsi adders. *Proc. 8th IEEE Symposium on Computer Arithmetic*, pages 49–56, 1987.

[11] Y. Taur and T. H. Ning. *Fundamentals of Modern VLSI Devices*. Cambridge University Press, New York, 1988.

[12] TSMC. *Spice Model of 0.18u LO EPI Process*. 2004.

[13] K. Venkat. Generalized delay optimization of resistive interconnections through an extension of logical effort. *Proc. IEEE International Symposium on Circuit and Systems*, pages 2106–2109, 1993.

[14] X. Yu and V. G. Oklobdzija. Application of logical effort on design of arithmetic blocks. *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, pages 872–874, 2001.

[15] M. M. Ziegler and M. R. Stan. A unified design space for regular parallel prefix adders. *Proc. Design, Automation and Test in Europe Conference and Exhibition*, pages 1386–1387, 2004.