Journal of VLSI Signal Processing, 3, 265–274 (1991)

Improved CLA Scheme with Optimized Delay

BRIAN D. LEE AND VOJIN G. OKLOBDZIJA*

Department of Electrical Engineering and Computer Science, University of California, Berkeley

Received January 31, 1991; Revised April 25, 1991.

Abstract. The delay characteristics of carry-lookahead (CLA) adders are examined with respect to a delay model that accounts for fan-in and fan-out dependencies. Though CLA structures are considered among the fastest topologies for performing addition, they have also been characterized as providing marginal speed improvement for the amount of hardware invested. This analysis shows that this inefficiency can be explained by the suboptimal nature of common CLA implementations. Simulation results show that the CLA structures in wide use can be improved by varying the block sizes and the number of levels within each adder. Examples of optimal CLA structures are given and heuristic methods for finding these structures are presented.

1. Introduction

Analysis of carry-lookahead adders is important in the design of high performance machines. In these designs, processor speed is a primary concern and carry-lookahead structures are often used because their delay times exhibit log dependence on the size of the adder and they are considered among the fastest circuit topologies for performing addition. However, adder comparisons [1], [2] have ranked CLA structures low on effective hardware utilization and this apparent inefficiency raises concerns over the optimality of current CLA implementations. Simulation results from this research show that the commonly used CLA structures can be improved by varying block sizes and levels within the adder.

Typical CLA implementations are made of lookahead units of relatively fixed sizes. This artificial constraint produces slack in the circuit and results in poor hardware utilization. The strategy of varying group sizes to reduce slack and improve performance is a promising idea and has been used successfully on carry-skip adders [3], [4]. A natural extension of this method is to also vary the number of lookahead levels [5], [6]. The example structures in figure 1 illustrate what it means to vary group sizes and lookahead levels. Each box corresponds to a BCLA/CLA unit and the enclosed number gives the unit size in bits. The dotted lines in figure 1(d) represent a connection to a primary input of the CLA network. All other connections to primary inputs are implicit and not shown. Since an accurate measure of the available slack is required to effectively implement these strategies, this work uses a delay model that accounts for fan-in and fan-out dependencies. The parameter values used in the model are based on industrial data.

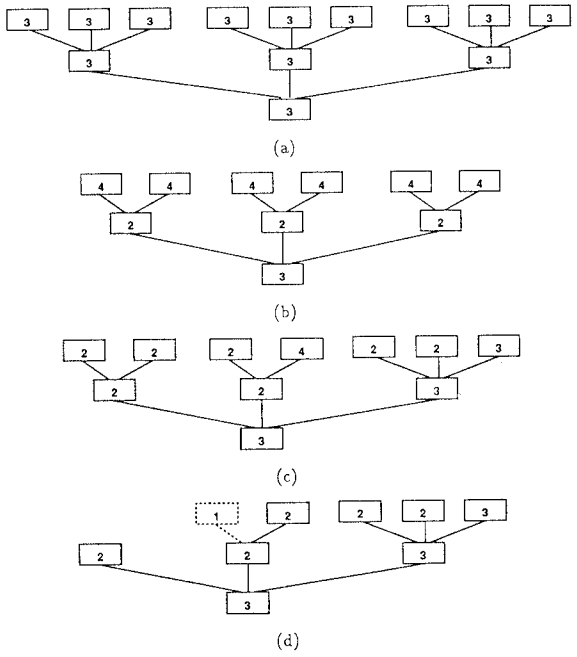

Fig. 1. Degrees of freedom in group sizes and lookahead levels: (a) fixed groups and fixed levels, (b) variable-sized groups between levels and fixed levels, (c) variable-sized groups anywhere (between levels and within levels) and fixed levels, and (d) variable-sized groups anywhere and variable levels.

*Currently with IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

A simulation program has been written to compare different CLA structures. Preliminary data on varying block sizes was obtained through exhaustive search. Based on this data and an analysis of delay and slack in the CLA scheme, heuristics were chosen to find structures with completely variable block sizes and levels. The structures found by these heuristics are faster than more constrained topologies.

2. Carry Lookahead Structure

The simulation results in this paper are based on carry-lookahead adders that implement full lookahead. A description of the basic organization of carry-lookahead adders can be found in references such as [7]. Each adder consists of three main components—the propagate and generate generation circuitry, the carry-lookahead network, and the sum generation circuitry. This work concerns varying the sizes of circuit blocks and the number of levels in the carry-lookahead network to optimize adder delay.

Given an n-bit adder with inputs, A and B, the logic equations for producing the initial propagate and generate signals and the final sum signals are

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

and

$$S_i = P_i \oplus C_{i-1}$$

for $0 \le i \le n - 1$. The simulations assume that P_i and S_i are produced by monolithic XOR gates instead of two levels of NAND gates and the sum XOR gate is assumed to have a fan-out of one.

The carry-lookahead network in a full carry-lookahead adder consists of a tree of block carry-lookahead (BCLA) units rooted at a single carry-lookahead (CLA) unit. Two different implementations of BCLA/CLA units are analyzed. Their performance differences are discussed later.

The first implementation generates carry signals in two levels of logic. Four-bit versions of these circuits are shown in figures 2 and 3. A k-bit carry-lookahead unit of this type generates

$$C_j = G_j + G_{j-1}P_j + G_{j-2}P_{j-1}P_j + \ldots + G_0P_1 \ldots P_j + C_{-1}P_0 \ldots P_j$$

where $0 \le j \le k - 1$ and a k-bit block carry-lookahead unit of this type generates

$$P^* = P_0P_1 \ldots P_{k-1}$$

$$G^* = G_{k-1} + G_{k-2}P_{k-1} + \ldots + G_0P_1 \ldots P_{k-1}$$

$$C_j = G_j + G_{j-1}P_j + G_{j-2}P_{j-1}P_j + \ldots + G_0P_1 \ldots P_j + C_{-1}P_0 \ldots P_j$$

where $0 \le j \le k - 2$.

The second implementation generates carry signals in three levels of logic. Four-bit versions of these circuits are shown in figures 4 and 5. A k-bit carry-lookahead unit of this type generates

$$C_j = G_j^* + C_{-1}P_j^*$$

where $0 \le j \le k - 1$ and a k-bit block carry-lookahead unit of this type generates

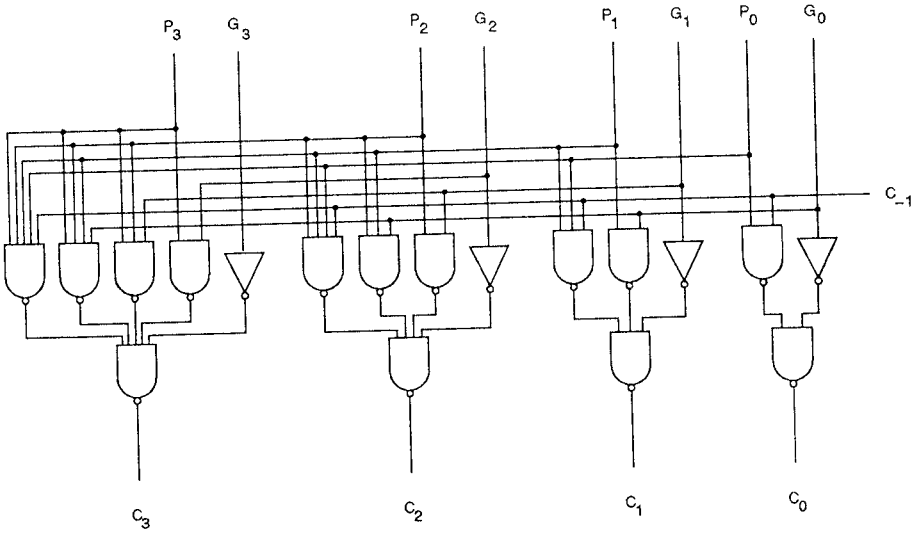

Fig. 2. A 4-bit carry-lookahead unit with 2-level C_i logic.

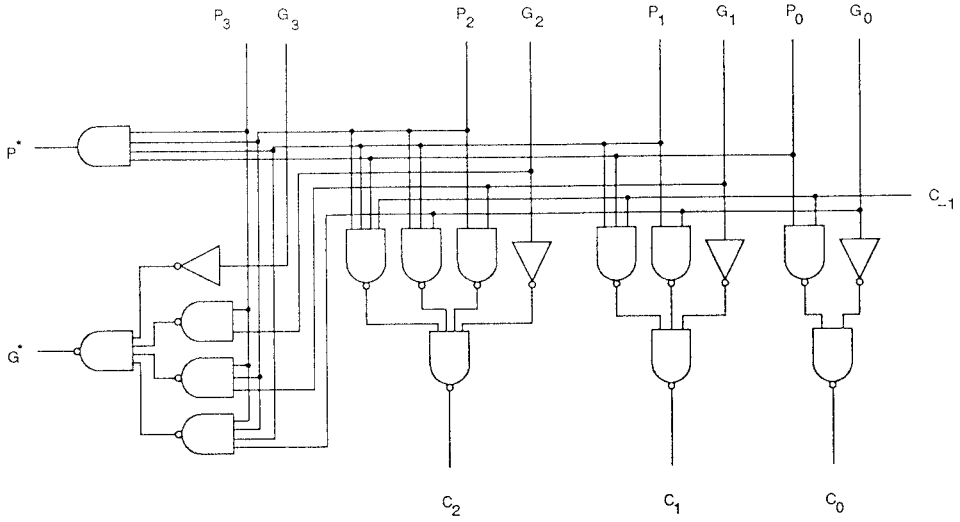

Fig. 3. A 4-bit block carry-lookahead with 2-level C_i logic.

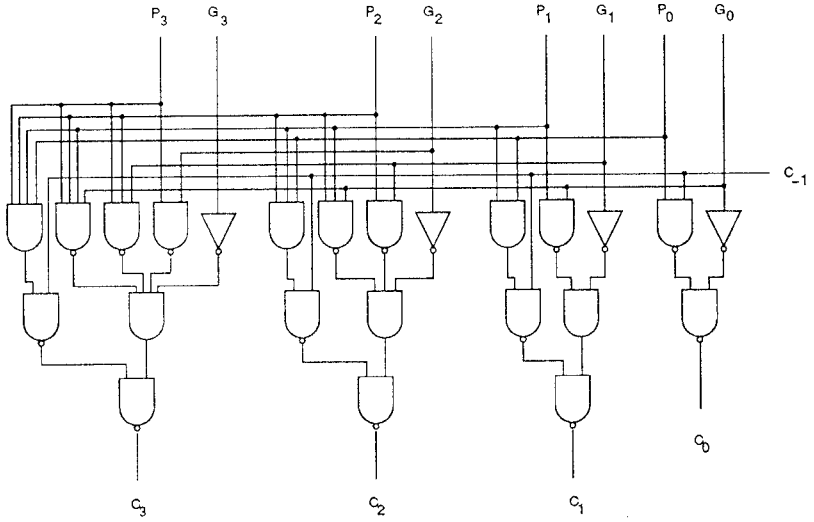

Fig. 4. A 4-bit carry-lookahead unit with 3-level C_i logic.

$$P^* = P^*_{k-1}$$

$$G^* = G^*_{k-1}$$

$$C_j = G^*_j + C_{-1}P^*_j$$

where $0 \leq j \leq k - 2$. For both circuit blocks,

$$G^*_j = G_j + G_{j-1}P_j + G_{j-2}P_{j-1}P_j + \ldots + G_0P_1 \ldots P_j$$

and

$$P^*_j = P_0P_1 \ldots P_j.$$

The simulations also assume that C_{-1} and all A_i, B_i are latched and available at time $t = 0$. Fan-out loading of A_i, B_i and C_{-1} is ignored and the adder delay is calculated as the time required to generate the slowest signal from among S_i and C_{n-1}.

3. Carry Lookahead Optimization

The basic goal of this research is to show how to optimize CLA structures by varying group sizes and lookahead levels. The purpose of these operations is to exploit the delay differences that represent under-utilized time. Early signals can be delayed by modifying the network to make the signals the result of more lookahead computation. Since this allows the addition of larger operands in the same amount of time, slack

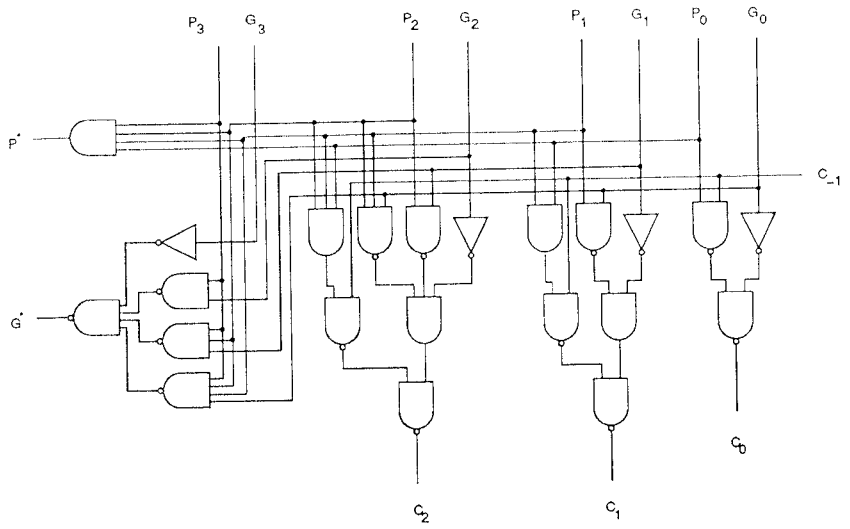

Fig. 5. A 4-bit block carry-lookahead unit with 3-level C_i logic.

reduction corresponds to adder structure optimization. The optimization requires a delay model that accurately measures slack in the circuit, an examination of the delay characteristics of the circuit blocks, and an analysis of critical delay paths to identify slack in the circuit. These requirements lead to a heuristic method for determining optimal CLA structures.

3.1. Delay Model

Logic gate delays are modeled as

$$\text{delay} = f(\text{fi}, \text{fo})$$
$$= A + B \cdot \text{fi} + (D + E \cdot \text{fi}) \cdot \text{fo}.$$

Simpler delay models that use unit gate delays are inadequate because they do not reveal all the slack in the circuit.

The simulations use the following delay functions:

$$t_{NAND} = 0.1058 + 0.1175\text{fi} + (0.0825 + 0.0148\text{fi})\text{fo}$$

$$t_{AND} = 0.2825 + 0.1675\text{fi} + (0.0911 + 0.0037\text{fi})\text{fo}$$

$$t_{INV} = 0.265 + 0.1016\text{fo}$$

$$t_{XOR} = 0.945 + 0.05645\text{fo}.$$

The constants in these functions are based on LSI Logic Corporation's 1.5 Micron Compacted Array™ Technology [8]. To limit complexity, the models assume that that all logic gates are single, possibly large, gate structures rather than multiple levels of smaller gates.

3.2. BCLA/CLA Delay Characteristics

An understanding of BCLA/CLA delay characteristics is important for finding opportunities to reduce slack in CLA structures. In particular, the fan-in and fan-out properties of the circuits must be analyzed.

The input and output loading on the propagate and generate signals of a BCLA can be derived by induction on the circuits of figures 2, 3, 4, and 5. By inspection, the following results are obtained. Given a k-bit BCLA unit connected to the jth input of an m-bit BCLA unit

- G^* of the k-bit BCLA unit has fan-out $(m - j)$.
- G^* of the k-bit BCLA unit has worst case fan-in k. Specifically, in the two-level implementation of G^*, the first level NAND gate i associated with input G_i has fan-in $k - i$.
- P^* of the k-bit BCLA unit has fan-in k.
- P^* of the k-bit BCLA unit has fan-out $(m - j)(j + 1)$

Examples of these relationships are shown graphically in figures 6 and 7. The analysis for a BCLA unit feeding into a CLA unit is similar and gives the same results.

The loading on carry signals may be derived by a similar analysis of the circuit diagrams. Each carry signal, C_i, of a BCLA/CLA unit is the C_{-1} of BCLA units on previous levels. An example of this is shown in figure 8. In particular, it connects to its $(i + 1)th$ fan-in unit, the zeroth fan-in unit of its $(i + 1)th$ fan-in unit, the zeroth fan-in unit of the zeroth fan-in unit of its $(i + 1)th$ fan-in unit, etc. Each carry signal also connects to an XOR gate in the sum generation circuitry. A BCLA unit of size k contributes $k - 1$ to the fan-out loading of its input C_{-1} signal. In the two-level implementation

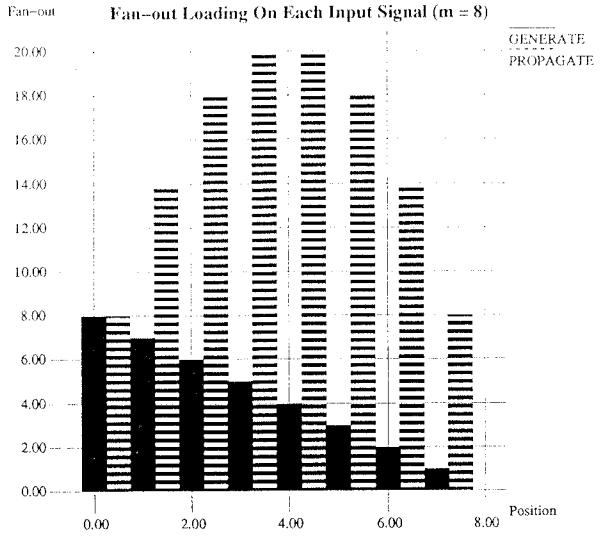

Fig. 6. Fan-out loading on input signals of an 8-bit BCLA/CLA unit.

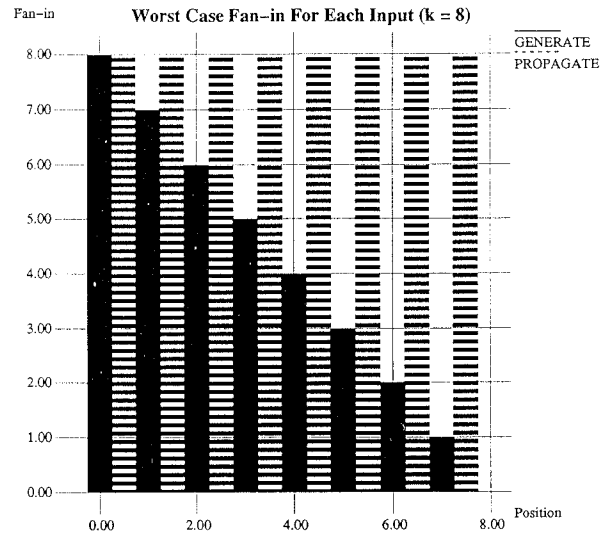

Fig. 7. Worst case fan-in gates for the inputs of an 8-bit BCLA unit.

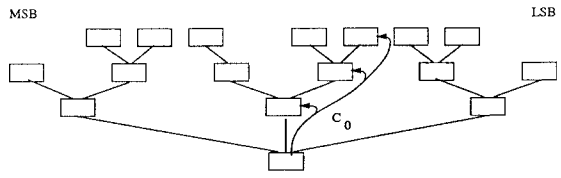

Fig. 8. Fan-out path within the CLA network of C_0 of the final lookahead unit.

of C_i, the worst case first-level NAND gate has fan-in $i + 2$. In the three-level implementation of C_i, C_{-1} always connects to an input NAND gate with a fan-in of 2.

These fan-in and fan-out properties have an important effect on the critical delay analysis in BCLA/CLA units. Unlike typical analyses, the P^* delay cannot be neglected with respect to the G^* delay. Even though computing P^* requires one less level of logic than calculating G^*, the potentially high group propagate fan-out loading may place generation of P^* on the critical path. The P^* delay path should be compared to the G_0-to-G^* delay path which contains the worst case fan-in gate.

Another important delay path is the carry propagation (assimilation) path. This is the path from C_{-1} to some C_i and is critical when C_{-1} arrives much later than all P_i and G_i. Assuming a k-bit BCLA unit, the worst case path for the two-level implementation of C_i is from C_{-1} to C_{k-2}. For the corresponding three-level implementation of C_i, the paths are equal for all i because C_{-1} feeds gates of constant fan-in. When C_{-1}-to-C_i delays are a significant fraction of the total adder delay, the three-level implementation should produce faster adders than the two-level implementation because of its superior fan-in properties. This condition should hold for larger adders.

An issue related to BCLA delay is the relationship between group size and number of lookahead levels. Clearly, larger BCLA units have longer delays than smaller units. Also, adding more levels of BCLA units tends to increase delay because of the extra logic levels. However, at some size, implementing a single large BCLA unit as multiple levels of BCLA units is advantageous. Unfortunately, determining this breakpoint is difficult because the delay of a BCLA unit depends on the block sizes on the next level of lookahead and typically, those sizes are determined by the number and sizes of blocks on all previous levels.

3.3. Critical Path Analysis

Identifying critical paths in CLA structures is important because the remaining noncritical delay paths represent opportunities for slack reduction. Standard CLA analyses assume that the critical path in the adder is always as shown in figure 9. Unfortunately, the validity of this assumption is not guaranteed when variable group sizes and lookahead levels are allowed. However, the actual critical path will have an analogous form. The first part of the critical path is the delay to the generation of a carry signal in the CLA unit and the last part of the critical path is the propagation of this carry signal back through some subtree of the BCLA network. Furthermore, each subtree of the network has an analogous critical path. Opportunities to reduce slack can be found in each portion.

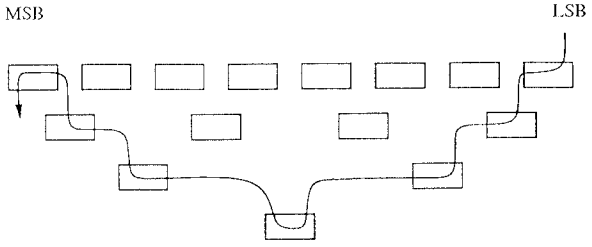

Fig. 9. Critical path assumed by typical analyses.

The delay to the CLA unit depends only on G^* and P^* delays. Most of this delay is expected to be from G_0-to-G^* delays because this path has both high fan-out loading and worst case fan-in gates. On a given level, the critical path might depend on the generation of P^*, but on the next level, the critical path will most likely be the group generate computation because G^* is a function of all the input propagate signals except P_0. The adder delay after the CLA unit depends only on carry propagation delays. All P_i and G_i signals have already settled and the critical path follows the worst C_{-1}-to-C_i path in a BCLA unit on each level.

Assuming that the generation of G^* is the critical path through a BCLA unit, then the group feeding G_i can be larger than the group feeding G_j for $i > j$, since G_j feeds a higher fan-in gate. This will reduce the slack between the different G_i-to-G^* delays. This argues for fewer levels of smaller lookahead units in the least significant bits than in the most significant bits of every subtree of lookahead units.

The slack in the second portion of a critical path arises from different C_i delay times in a lookahead unit. The subtrees fed by C_j should be faster than those fed by C_i for $j > i$. This indicates that fewer levels of smaller lookahead units should handle the most significant bits of each subtree than should handle the least significant bits.

The criteria for each portion of the critical path contradict each other. This indicates that the best opportunities to increase group sizes or add levels of lookahead may occur in the middle bits of each subtree rather than at the ends. Unfortunately, this criteria cannot be used to make specific decisions except for trivial cases. The problem is that the fan-in and fan-out dependencies of the delay model interfere with local optimizations. The whole adder must be analyzed because decisions on each level depend on the sizes of the next level which depend on the structure of the previous levels. A dynamic programming method is used to avoid the complexity of analyzing size combinations for entire adder structures.

3.4. Heuristics

The basic algorithm for finding an optimal CLA structure is based on dynamic programming. The generic pseudo-code for this approach is shown in figure 10. The problem of finding the optimal n-bit adder structure is reduced to a series of subproblems. Each subproblem requires finding an optimal $(k + b)$-bit adder structure given a k-bit adder structure. Note that the number of subproblems is bounded by $n - k_0$. The main components of the algorithm are the transformation step and the choice of initial adder structure.

The transformation step increases adder size by adding levels and/or increasing group sizes. Greedy heuristics are used to determine the location and magnitude of these increments. When increasing the size of a group, the extra bits are added to the most significant end of the group and the inputs to these new positions are connected directly to the initial propagate and generate generation circuitry. The solution of the transformation problem is simplified by formulating the decision to increase the number of lookahead levels as a decision to increase group sizes. This reduction is obtained by viewing all the inputs into the carry-lookahead network as groups of size one. Thus, increasing the size of a 1-bit *input* group often corresponds to increasing the number of lookahead unit levels in the adder. Increasing the size of such an input group does not always increase the number of levels because for any given network, multiple input groups may exist which will increase the number of lookahead levels and only the first one of these groups which is enlarged will actually increase the number of levels.

The choice for the initial adder structure is constrained to single CLA units. Ideally, the initial structure should be optimal and should admit a transformation path to a final optimal structure. The results show that the simple constraint of starting with a single CLA unit is effective.

Many different heuristics can be used to perform the transformation step and to choose the initial CLA unit size. Two different sets of heuristics were used to implement two different versions of the basic algorithm.

```
let i = 0
let initial structure = k0-bit adder

while ( ki ≤ n )
   transform the current ki-bit adder into a (ki + bi)-bit adder, bi ≥ 1
   let ki+1 = ki + bi
   let i = i + 1
end
```

Fig. 10. Basic algorithm for finding the structure of an n-bit adder.

The first version runs the basic algorithm for different starting CLA unit sizes and chooses the best structure from among all the runs. The current range of starting sizes is from 2 to 16. The transformation step constrains $b_i = 1$, $\forall i$. Each block in the network except for the CLA unit is considered for receiving this extra bit. For each block, the delay of the structure that results from increasing the block size by one is calculated. The transformation is performed by enlarging the block that results in the adder structure with the shortest delay.

The second version always starts with a 2-bit CLA unit and then uses a more complex transformation step. The basic goal of the transformation heuristic is to achieve the maximum increase in adder size per unit delay increase. Increasing by one the size of some group in an adder of size k and delay d results in an adder of size $k + 1$ and delay $d + \delta$. Depending on the location of the enlarged block, the size of other groups may also be increased without further increase in adder delay. In general, it is possible to increase the size of other groups to produce an adder of size $k + b$ and delay $d + \delta$ where $b \geq 1$. Given the selection of the initial enlarged group, the location of the remaining $b - 1$ bits can be found by adding as many extra bits as possible to each group of the network such that the adder delay remains $d + \delta$. The value of $b - 1$ is maximized by processing groups by level, starting with the final CLA unit and working back to the initial propagate and generate circuitry. Within levels, groups are processed from least significant to most significant. The transformation heuristic at step i calculates for each group j in the network

$$\text{benefit}_j = \frac{\min(b_j, n - k_i)}{\delta_j}$$

where

δ_j = increase in delay caused by increasing the size of group j by 1

b_j = maximum increase in adder size possible given the selection of group j for the initial 1 bit increment and a delay increase limit of δ_j

k_i = size of adder before transformation step i

and

n = desired final adder size.

The group j with the largest benefit value is chosen to transform the k_i-bit adder into a $(k_i + \min(b_j, n - k_i)$-bit adder.

A similar method has been implemented by [5]. That method also uses dynamic programming and the recursive step increases adder size by determining an optimal subtree of BCLA units to connect to a particular input position of the final CLA (BCLA) unit. Input positions are processed from least significant to most significant. In contrast, the method described here increases adder size by increasing group sizes or levels anywhere in the transitive fan-in of the final CLA unit. Also, the method of [5] requires delay models that are linear monotone nondecreasing non-negative functions of fan-in and fan-out and only optimizes the carry-lookahead network. This work allows delay models that are any arbitrary function of fan-in and fan-out and optimizes the whole adder. Simulating the whole adder exploits the delay differences between the initial propagate and generate signals and properly accounts for loading by the sum generation circuitry. Some comparisons with the results of the method in [5] are given in the next section.

4. Results

Figures 11 and 12 show results for the two different implementations of BCLA/CLA units. Each graph has three curves corresponding to requiring fixed-sized groups and fixed number of levels (**Fixed**), allowing different sized groups only on different levels and requiring fixed levels (**Inter-level**), and allowing variable sizes and levels anywhere (**Variable**). The delay values are for the best structures of each category. The **Fixed** and **Inter-level** curves were obtained by simulation of all possible combinations. If an integral number of groups of the chosen size for a level handled more

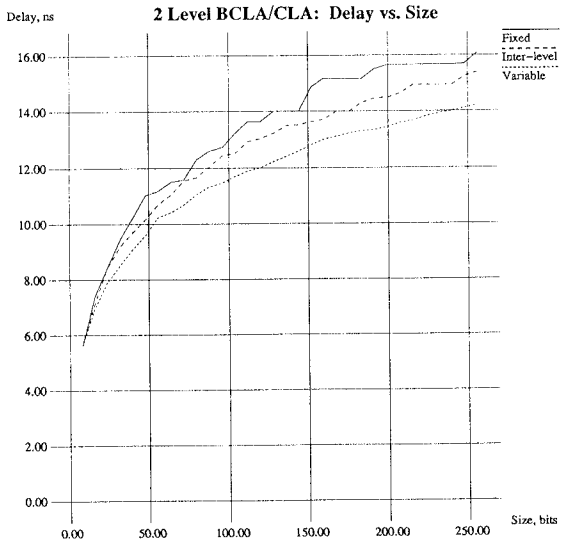

Fig. 11. Delay versus adder size in adders using a 2-level BCLA/CLA carry implementation.

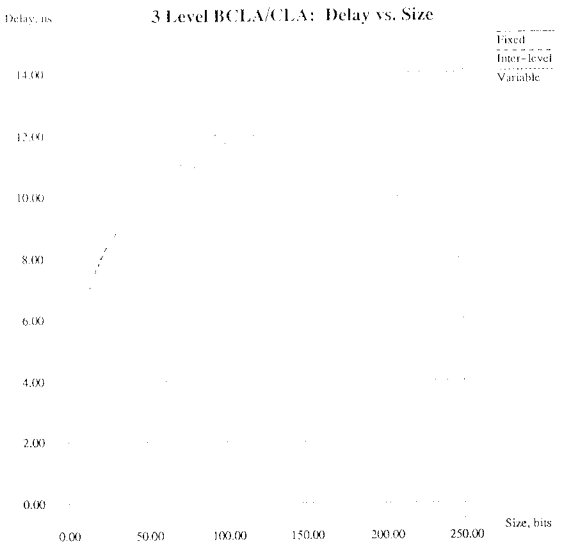

Fig. 12. Delay versus adder size in adders using a 3-level BCLA/CLA carry implementations.

signals on that level than was necessary, then the size of the group corresponding to the most significant bits of the adder was decreased to eliminate the excess capability. The **Variable** curve for the two-level BCLA/CLA carry implementation (figure 11) was obtained using the first set of heuristics. This version of the algorithm worked well for this lookahead implementation but not as well for the three-level implementation. The **Variable** curve for the three-level BCLA/CLA carry implementation (figure 12) was obtained using the second set of heuristics since it produced better results than the first set.

Figure 13 gives the optimal 32-bit adder structure found by each algorithm version. The 1's represent input from the initial bit positions.

The results correspond well to theory. Adders of fixed-size groups are slower than adders allowing variable inter-level groups. Adders with variable groups and levels are faster than both other types. The optimal **Variable** structures tend to have more levels and larger group sizes in the middle of the adder than on the ends. Results of comparing the different lookahead implementations are mixed. The **Fixed** and **Inter-level** three-level implementations performed much better than the two-level implementations. However, the heuristics improved the performance of the two-level implementations more than they improved the three-level implementations. The delay differences between the **Variable** adders of the two implementations is smaller, though the three-level implementation is still faster for larger adders.

Table 1 compares the delays of adders generated from the carry-lookahead network configurations of [5] with the delay adder structures found by the first version of the basic algorithm. The comparisons are based on the two-level implementations of the BCLA/CLA units. The delay models are those used in [5] plus an equivalent XOR delay model. The delay functions used are:

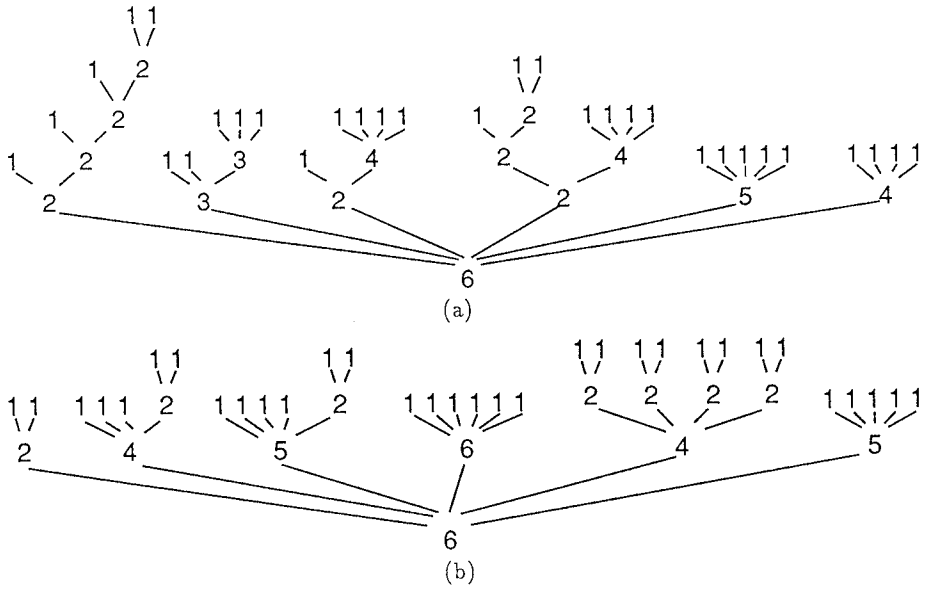

Fig. 13. 32-bit adders found by (a) the first set of heuristics using 2-level BCLA/CLA units (delay = 8.5 ns) and (b) the second set of heuristics using 3-level BCLA/CLA units (delay = 8.9 ns).

$$t_{NAND} = 5\text{fi} + 20\text{fo}$$

$$t_{AND} = 17 + 20\text{fi} + 5\text{fo}$$

$$t_{INV} = 12 + 5\text{fo}$$

$$t_{XOR} = 5\text{fo}.$$

The adder structures found by the first version of the basic algorithm are faster in all cases.

Table 1. Adder *sum* delays for configurations from [1] and for adders found by the first version of the basic algorithm.

	Delay	
Adder Size, bits	Chan, et al.	First Algorithm
8	407	394
16	524	514
24	627	584
32	652	637
48	736	721
64	821	779
66	824	779
84	877	834

Table 2 compares the adder structures corresponding to the delays in table 1. The structures are represented in parentheses notation as in [5]. For example, the adder structures shown in figures 13(a) and 13(b) are represented as

((1 (1 (1 (1 2)))) (1 1 3) (1 4) ((1 2) 4) 5 4)

and

(2 (1 1 1 2) (1 1 1 1 2) 6 (2 2 2 2) 5),

respectively. The numbers in the expression represent the block sizes at the highest level.

5. Conclusion

Varying group sizes and lookahead levels improves the performance of commonly used CLA implementations. Unfortunately, finding these improved structures is difficult because of delay fan-in and fan-out dependencies. In general, the whole adder structure must be known before a decision to increase group sizes or the number of lookahead levels can be made. Fortunately, simple heuristics can deal effectively with this problem. Simulation results show that heuristic methods can find CLA adder structures with variable group sizes and levels that are faster than more constrained carry-lookahead adders.

Table 2. Adder structures corresponding to the delay values in table 1.

Adder size, bits	Source	Adder Structure
8	Chan, et al.	(1 1 2 2 2)
	First Algorithm	(1 2 3 2)
16	Chan, et al.	(((1 2) (1 1 2) 4 (2 1)) 2)
	First Algorithm	((1 (1 2)) (1 1 2) 4 (2 1 1))
24	Chan, et al.	((1 3) (2 (2 2)) (2 3 2) (3 2 2))
	First Algorithm	(1 (1 (1 2)) (1 1 2) 4 4 (2 1 1) 3)
32	Chan, et al.	(((1 (1 (1 2))) ((1 2) ((1 2) 3)) (2 3 2) (3 2 2)) 2 2)
	First Algorithm	(((1 (1 2)) (1 1 3) (1 1 2 2) (2 3 2) (3 2 2)) 2 1)
48	Chan, et al.	(((1 2) ((1 (1 2) (2 2)) ((1 2) (1 2) (3 2)) ((1 2) 3 3 2) (4 3 3))) 3 2)
	First Algorithm	(((1 (1 (1 2))) (1 1 (1 1 2)) (1 1 (1 2) 4) ((1 2) 3 (2 1 1)) (4 (2 1 1) 3)) (2 1 1) 3)
64	Chan, et al.	((2 3 ((1 2 3 3) (2 3 4 (2 2)) ((1 2 2) (2 2 2) (3 2)) ((2 2 2) (3 2) 3)) (3 2)) 2)
	First Algorithm	(((1 (1 (1 (1 2)))) (1 1 (1 1 3)) (1 1 (1 2) 4) ((1 (1 2)) (1 1 2) (2 2 2)) ((1 1 2 1 1) (3 2 2) (3 2 1))) (2 2 1) (2 1 1))
66	Chan, et al.	((2 3 ((1 2 3 3) (2 3 4 (2 2)) ((1 2 2) (2 2 2) (3 2)) ((3 2 2) (3 2) (2 2))) (3 2)) 2)
	First Algorithm	(((1 (1 (1 (1 2)))) (1 1 (1 1 3)) (1 1 (1 2) 4) ((1 (1 2)) (1 1 3) (2 3 2)) ((1 1 2 1 1) (3 2 2) (3 2 1))) (2 2 1) (2 1 1))
84	Chan, et al.	((3 4 ((2 3 (1 2 2) (3 2 2)) ((1 2 2) (2 3 2) (3 2 2)) ((2 3 2) (3 2 2) (3 2)) ((3 2 2) (3 2) 3)) (3 2)) 2)
	First Algorithm	(((1 (1 (1 (1 2)))) (1 1 (1 1 (1 1 2))) (1 1 (1 (1 2)) (1 1 2 2)) ((1 (1 2)) (1 1 3) ((1 2) 3 3)) ((1 1 2 2 2) (3 3 3) ((2 1 1) 3 2))) (3 3 2) (3 2 1))

Work is in progress to re-run the simulations in this paper for ECL delay data. This is particularly important in the domain of high performance machines where bipolar is the dominant technology. Also, more heuristics and implementations will be examined. For example, a hybrid of the two sets of heuristics presented here will be tried and an adder structure that uses the three-level BCLA carry implementation with the two-level CLA carry implementation will be tested.

Acknowledgments

The authors would like to thank Professor Pak Chan for sharing his ideas and insight on carry-lookahead adders. His many helpful discussions and suggestions are greatly appreciated.

References

1. R. Sherburne, Jr., "Processor design tradeoffs in VLSI," Technical Report UCB/CSD 84/173, University of California, Berkeley, 1984.
2. J. Sklansky, "An evaluation of several two-summand binary adders," *IRE Trans.*, EC-9(2), 1960, pp. 213–226.
3. V.G. Oklobdzija and E.R. Barnes, "On implementing addition in VLSI technology," *Journal of Parallel and Distributed Computing*, 5, 1988, pp. 716–728.
4. S. Turrini, "Optimal group distribution in carry-skip adders," In *Proceedings of the 9th Symposium on Computer Arithmetic*, 1989, pp. 1–18.
5. P.K. Chan, M.D.F. Schlag, C.D. Thomborson, and V.G. Oklobdzija, "Delay optimization of carry-skip adders and block carry-lookahead adders," in *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, June 1991, pp. 159–164.
6. B.D. Lee and V.G. Oklobdzija, "Optimization and speed improvement analysis of carry-lookahead adder structure," In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, 1990, pp. 918–922.
7. K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, New York: Wiley & Sons, 1979.
8. LSI Logic Corporation, *Databook: 1.5 Micron Compacted Array™ Technology*, 1987.

Brian D. Lee (S '82) received his B.S. and M.S. degrees in electrical engineering from the University of California, Berkeley in 1985 and 1989, respectively. He is currently working towards the Ph.D. degree in electrical engineering at that institution.

His research interests include CAD for VLSI, computer architecture, and computer arithmetic.

Vojin G. Oklobdzija obtained Dipl. Ing. degree from the Electrical Engineering Department, University of Belgrade, Yugoslavia in 1971. He came to the U.S. in 1976 as a Fulbright Scholar to study computer science and he obtained his M.Sc. and Ph.D. degrees from the University of California in 1978 and 1982. He also worked at VLSI division of Xerox Corporation during the last phase of his Ph.D. program. In 1982 he moved to T.J. Watson Research Center of IBM Corporation and worked on several programs developing IBM RISC architecture. He holds several patents in the area of RISC architecture and VLSI. Most recently he became co-holder of the patent on IBM RS/6000 architecture. In 1988 he left IBM and was teaching computer architecture and design at the University of California Berkeley. In October of 1991 he resumed working at IBM T.J. Watson Research. His work and interest has been in the areas of Computer Arithmetic, High Performance Architectures and VLSI oriented arithmetic and implementations. Most recently he has been working on development of parallel super-computer architecture.