1999

A 1.0 nsec 32-bit prefix tree adder in 0.25 micron static CMOS

Alexander Goldovsky
Lehigh University

Goldovsky, Alexander

A 1.0 nced 32-bit prefix tree adder in 0.25 micron static CMOS

May 31, 1999

A 1.0 nsec 32-bit Prefix Tree Adder in 0.25 Micron Static CMOS.

by

Alexander Goldovsky

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Department of Electrical Engineering and Computer Science

Lehigh University

May 1999

This thesis is accepted and approved in partial fulfillment of the requirements for Master of Science.

05/05/99

Date

Thesis Advisor

Chairperson of Department

Acknowledgments

Although this work bears my name, it is a result of the efforts and contributions of several people. First and foremost, I would like to thank Professor Michel Schulte, my thesis supervisor, for his original contributions and help throughout the project. His willingness to discuss ideas, and enthusiasm have made it a unique and enjoyable learning experience for me.

I am also grateful to Ravi Kolagotla of Lucent Technologies, Microelectronics for helping me learn about different adder architectures, assisting me in testing of the shuttles and helping me in implementing this project. He has been a good friend and a source of inspiration.

I would also like to thank the following people for their contribution to the project: Chris Nicol, for his ideas on superimposed adder architecture, Mat Besz, Tracey Dellarova, Scott van-Horn and David Koehler for helping with the layout design of the adders, and Vance Archer for his help in including the adders on the Lucent 0.25 micron test shuttle.

Table of Contents

List of Tables vi

List of Figures vii

Absrtact 1

1. Introduction 2

1.1 Carry-lookahead addition 2

1.2 Binary addition 4

1.3 Implementation issues 6

2. Existing architectures for binary addition 7

2.1 Carry-ripple adder 8

2.2 Carry-skip adder 8

2.3 Carry-lookahead adder 10

2.4 Brent-Kung adder 11

2.5 Superimposed tree CLA 13

2.6 Superimposed prefix tree Ling adder 17

3. New architecture for superimposed prefix tree CLA 19

3.1 New architecture for prefix tree adder 19

3.2 Improved architecture of prefix tree adder for low power 21

4. Prefix tree CLA architecture comparison 24

4.1 Architecture comparison 24

4.2 VLSI implementation of 32-bit architecture 24

4.3 Test results 25

4.4 Test structure 28

Conclusions and Future Work 30

References 31

Appendix 34

Vita 46

List of Tables

Table 1. Truth table for the full adder cell.

Table 2. Comparison of different prefix tree adder architectures.

Table 3. Lucent 0.25-um CMOS process parametrics.

Table 4. Comparison with other 32-bit adders.

Table 5. Measured performance of the different implementations of the adder in Fig. 7.

List of Figures

Figure 1. Modified full adder for fast carry path propagation.

Figure 2. Modified full adder for fast carry path propagation with carry buffering.

Figure 3. XOR and XNOR mux-type structure.

Figure 4. N-bit carry-ripple adder design.

Figure 5. A 16-bit carry-skip adder design using multi-level skip and variable block size.

Figure 6. A 16-bit Brent-Kung adder architecture.

Figure 7. A 16-bit superimposed prefix tree CLA adder architecture.

Figure 8. A 16-bit superimposed prefix tree CLA adder architecture with modification to *Go*.

Figure 9. New architecture for 16-bit prefix tree adder with carry incorporated into the tree.

Figure 10. New architecture for 16-bit prefix tree adder with carry incorporated into the tree with low power solution.

Figure 11. Speed vs. transistor count of different 32-bit adders in 0.25-um. technology.

Figure 12. AND gate delay measurement results.

Figure 13. Divide network for testing the adder off-chip.

Figure 14. Examples of applications for a prefix tree adder with carry-in incorporated into the tree.

Figure A-1. 32-bit circuit implementation of the carry-skip adder.

Figure A-2. 32-bit circuit implementation of the Brent-Kung adder.

Figure A-3. 32-bit circuit implementation of the Ling adder.

Figure A-4. 32-bit circuit implementation of the superimposed tree CLA, according to Fig. 7.

Figure A-5. 32-bit circuit implementation of the superimposed tree CLA with carry in incorporated into the tree, according to Fig. 9.

Figure A-6. 32-bit circuit implementation of the superimposed tree CLA with carry in incorporated into the tree (low power solution), according to Fig. 10.

Figure A-7. Layout of the 32-bit prefix tree adder according to Fig.7.

Figure A-8. Layout of the 32-bit prefix tree adder according to Fig.9.

Figure A-9. Circuit diagram for the test structure on the shuttle.

Figure A-10. Two different layout styles according to Table 5 [Section 4.3].

Figure A-11. Photograph of the die.

Abstract

The carries in a carry-lookahead adder can be computed by using a separate prefix tree for each bit location. This is nearly twice as fast as the standard Brent and Kung addition technique. This thesis shows that the primary carry input signal can be incorporated into the prefix trees without incurring any additional delay. The proposed architecture reduces the logic depth of an n-bit adder by one, compared to existing architectures. Using this technique with fully-static circuits, a 32-bit, radix-2 prefix tree adder has a delay of 1.0 nsec in the Lucent 0.25um CMOS Technology. The proposed adder architecture can easily be extended to other word sizes and radices.

Chapter 1: Introduction.

1.1 Carry Lookahead Addition

With ever shrinking VLSI process geometries, transistor count and chip area are becoming secondary considerations to delay and power. Hence, it is necessary to reexamine the tradeoffs that have been made in existing designs and implementations of computer arithmetic algorithms.

The carry lookahead technique, first reported by Weinberger and Smith, speeds up the addition process by unrolling the recursive carry equation [1]. Both transistor count and interconnection complexity have typically limited unrolling to four bits. Larger adders have been built as block carry-lookahead adders, where the lookahead operation occurs within small blocks [2].

The recursive carry-computation can also be reduced to a prefix computation [3] and [4]. With this technique, a prefix tree computes the carry at the most-significant bit position, and an additional tree superimposed on the prefix tree is used to compute the intermediate carries [5]. Faster computation of the carries can be achieved by using a separate prefix tree for each bit position [6], [7], however, this approach requires more hardware.

Full prefix tree adders, also known as Kogge-Stone adders, have not been frequently used because of the additional delay and area introduced by their exponentially growing interconnect complexity [2], [8]. Existing architectures have emphasized the reduction of interconnection complexity at the expense of higher gate fanouts [6], [9]. Interconnection complexity can also be reduced by using hybrid carry-lookahead / carry-select architectures, which eliminate the need to implement a full prefix tree for each bit position [10]. The imminent, widespread use of low R and C materials [11] reduces the negative effects of architectures that depend on large amounts of interconnect [12]. Furthermore, with additional levels of interconnect, the area overhead of implementing these adders is alleviated through the use of extensive over-the-cell routing, which removes the routing channels and further minimizes the interconnect capacitance.

This thesis show that the primary carry input can be incorporated into the full prefix tree adder without additional overall delay. To demonstrate the benefits of this approach, 32-bit prefix tree adders were implemented with the carry input based on both the existing and proposed architectures. Both implementations use fully static circuits in the standard Lucent 0.25-um CMOS technology. Static circuits are preferred to dynamic circuits for their low power, and their ease of design. The measured delay of the adder with the existing architecture is 1.1 nsec, while the measured delay of the adder with the proposed architecture is 1.0 nsec. This delay is expected to be lower if the adders are implemented in technologies with lower interconnect RC delays [12].

1.2 Binary addition

The addition of two numbers,

$$A = -a_{n-1} \cdot 2^{n-1} + \sum_{j=0}^{n-2} a_j \cdot 2^j \quad \text{and} \quad B = -b_{n-1} \cdot 2^{n-1} + \sum_{j=0}^{n-2} b_j \cdot 2^j$$

represented in two's complement binary form, can be accomplished by computing:

$$g_j = a_j \cdot b_j$$

$$p_j = a_j \oplus b_j$$

$$c_j = g_j + p_j \cdot c_{j-1}$$

$$s_j = p_j \oplus c_{j-1}$$

where $0 \le j < n$ and c_{-1} is the primary carry-input. An overflow occurs, and the resulting sum is invalid, if

$$c_{n-1} \oplus c_{n-2} = 1$$

A straightforward implementation of a 1-bit adder is achieved through a full adder, which implements the above equations. In an n-bit addition, the longest path through the adder is the carry propagation path. Hence, the design of the full adder is optimized to minimize the delay of the carry path. The diagram for a modified full adder is shown below. The following equations are used to implement the full adder:

$$c_j = a_j \cdot \overline{p_j} + p_j \cdot c_{j-1}$$

$$s_j = a_j \oplus b_j \oplus c_{j-1}$$

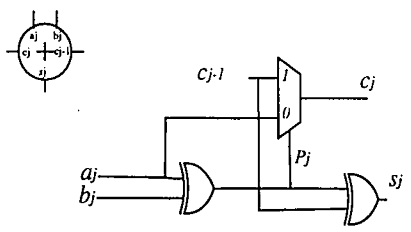


FIGURE 1. Modified full adder for fast carry path propagation.

In practice, when designed in 0.25um technology, all pass-gate structures, such as muxes and latches, are buffered for better performance. This also applies to the full adder cell. The carry out signal is inverted to buffer the 2-to-1 mux. The full adder which follows the one with the inverted carry out signal recovers the positive logic, as shown in Figure 2B.

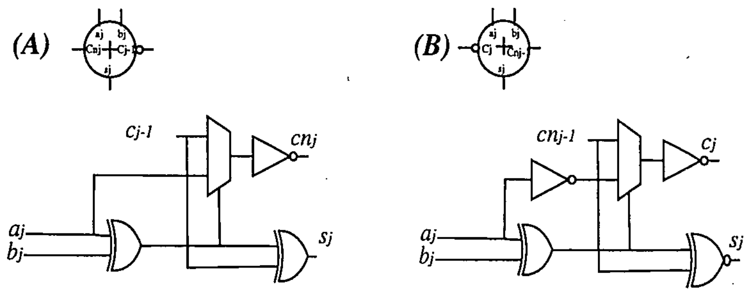


FIGURE 2. Modified full adder for fast carry path propagation with carry buffering.

Section 1.3 covers basic blocks for the adder design in 0.25um technology. Assuming, the delay for an inverter cell (2 transistors) is 0.5Δ, the delay for XOR/ XNOR (10 transistors) is 2Δ and the delay for a mux (4 transistors) is 1Δ, the following hold for the diagram in Figure 2(B):

$$\text{Number of transistors} = 28$$

$$\text{logic depth from } c_{j-1} \text{to } s_j = 2\Delta$$

$$\text{logic depth from } c_{j-1} \text{to } c_j = 1.5\Delta$$

$$\text{logic depth from } a_j \text{to } s_j = 4\Delta$$

$$\text{logic depth from } a_j \text{ to } c_j = 3.5\Delta$$

1.3 Implementation issues.

With shrinking process geometries, it is critical to find the best implementation for the basic cells, due to threshold voltage limitations of static CMOS technology. Simulation shows that for high performance 0.25um designs should have 2-input NOR gates, 2-input or 3-input NAND gates, and full CMOS type structures such as muxes and latches.

XOR and XNOR cells are designed using mux-type structures as shown in Figure 3. Although the transistor count is larger than typical implementations of XOR and XNOR gates, these designs are about 20% faster than the standard weak PMOS pull-up structures in 0.25 um technology at 1.6 Volts. These XOR and XNOR designs each require 10 transistors.

All the mux structures are designed with fully complementary pass gates, with local buffering to reduce capacitive loading on the output nodes.

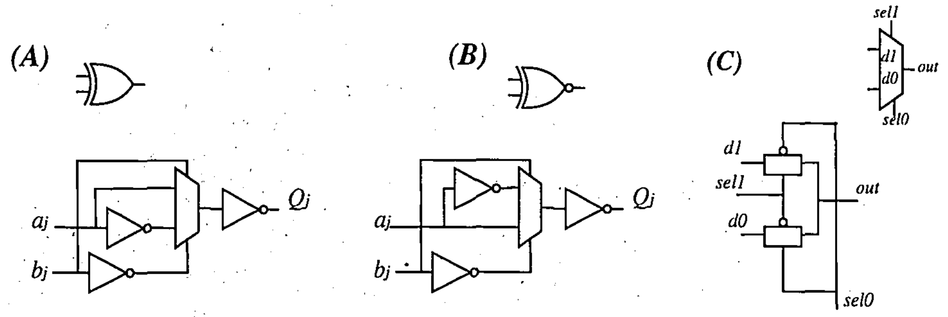


FIGURE 3. XOR and XNOR mux-type design.

2 Existing architectures for binary addition.

Parallel adders can be classified into two categories based on the way in which internal carries from stage to stage are handled; ripple carry and lookahead carry. Externally, both types of adders are the same in terms of inputs and outputs. The differences are the speed at which they operate, the area of the layout, and the power consumption. Carry-ripple adders, as described in Section 2.1, are simpler to design, require very little area and hardware. However, they are also the slowest type of adder. Hence, if the performance is not an issue, carry-ripple adders are often used. To improve the speed of binary addition, carry-skip techniques, as described in Section 2.2, were introduced. Although, it takes more hardware to incorporate the skip logic, the gain in speed is more than 200% for 16-bit implementations, according to simulations data presented in Figure 11. When designing even faster adders, it is essential to get around the rippling effect of the carry that is present in both carry-skip and carry-ripple adders. The carry-lookahead principal offers a possible way to do so. This thesis describes 3 types of carry-lookahead adders: Brent-Kung adders [Section 2.2], Ling adders [Section 2.3], and superimposed prefix tree adders [Section 2.4].

2.1 Carry-ripple adder design.

The ripple carry adder (RCA) provides a slow, but hardware efficient, method for adding two binary numbers. An n-bit RCA is formed by cascading *n* full adders (FAs). For high speed, two types of FAs are used, as described in Section 1.2. The carry out of j^{th} FA is used as the carry in of the $(j+1)^{th}$ FA, as shown in Figure 4. The carry propagation delay for each full adder is the time from the application of the input carry until the output carry is valid, assuming the *a* and *b* inputs are already present. For the *n*-bit RCA, the number of transistors and number of logic stages (or logic depth) are

$$\text{Number of transistors} = 28 \cdot n$$

$$\text{depth for RCA adder} = 4\Delta + (1.5\Delta \cdot (n-2)) + 2\Delta = (1.5n+3)\Delta$$

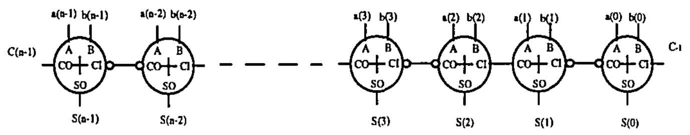


FIGURE 4. N-bit RCA design.

2.2 Carry-Skip adder design.

To reduce the carry propagation path of the RCA, the carry-skip adder (CSKA) is introduced, where each carry is evaluated from the previous adder stages. The CSKA is based on the following observation. The propagation process can skip any adder stage for which $a_j \neq b_j$, or equivalen[illegible] $a_j \oplus b_j = 1$. Several stages can be skipped if all satisfy $a_j \neq b_j$. Thus, an adder consisting of *n* stages is divided into blocks of consecutive stages with a simple RCA scheme used within each block. Every block of length k (which is called the block size), also generates a block-carry-propagate signal that is defined as

$$P_j^{j+k-1} = p_j \cdot p_{j+1} \cdots p_{j+k-1}$$

The carry out of block k is expressed as

$$c_{j+k} = P_j^{j+k-1} \cdot c_j + G_j^{j+k-1}$$

where c_{j+k} is the carry out of the last FA in subgroup k.

A good strategy when designing CSKA is to vary the block size to optimize the carry propagation timing. Also, for improved performance multi-level skip is performed. This is illustrated in Fig. 5. Empty squares represent full adders, filled rectangles implement skip equations, dashed lines are propagate signals, and solid lines are carry paths. Appendix [Figure A-1] shows the circuit implementation of a 32-bit carry-skip adder.

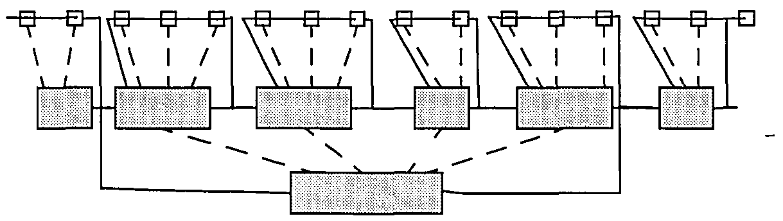


FIGURE 5. A 16-bit carry-skip Adder design using multi-level skip and variable block size techniques.

2.3 Carry-lookahead adder design.

In the RCA, the speed with which an addition is performed is limited by the time required for carries to propagate, or ripple, through all stages of the adder. One method for speeding up the addition process is to eliminate this ripple carry delay by carry look-ahead addition. This method is based on the two ways in which the full-adder produces an output carry: carry generation and carry propagation. Carry generation occurs when an output carry is set independently of the carry input. A carry is generated only when both a_j and b_j is *one*. The generate signal is expressed as:

$$g_j = a_j \cdot b_j$$

An input carry is propagated by the full-adder when either a_j or b_j is *one,* as discussed in Section 2.3. The propagate signal is expressed as:

$$p_j = a_j \oplus b_j$$

Alternatively, a transmit signal can be used, where

$$t_j = a_j + b_j$$

The truth table for carry generation and carry propagation conditions are shown in Table 1.

Table 1: Truth table for the full adder cell.

a_j	b_j	c_{j-1}	p_j	t_j	g_j	s_j	c_j
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	1	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	1	1	0	0	1
1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	1

From Table 1, we can derive the relationship between the carry in c_{j-1} and carry out c_j:

$$c_j = g_j + p_j \cdot c_{j-1} \text{ or } c_j = g_j + t_j \cdot c_{j-1}$$

As mentioned previously, an overflow occurs when $c_{n-1} \oplus c_{n-2} = 1$.

2.4 Brent and Kung adder.

The Brent and Kung adder uses a generic associative operator, called the dot operation (*o*). The associativity property implies that the following statement is valid: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$. Under these conditions, combining "*n*" arguments using the dot operator "*o*" can be executed with critical path equal to $\lceil \log_2 n \rceil \cdot t_o$, where t_0 is the propagation delay of the dot-operator, defined later as a propagation stage. This property can be applied to an *n*-bit adder. This requires the definition of a "*o*" operator that establishes the following relationship between two tuples (g_j, p_j).

As in [5] and [6], we define $(G_j^j, P_j^j) = (g_j, p_j)$, and

$$(G_i^j, P_i^j) = (g_j, p_j) o (g_{j-1}, p_{j-1}) o \ldots o (g_i, p_i) \text{ if } j > i$$

where "*o*", the fundamental carry operator, is associative [5]. In particular for radix-2 operation, the "*o*" operator is a function that takes in two sets of two inputs (g_j, p_j) and

(g_i, p_i) and produces a set of two outputs (G_i^j, P_i^j) . At each bit position, the carry is given by

$$c_j = G_0^j + P_0^j \cdot c_{-1}$$

where the c_{-1} is the primary carry input. If there is no primary carry input, then c_j is simply G_0^j . This is illustrated in Fig. 6 for a 16-bit adder. Filled circles implement the fundamental carry equation, empty circles are buffers, and empty squares compute first order of propagate and generate signals. The circuit diagram is shown in the Appendix [Figure A-2].

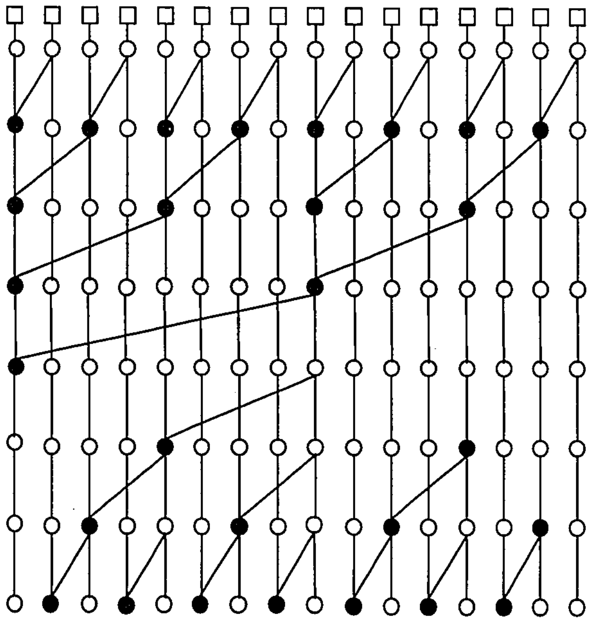


FIGURE 6. A 16-bit Brent and Kung adder [5] where the carries propagate from top to bottom.

The number of stages needed to implement a Brent-Kung adder is $2 \cdot \lceil \log_2 n \rceil$.

2.5 Superimposed tree CLA adder.

With a superimposed tree CLA, the computation of (G_0^j,P_0^j) for $0 \le j < n$ from $p_0 \cdots p_{n-1}$ and $g_0 \cdots g_{n-1}$ can be accomplished in $\lceil \log_2 n \rceil$ stages [3], [6]. A complete adder is constructed by implementing the following steps.

Step 1 (1 stage)

calculate $g_j = a_j \cdot b_j$ and $p_j = a_j \oplus b_j \quad 0 \le j < n$

Step 2 ($\lceil \log_2 n \rceil$ stages)

For k=1 to $\lceil \log_2 n \rceil$ calculate

$$(G_0^j,P_0^j) = (G^j_{j-2^{k-1}+1},P^j_{j-2^{k-1}+1})o(G_0^{j-2^{k-1}},P_0^{j-2^{k-1}})$$

$$2^{k-1} \le j < 2^k - 1$$

$$(G^j_{j-2^k+1},P^j_{j-2^k+1}) = (G^j_{j-2^{k-1}+1},P^j_{j-2^{k-1}+1})o(G^{j-2^{k-1}}_{j-2^k+1},P^{j-2^{k-1}}_{j-2^k+1})$$

$$2^k - 1 \le j < n$$

Step 3 (1 stage)

calculate $c_j = G_0^j + P_0^j \cdot c_{-1} \quad 0 \le j < n$

Step 4 (1 stage)

calculate $s_j = p_j \oplus c_{j-1} \quad 0 \le j < n$

This is illustrated in Fig.7 for a 16-bit adder. The open squares at the top compute p_j and g_j for each bit position according to step 1. The empty circles apply the fundamental carry operator according to step 2, and the filled circles represent buffers. The last stage shown in Fig. 6 using crossed circles applies c_{-1} to every (G_0^j, P_0^j) according to step 3. The output of this array is the carry at each bit position.

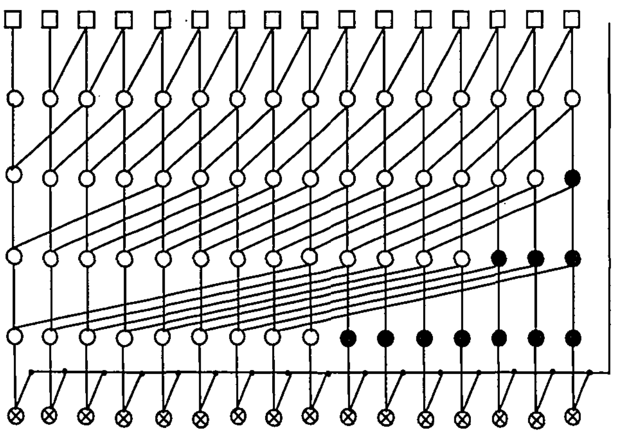


FIGURE 7. Computation of the carry equation using prefix trees for a 16-bit adder/subtractor or a 16-bit adder with carry input. The empty circles implement the fundamental carry operator, the filled circles are buffers, the crossed circle implement the equation in step 3, the empty squares compute generate and propagate signals.

An additional stage (not shown) is needed to generate the sum at each bit position from p_j and c_{j-1} according to step 4. The logic depth of this adder is $3 + \lceil \log_2 n \rceil$. If there is no carry input, then the last stage shown in Fig.7 is not needed.

Alternatively, the contribution due to the carry input can be incorporated by redefining the first generate in adder/subtractor as [6]

$$g_0 = a_0 \cdot b_0 + (a_0 + b_0) \cdot c_{-1}$$

with this change

$$G_0^j = c_j \quad 0 \le j < n$$

This is illustrated in Fig. 8 for the 16-bit adder. This replaces the hardware required to implement step 3 above and reduces the fanout on the c_{-1} input from n to 1. However, the logic depth remains $3 + \lceil \log_2 n \rceil$, and the overall theoretical delay of the adder is unchanged.

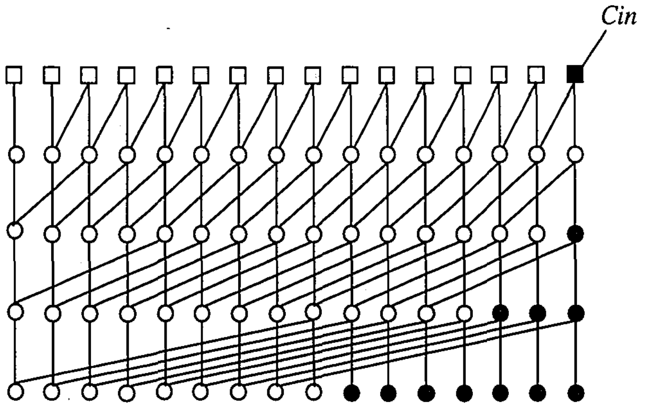


FIGURE 8. Computation of the carry equation using prefix trees for a 16-bit adder with carry input according to [6]. The filled square implements the equation for the first generate signal, the empty squares compute generate and propagate signals, the empty circles implement the fundamental carry operator, the filled circles are buffers.

In CMOS technology a small speedup can be achieved by using transmit signals instead of propagate signals to compute the carries for each bit position. The final sum computation still requires the propagate signals to be generated from the primary inputs. The addition operation in this case is defined as

$$g_j = a_j \cdot b_j$$

$$t_j = a_j + b_j$$

$$c_j = g_j + t_j \cdot c_{j-1}$$

$$s_j = p_j \oplus c_{j-1}$$

where $0 \le j < n$ and c_{-1} is the primary carry-input.

We define $(G_j^j, T_j^j) = (g_j, t_j)$ and

$$(G_i^j, T_i^j) = (g_j, t_j) o (g_{j-1}, t_{j-1}) o \ldots o (g_i, t_i) \text{ if } j > i$$

where "o" is fundamental carry operator. The computation of (G_0^j, T_0^j) follows the same methodology as in step 2 for (G_0^j, P_0^j). At each bit position, the carry is given by

$$c_j = G_0^j + T_0^j \cdot c_{-1}$$

If there is no primary carry input, then c_j is simply G_0^j.

The t_j signals can be computed faster than the p_j signals since an OR gate is typically faster than an XOR gate. Hence, the carry computation through the prefix tree can start slightly earlier if transmit signals are used. Since, the sum generation step still uses the propagate signals, the load on the transmit signals in this architecture is smaller than the load on propagate signals in the earlier architecture. However, the load on the input signals is now higher since both transmit and propagate signals need to be generated.

In Fig.7, the open squares at the top need to compute the transmit signal in addition to the generate and propagate signals. The remaining circles operate on the transmit signals instead of the propagate signals. The circuit diagram is shown in the Appendix [Figure A-4].

The superimposed prefix tree adder has smaller logic depth than the Brent-Kung adder, hence there is less delay through the adder. Although the superimposed prefix tree adder implementation in VLSI requires more hardware than the Brent-Kung adder as shown in Figure 11 (transistor count), the layout area in the regular datapath structure is smaller, due to the decreased number of stages (rows in the layout), with interconnect complexity not being an issue with the use of the multilevel metal interconnect.

2.6 Ling CLA.

A new approach to represent the carry formation and propagation was introduced in [18]. A new *H* function was derived that represents the relationship of neighboring bits, similar to group transmit (represented as *T*) and group generate (represented as *K*) signals in the CLA. The adder can be constructed using the following steps

Step 1 (1 stage)

calculate $k_j = a_j \cdot b_j \quad 0 \le j < n$ and $t_j = a_j + b_j \quad 0 \le j < n$

Step 2 ($\lceil \log_2 n \rceil$ stages)

For k=1 to $\lceil \log_2 n \rceil$ calculate

$$H_0 = k_0 + c_{-1} \cdot t_0$$

$$H_j = K^j_{j-2^{k-1}+1} + T^j_{j-2^{k-1}+1} \cdot H_{j-2^{k-1}} \qquad 2^{k-1} - 1 \le j < 2^k - 1$$

$$(K^j_{j-2^k+1}, T^j_{j-2^k+1}) = (K^j_{j-2^{k-1}+1}, T^j_{j-2^{k-1}+1}) o (K^{j-2^{k-1}}_{j-2^k+1}, T^{j-2^{k-1}}_{j-2^k+1})$$

$$2^k - 1 \le j < n$$

Step 3 (2 stages)

calculate $s_j = H_j \oplus T^j_0 + T^j_0 \cdot K^{j-1}_0 \cdot H_j \quad 1 \le j < n$,

$$s_0 = c_{-1} \cdot K_0 + H_0 \oplus T_0 \qquad \text{and } c_{n-1} = H_n \cdot T_n$$

To reduce the fanout of c_{-1} to 1, c_{-1} is incorporated into the first *H* function. This is illustrated in Figure 8. The filled square now calculates the H_0 and the remaining circles operate on the transmit/kill signals instead of the propagate/generate signals. The final sum computation still requires 2 stages according to step 3. The logic depth of this adder is still $3 + \lceil \log n \rceil$. The proof of algorithm is given in [18]. The circuit implementation of superimposed tree [Section 2.4] Ling adder is shown in the Appendix [Figure A-3].

The significance of the Ling adder compared to other adders is the use of the OR gate for the transmit signal in conjunction with superimposed prefix trees with low order *H* functions used inside the tree for generating the high order *H* function. Also, similar to the adder shown in Figure 8, the primary carry input c_{-1} of the adder has a fanout of one, compared to adder in Figure 7. It requires the same layout area as the superimposed prefix tree adder with smaller propagation delay. This is due to the use of the OR gate for the transmit signal, instead of XOR for the propagate signal, and faster logic for the final sum calculation.

3 New architecture for prefix tree CLA.

3.1 New architecture for prefix tree adder.

An alternative solution to the carry computation shown in Fig. 7 is to allow the low order carries to be used to compute the high order carries in parallel inside the prefix tree. For example, c_0 can be used in stage 2 and further stages of the prefix tree without affecting the delay of the carry at the higher bit positions. This algorithm is described below for a radix-2 prefix tree.

Step 1 (1 stage)

calculate $g_j = a_j \cdot b_j$ and $t_j = a_j + b_j \quad 0 \le j < n$

Step 2 ($\lceil \log_2 n \rceil$stages)

For k=1 to $\lceil \log_2 n \rceil$ calculate

$$c_j = G^j_{j-2^{k-1}+1} + T^j_{j-2^{k-1}+1} \cdot c_{j-2^{k-1}} \qquad 2^{k-1} - 1 \le j < 2^k - 1$$

$$(G^j_{j-2^k+1}, T^j_{j-2^k+1}) = (G^j_{j-2^{k-1}+1}, T^j_{j-2^{k-1}+1}) o (G^{j-2^{k-1}}_{j-2^k+1}, T^{j-2^{k-1}}_{j-2^k+1})$$

$2^k - 1 \le j < n$

Step 3 (1 stage)

calculate $c_{n-1} = G_0^{n-1} + T_0^{n-1} \cdot c_{-1}$ and $s_j = p_j \oplus c_{j-1} \quad \forall j, 0 \le j < n$.

This is illustrated in Fig. 9 for a 16-bit adder. The open squares at the top compute t_j and g_j for each bit position according to step 1. The crossed circles implement the first equation in step 2 and the first equation in step 3. The empty circles apply the fundamental carry operator according to the second equation in step 2, and the filled circles are buffers. An additional stage (not shown) is needed to generate the sum at each bit position from p_j and c_{j-1} according to step 3. The sum computation occurs in parallel with the computation of the final carry output c_{n-1}. The logic depth of this adder is $2 + \lceil \log n \rceil$. The fanout of c_{-1} is $1 + \lceil \log n \rceil$. This algorithm can also be extended to higher radix prefix trees. The circuit implementation is shown in the Appendix [Figure A-5].

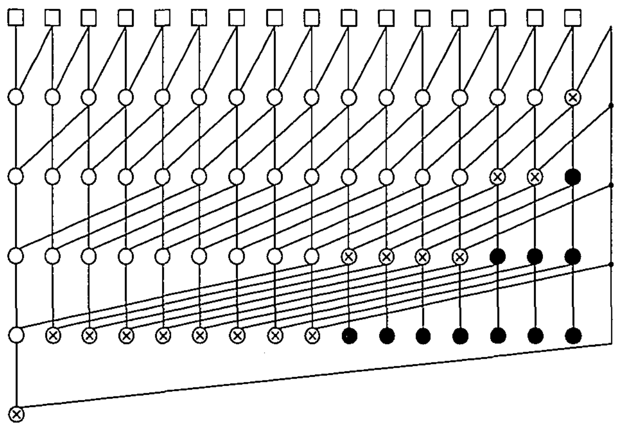


FIGURE 9. Prefix tree adder with carry incorporated into the tree. The empty circles implement the fundamental carry operator, the filled circles are buffers, the crossed circles compute carries according to Step 2 and 3, the empty squares compute generate and propagate signals.

3.2 Improved architecture of prefix tree adder for low power.

With shrinking technology, the interconnect capacitance becomes a major factor in the loading per node. Especially, the routing in the datapath used to connect the high and low bits of the structures. In the adder structure of Fig. 9, the previous stage (low order) carries are used to produce the high order carries. The interconnect lines to the last stage of carry generation need to run a distance of *n/2* bits, where *n* is the adder size. Also, to generate the carry out (c_{n-1}), the primary carry in (c_{-1}) needs to run from bit position 0 to *n-1*.

The carry out (c_{n-1}) in Fig. 9 with n=16 uses EQ. 1 below, requiring an extra column of carry operators for calculating (G_0^n, T_0^n).

$$C_{n-1} = G_0^{15} + T_0^{15} \cdot C_{-1} \qquad \text{(EQ 1)}$$

Instead, the new design implements the EQ. 2, thus, eliminating some one the long interconnecting wires, which compensates for the increase of the load on c_{14}.

$$C_{out} = G_{15} + T_{15} \cdot C_{14} \quad \text{(EQ 2)}$$

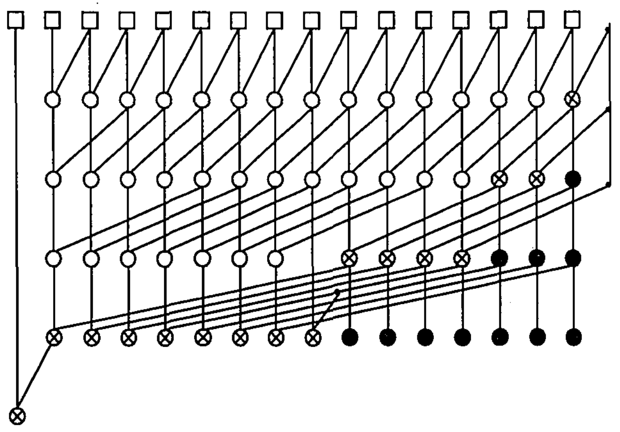


FIGURE 10. Prefix tree adder with carry input incorporated into the tree and with the low power solution with the same delay as the carry tree shown in Figure 9.

The same idea is applied to the intermediate carry out in bit position '7', which is c_7, shown in EQ.3 and EQ.4. c_7 is an intermediate carry, which was generated by (G_0^7, T_0^7) and c_{-1} EQ.3. Eliminating the cell which generated (G_0^7, T_0^7) and using (G_4^7, T_4^7) and c_3 to generate c_7 reduces the total number of gates needed to implement the above architecture. Since the fanout of the cell which generates (G_3^7, T_3^7) is reduced from 2 to 1, and the fanout of c_3 is increased from 1 to 2. This balances the overall delay for c_3 generation.

$$C_7 = G_0^7 + T_0^7 \cdot C_{-1} \quad \text{(EQ 3)}$$

$$C_7 = G_4^7 + T_4^7 \cdot C_3 \quad \text{(EQ 4)}$$

This idea can be generalized for an n-bit prefix-tree adder with the primary carry input incorporated into the tree. The final carry out is generated as:

$$C_{n-1} = G_0^{n-1} + T_0^{n-1} \cdot C_{n-2} \qquad \text{(EQ 5)}$$

The first (from the right) intermediate carry for every carry generation stage starting with c_7, as shown in Figure 9, is generated as follows:

$$C_{j-1} = G_{j/2}^{j-1} + T_{j/2}^{j-1} \cdot C_{(j/2)-1} \qquad \text{(EQ 6)}$$

, where j = 8, 16, 32,....

Table 4 compares the 32-bit prefix tree adder according to Fig. 9 with the new and improved low power version of the same adder according to Fig. 10. This shows that for the same delay in both designs, the number of transistors used is improved by 3.3%. The circuit implementation is shown in the Appendix [Figure A-6].

The power numbers improve for two main reasons: 1) A full bit slice for c_{n-1} generation and an additional cell in each row, shown in Figure 10, represented as $c_{\frac{j}{2}-1}$ with *j=8, 16, 32...* were eliminated. This reduces the amount of hardware required for the adder implementation; 2) The fanout of the primary carry input signal was reduced to 3.

4 Prefix tree CLA architecture comparison.

4.1 Architecture comparison.

Table 2 compares different prefix tree adder architectures with primary carry input. The architecture shown in Figures 9 and 10 have the smallest logic depth and intermediate amounts of fanout on the c_{-1} input. The wiring complexity is manageable in 0.25um and smaller CMOS technologies, which have several levels of interconnect.

Table 2: Comparison of different prefix tree adder architectures.

Figure	*G, P, T* fanout	c_{-1} fanout	logic depth	wiring
1a [6]	n/2	1	$3 + \lceil \log n \rceil$	low
1b [6]	2	1	$3 + \lceil \log n \rceil$	high
2 [6]	2	1	$3 + \lceil \log n \rceil$	med
Fig. 7	2	n	$3 + \lceil \log n \rceil$	high
Fig. 9	2	$1 + \lceil \log n \rceil$	$2 + \lceil \log n \rceil$	high
Fig.10	2	3	$2 + \lceil \log n \rceil$	high

4.2 VLSI Implementation.

32-bit versions of the following adders were implemented in the Lucent 0.25-um. CMOS process: the Brent-Kung adder, carry-skip adder, Ling adder, adders from Figures 6 and 7, Figure 9, and 10. All designs use fully static circuits. Table 3 summarizes the characteristics of the Lucent 0.25-um. CMOS process.

Table 3: Lucent 0.25-um. CMOS process parametrics.

	NMOS	PMOS
Tox	50 A	50 A
Lpoly	0.24um.	0.28um.
Vth	0.55V	0.85V
Ion	570uA/um.	230uA/um.
M1 pitch	0.84	0.84um.
M2 pitch	0.88	0.88um.
M3 pitch	0.88	0.88um.

The appendix [Figure A-7] shows the layout of a 32-bit prefix tree adder according to the architecture shown in Fig. 7. Appendix [Figure A-8] shows the layout of a 32-bit prefix tree adder according to the architecture shown in Fig. 9.

4.3 Test results.

The adders have been implemented on a 0.25-um. CMOS test chip and hooked up as ring oscillators that exercise their critical paths. The critical paths were identified from Pathmill simulations. Pathmill is a static timing analysis tool from Synopsys Inc. The output wave form frequency of the ring oscillators is divided by 2^{12} for observation off-chip. Additional test structures on the chip are used to determine the delay of the control circuitry in the ring oscillator path, as described in Section 4.4. The delay of these control circuits is subtracted from the adder delay measured and reported results are given in Table 4. The power numbers reported in Table 4 are based on Powermill simulations (Powermill is a power analysis tool from Synopsys Inc.)

Table 4: Comparison with other 32-bit adders.

Reference	Delay, *ns*	Power, *mW*	Area, um^2	Vdd, V	Technology	Year
Fig. 7	1.1	32@400MHz	0.04	2.5	0.25um. CMOS	1999
Fig. 9	1.0	32@400MHz	0.03	2.5	0.25um. CMOS	1999
Fig. 10	1.0	30@400MHz	0.03	2.5	0.25um. CMOS	1999
Ref. [14]	1.27	114@580MHz	0.3	0.9	0.6um. GaAs	1997
Ref. [15]	2.7		0.71	5.0	1.2um. EMODL	1997
Ref. [16]	3.1		0.28	5.0	0.9um. MODL	1989
Ref. [17]	2.1	900@(?)MHz	27.84	4.5	3.5um. ECL	1988

Table 4 shows the speed of the adder shown in Figures 7,9 and 10 in comparison with other published work. The speed was achieved with static CMOS circuits. Static circuits are preferred to dynamic circuits because of their ease of design. In addition, static circuits consume less power because they do not need clocks to precharge internal nodes. As shown in Table 4, the area of the adders in Figures 9 and 10 are the smallest reported so far. Figure 11 shows the trade-off in speeds of 32-bit adders vs. transistor count in 0.25-um.

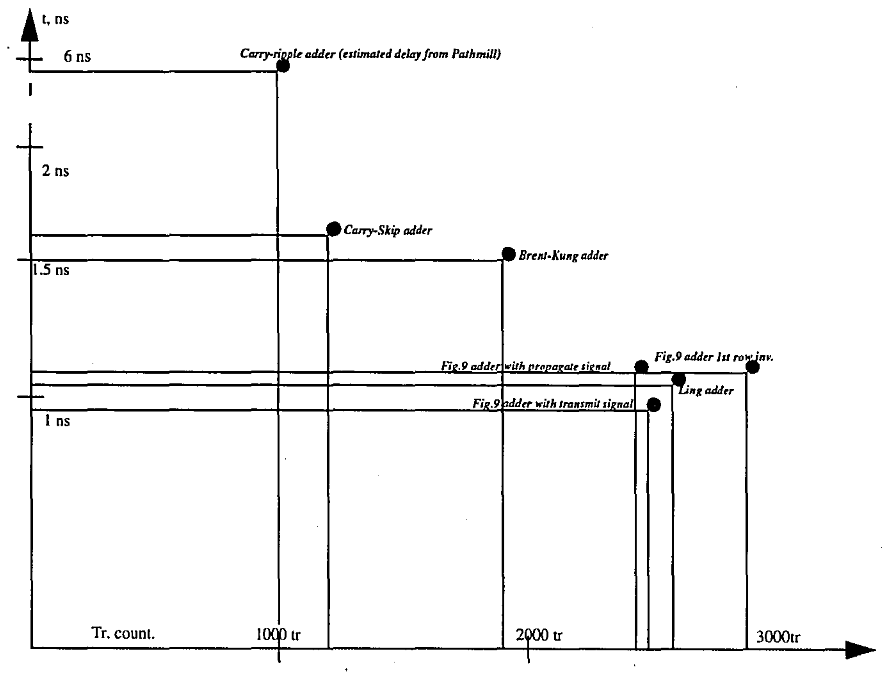


FIGURE 11. Speed vs. transistor count of different 32-bit adders in 0.25-um.

For the adder architecture in Figure 7, since it was implemented first on a test shuttle, three different gates styles were implemented. The first of these used and-or-invert (AOI) and or-and-invert (OAI) gates to implement the fundamental carry operator at alternate stages of the prefix trees. The second implementation used and-or (AO) gates at each stage, and hence had better buffering to drive the interconnect wires. The third implementation used OAI and AOI gates at the first two stages of the prefix tree and AO gates for the remaining stages. This was done to measure the effect of buffering and drive only on the stages with large interconnect.

Table 5 summarizes the measured delays of the adders based on the architecture in Fig. 7. The fastest implementation is the one with AOI/OIA gates at alternate stages of the prefix trees and with reduced interconnect coupling. Each adder has been implemented with two different wiring schemes. An implementation with horizontal metal3 and vertical metal2 is area optimal and an implementation with vertical metal3 and horizontal metal2 is delay optimal.

Table 5: Performance of the different implementations of the adder in Figure 7.

Type implemented	typical delay, *ns*	worst case delay, *ns*	Area, nm^2	Vdd, V	Mt3 direction
AOI/OAI @ alternate stages	1.00	1.08	0.035	2.5	horizontal
AO @ every stage	1.16	1.26	0.035	2.5	horizontal
OAI/AOI @ first two stages AO @ other stages	1.06	1.14	0.035	2.5	horizontal
AOI/OAI @ alternate stages	0.97	1.05	0.047	2.5	vertical
AO @ every stage	1.12	1.22	0.047	2.5	vertical
OAI/AOI @ first two stages AO @ other stages	1.02	1.11	0.047	2.5	vertical

The appendix [Figure A-10] shows two layout styles of the adders discussed above one with MT3 horizontal and MT2 vertical and the other one with MT3 vertical and MT2 horizontal. The appendix [Figure A-11] shows the photograph of the die on which the described adders were implemented.

4.4 Test structure.

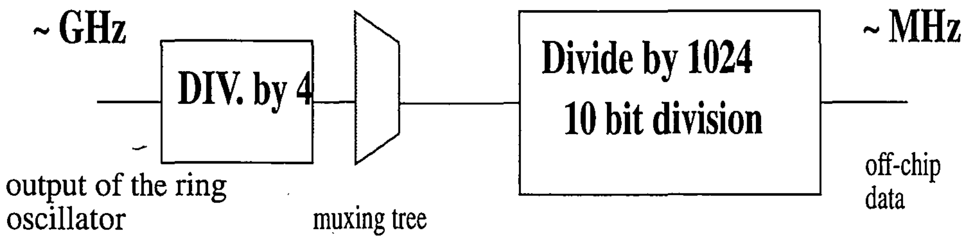


FIGURE 12. Divide network for testing the speed of adders off-chip.

To test the speed of the ring oscillators on the test chip, a mux-tree network was build with separate enables to measure the speed of one ring oscillator at a time. Since the output frequency of the ring oscillators are in the GHz range, which is hard to measure on a scope, a division by 1024 (10 bits) is performed to slow down the output signal. Also, to allow more time for the muxes in the mux-tree to switch, a divide-by-4 network is used before the mux-tree for each oscillator circuit. This is shown in Fig. 12.

The same enable signals were used to allow for isolation of each oscillator circuit via an AND gate (or a NAND gate depending on the output-to-input relationship), where the output of the oscillator is feed back to the input of the AND gate. To measure the delay of the AND (NAND) gate, 5 different ring oscillators were built: a chain of 7 inverters, a chain of 11 inverters, a chain of 15 inverters, a chain of 19 inverters, and a chain of 23 inverters of the same size gates connected via an AND gate, enabled by the same enable signals as the adder's ring oscillators. To calculate the delay of the AND gate, the delays of these 5 oscillators are measured as follows:

T(delay7) = T(AND) + T (7 inv.)

T(delay11) = T(AND) + T (11 inv.)

T(delay15) = T(AND) + T (15 inv.)

T(delay19) = T(AND) + T (19 inv.)

T(delay23) = T(AND) + T (23 inv.)

The zero point on the time plot shows the delay for an AND gate. The results of this experiment are shown in Fig. 13. According to Figure 13, the average measured delay for an AND gate is 0.38 nsec. The circuit diagram for the test structure is shown in the Appendix [Figure A-9].

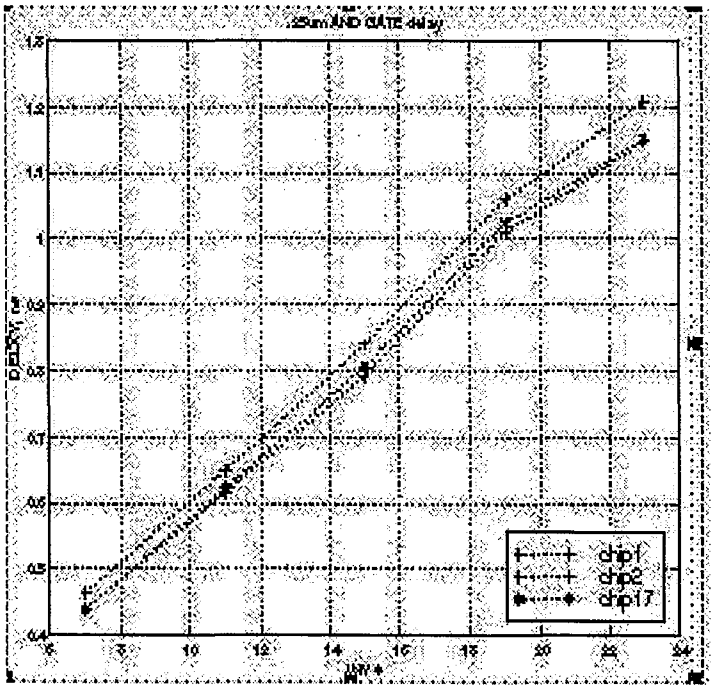


FIGURE 13. AND gate delay measurement results.

5 Conclusions and Future Research.

The prefix tree adders were implemented based on architectures that use a separate prefix tree for each bit position. The architecture was described that incorporates the contribution due to the primary carry input into the prefix trees without any additional overall delay. Measured results from the test chip fabricated in the Lucent 0.25um. CMOS Technology using fully static circuits verify adder operation at 1.0 ns or 1 GHz.

The new architecture presented in this work for 32-bit adders can be extended to 64-bit adders or other word sizes. Also, the proposed adder could be compared with Tyagi adders [19] and carry-select adders [20]. Furthermore, the architecture of the superimposed prefix tree adder can be incorporated into other applications, such as comparators, arithmetic and logic units (ALU and DAU (Data Arithmetic Unit)), and multiply-add units. Examples of such applications for a prefix tree adder with carry-in incorporated into the tree are shown in Figure 14.

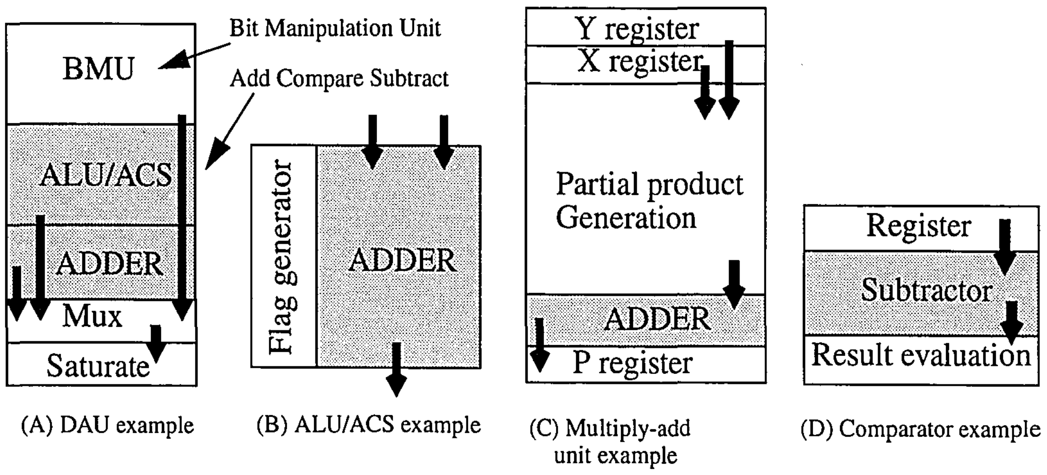


FIGURE 14. Examples of possible applications for a prefix tree adder with carry-in incorporated into the tree. Filled rectangles indicate the place, where the adder/ subtractor based on the new architecture would be used.

References

[1] A. Weiberger and L. J. Smith, "A one-microsecond adder using one-megacycle circuitry," *IEEE Transaction on Electronic Computers*, pp. 65-73 June 1956

[2] T.-F. Ngai, M. J. Irwin, and S. Rawat, "Regular, area-time efficient carry-lookahead adders," *Journal of Parallel and Distributed Computing*, vol. 3, pp. 92-105, 1986.

[3] P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transaction on Computers*, vol. C-22 pp. 786-793, Aug. 1973.

[4] R.E. Ladner and M. J. Fisher, "Parallel prefix computation," *Journal of the ACM*, vol. 27, pp. 831-838, Oct. 1980.

[5] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Transaction on Computers*, vol. C-31, pp. 260-264, Mar. 1982

[6] D. Dozza, M Gaddoni, and G. Baccarani, "A 3.5ns, 64 bit, carry-lookahead adder," in *Proceedings of the International Symposium on Circuits and Systems*, pp. 297-300, 1996.

[7] Synopsys, *Module Compiler User's Guide*, Feb. 1998.

[8] T. K. Callaway and E. E. Swartzlander, "Low power arithmetic components," in *Low Power Design Methodologies, Jan M. Rabaey and Massoud Pedram,* eds., pp. 161-200, Kluwer Academic Publishers, 1996.

[9] W.Liu, C. T. Gray, D. Fan, W. J. Farlow, T. A. Hughes and R. K. Cavin, "A 250-MHz wave pipelined adder in 2-um CMOS," *IEEE Journal of Solid State Circuits*, vol. 29, pp. 1117-1128, Sept. 1994.

[10] T. Lynch and E. E. Swartzlander, "A spanning tree carry lookahead adder," *IEEE Transaction on Computers*, pp.931-939, Aug. 1992.

[11] P. Singer, "Tantalum, copper and damascene: The future of interconnects," *Semiconductor International*, pp. 90-98, June 1998.

[12] J. Silberman, N. Aoki, D. Boerstler, J. Burns, S. Dhong, A. Essbaum, U. Ghoshal, D. Heidel, P. Hofstee, K. Lee, D. Meltzer, H. Ngo, K. Nowka, S. Posluszny, O. Takahashi, I. Vo, and B. Zozic, "A 1.0 GHz singe-issue 64b PowerPC integer processor," in *IEEE International Solid-State Circuits Conference*, pp. 230-231, Feb. 1998.

[13] I. C. Kizilyalli, R. Huang, D. Hwang, H. Vaidya, B. Kane, R. Ashton, S. Kuehne, X. Deng, M. Twiford, D. Shuttleworth, E. Martin, X. Li, and M. J. Thoma, "A merged 2.5V and 3.3V 0.25-um CMOS technology for ASICs," in *Proceedings out IEEE CICC*, pp. 159-162, May 1998.

[14] A. Beaumont-Smith and N. Burgess, "A GaAs 32-bit adder," in *IEEE 13th Symposium on Computer Arithmetic*, pp. 10-17, July 1997.

[15] Z. Wang, G. A. Jullien, W. C. Miller, J. Wang, and S. S. Bizzan, "Fast adders using enhanced multiple-output domino logic," *IEEE Journal Solid State Circuits,* vol. 32, pp.206-214, Feb. 1997.

[16] I. S. Hwang and A. L. Fisher, "Ultrafast compact 32-bit CMOS adder in multiple-output domino logic," *IEEE Journal of Solid State Circuits*, vol. 24, pp.358-369, Apr. 1989.

[17] G. Bewick, P. Song, G. D. Micheli, and M. J. Flynn, "Approaching a nanosecond: A 32-bit adder," in *IEEE International Conference on Computer Design*, pp. 221-226, Oct. 1988.

[18] H. Ling, “High-Speed Binary Adder”, in *IBM Journal on Res. Development*, vol. 25, no.3, pp. 156-166, May 1981

[19] Akhilesh Tyagi, “A Reduced-Area Scheme for Carry-Select Adder”, *IEEE Transactions on Computers*, vol. 42, pp.1163-1170, Oct. 1993.

[20] O. J. Dedrij "Carry-select adder," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 340-346, 1962

Appendix

?

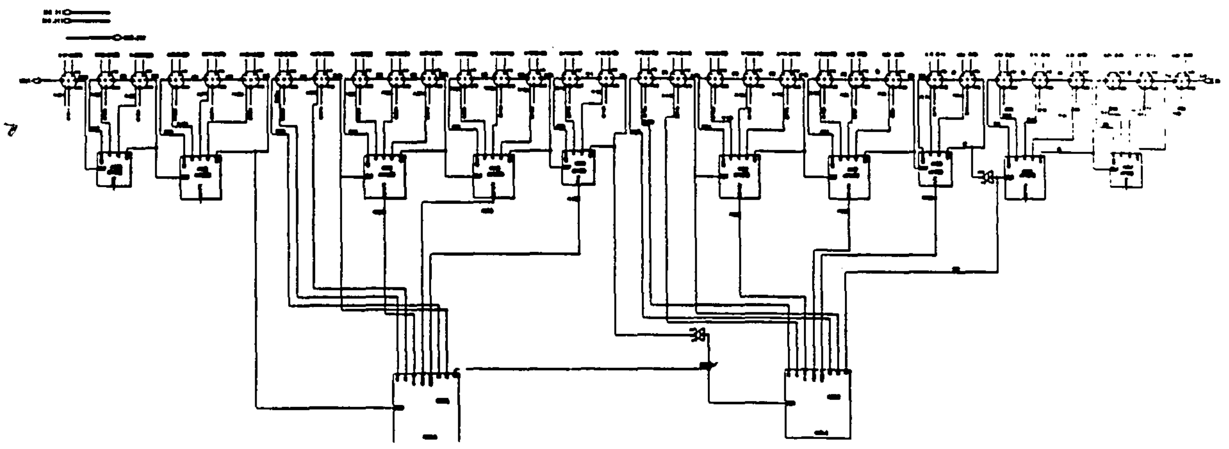


Figure A-1. **32-bit circuit implementation of the carry-skip adder.**

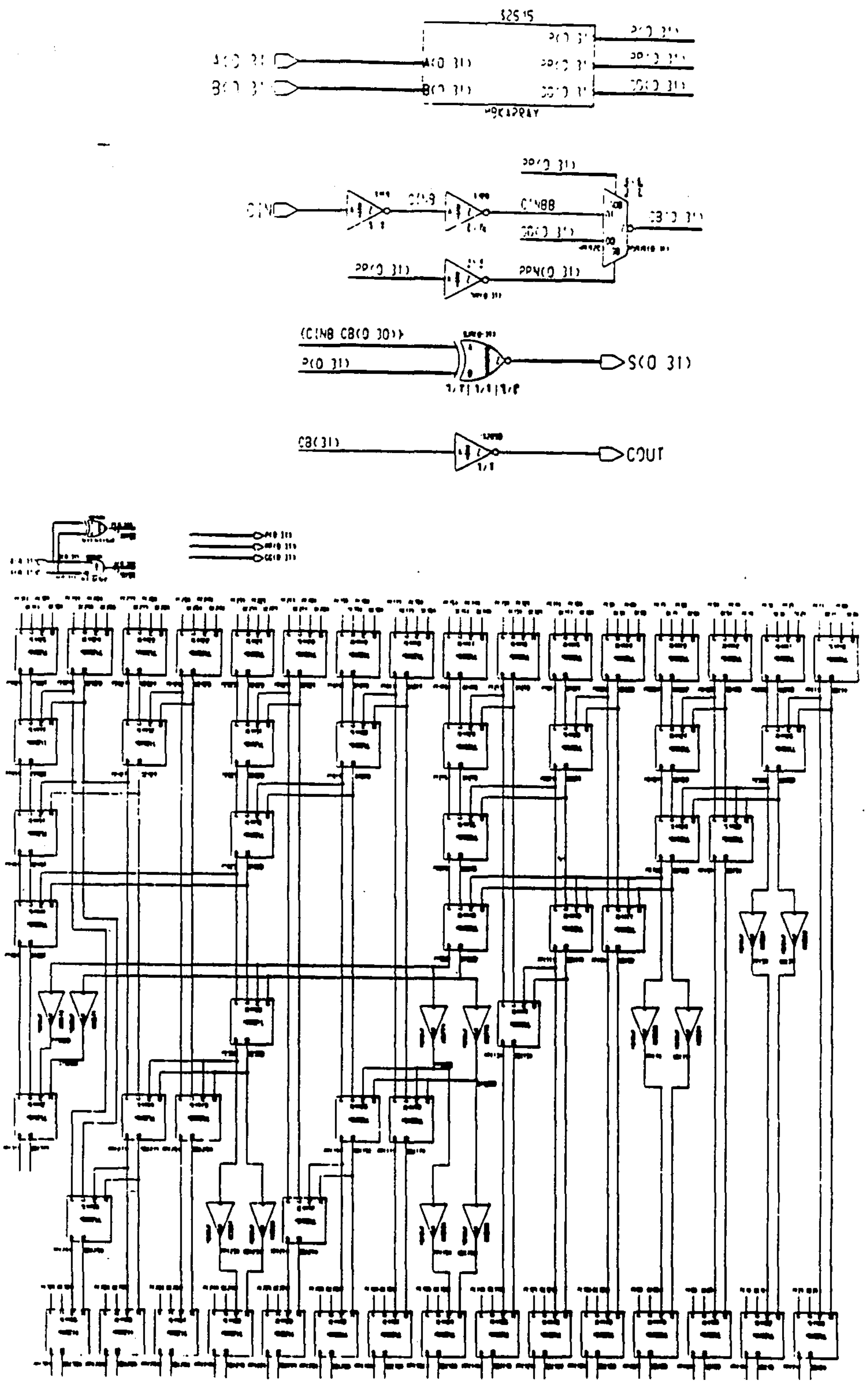


Figure A-2. 32-bit circuit implementation of the Brent-Kung adder.

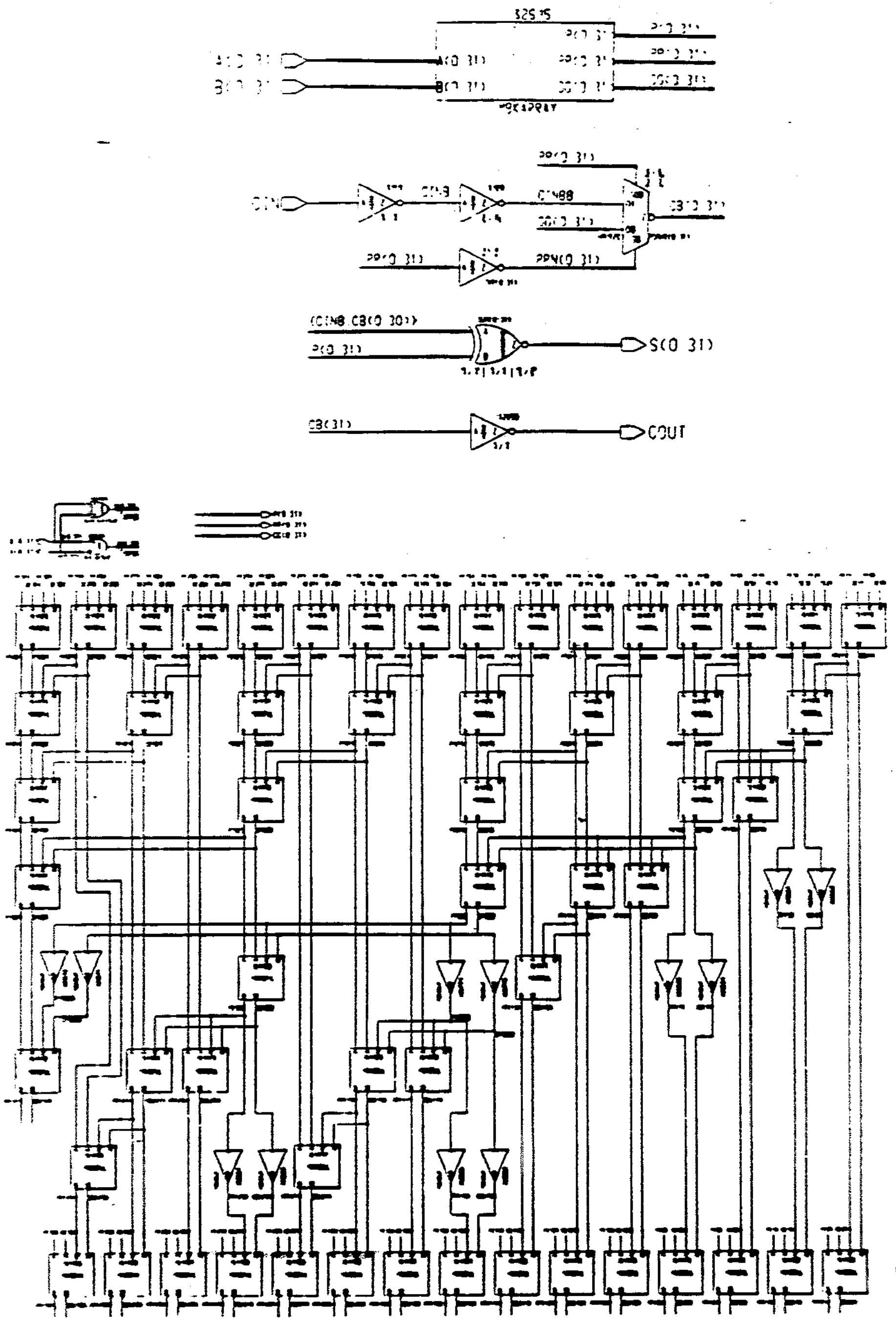


Figure A-2. 32-bit circuit implementation of the Brent-Kung adder.

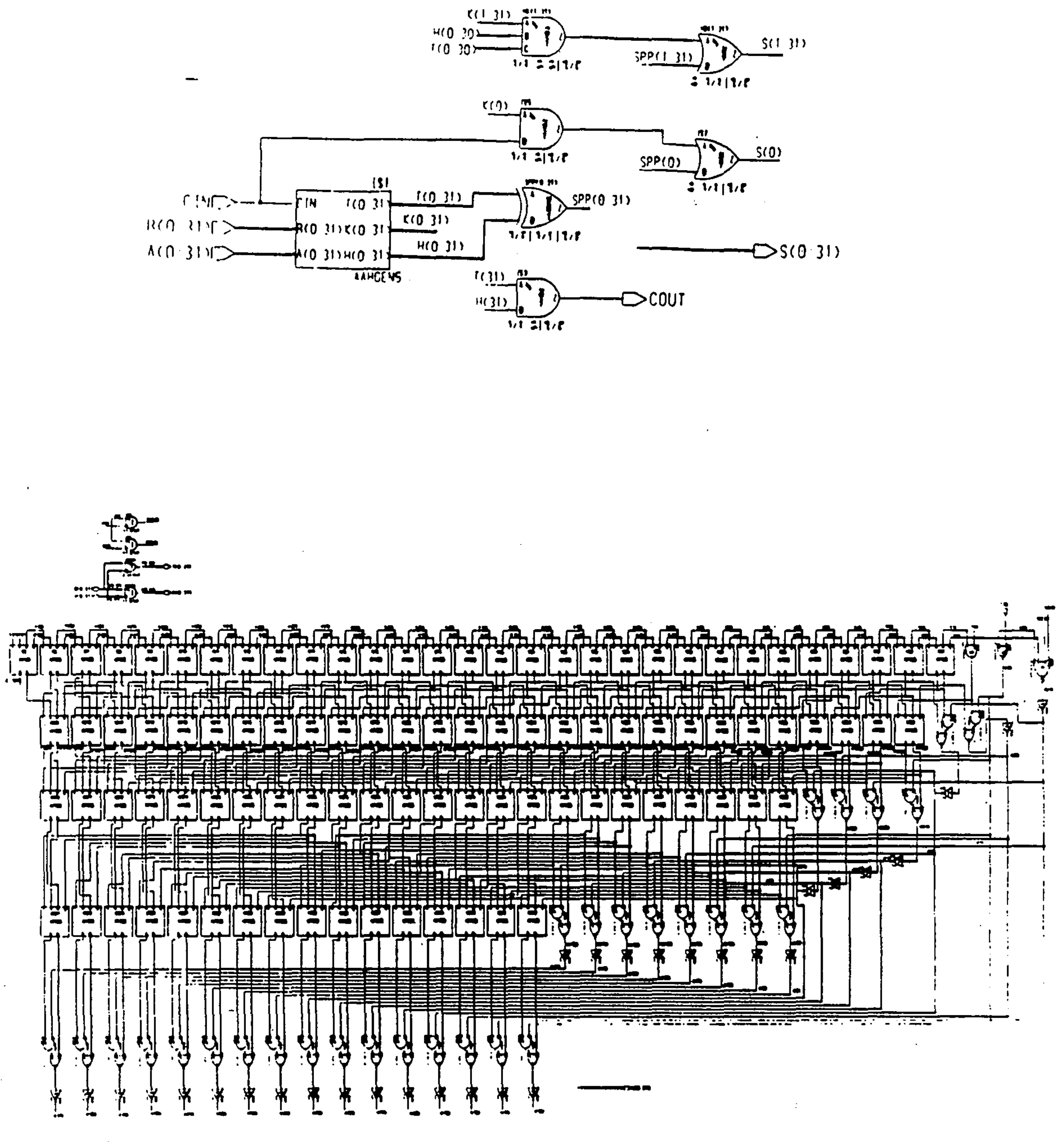


Figure A-3. 32-bit circuit implementation of the Ling adder.

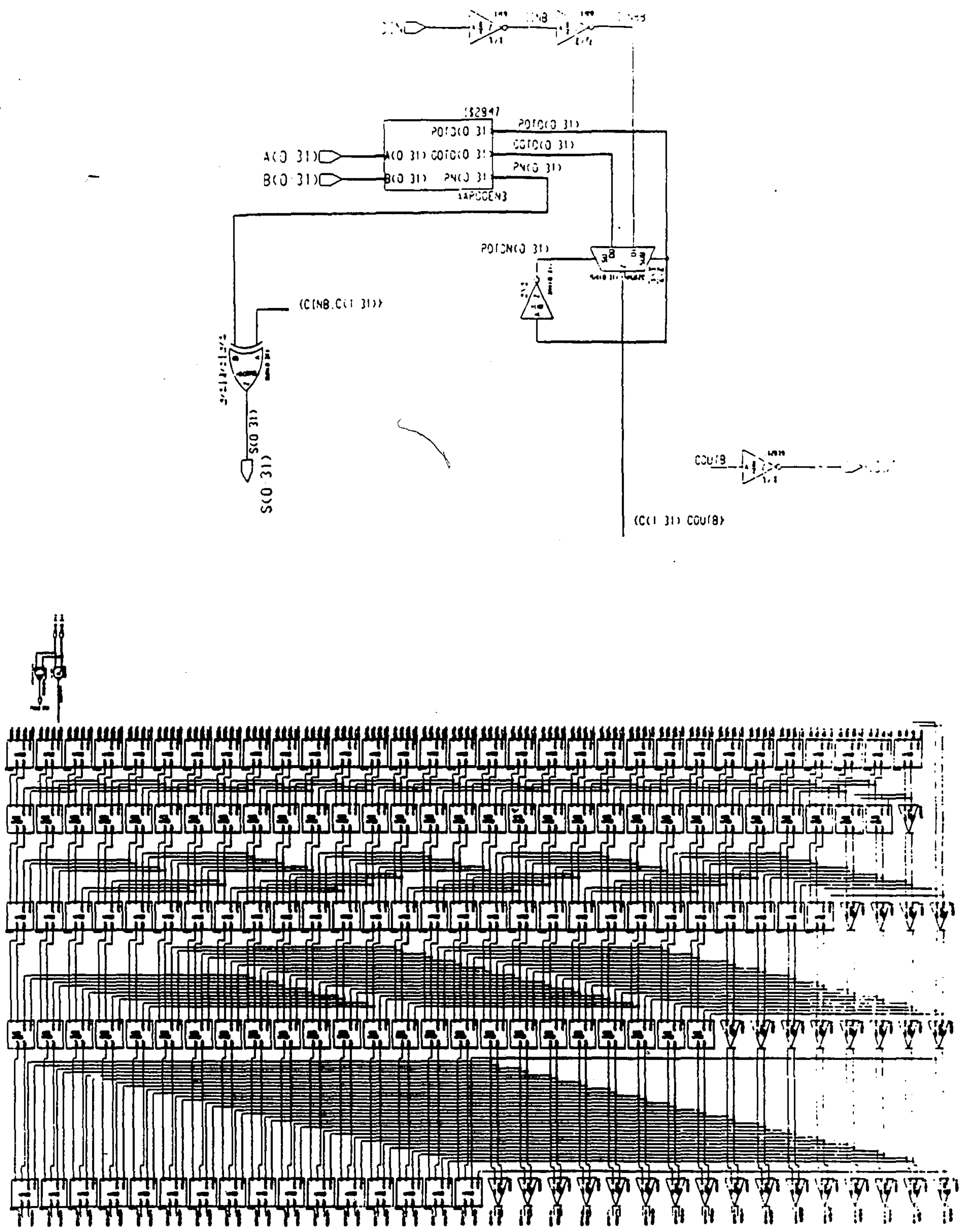


Figure A-4. 32-bit circuit implementation of the superimposed tree CLA, according to Fig. 7.

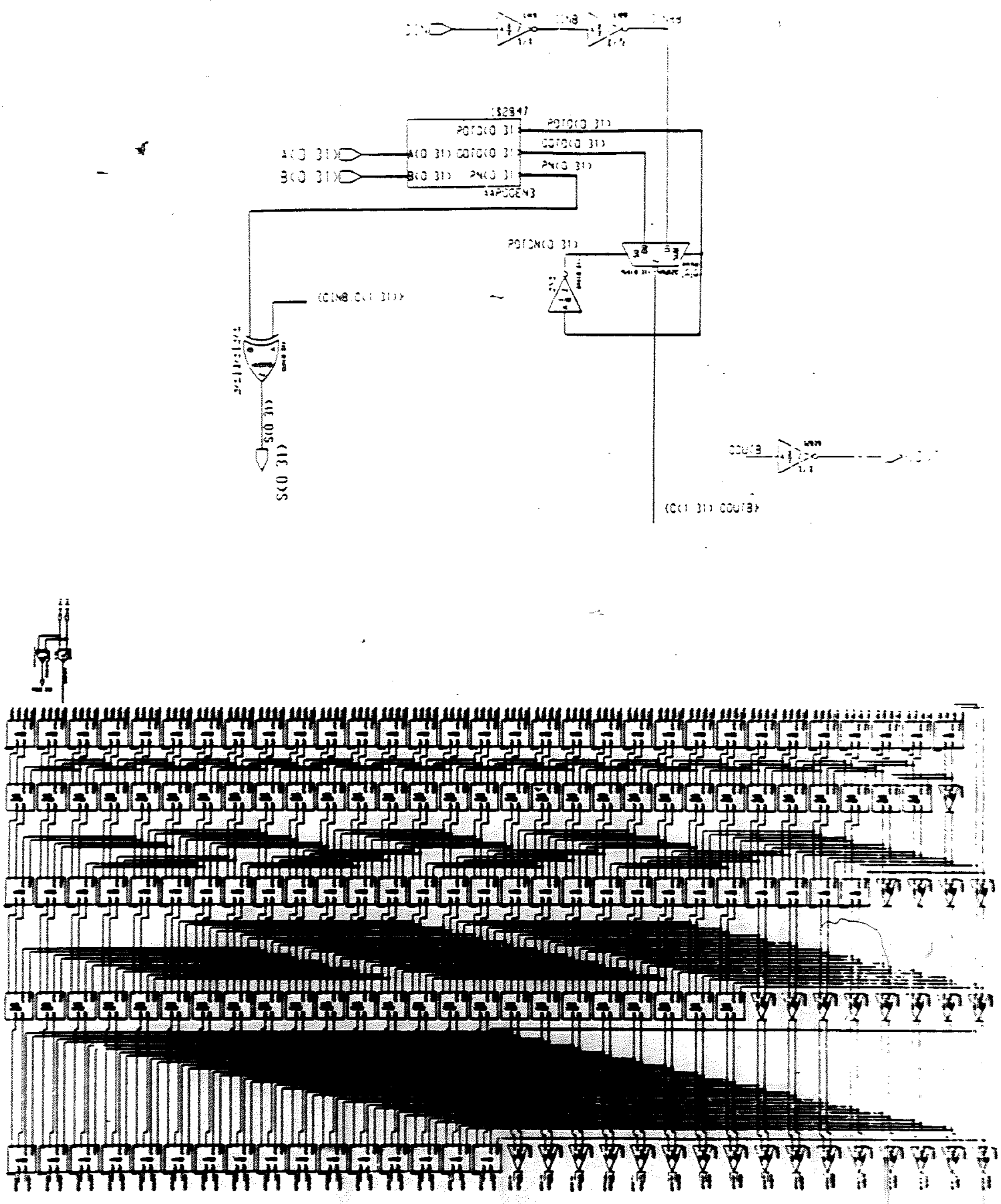
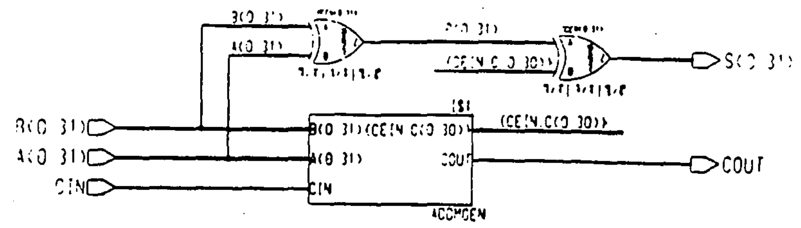


Figure A-4. 32-bit circuit implementation of the superimposed tree CLA, according to Fig. 7.



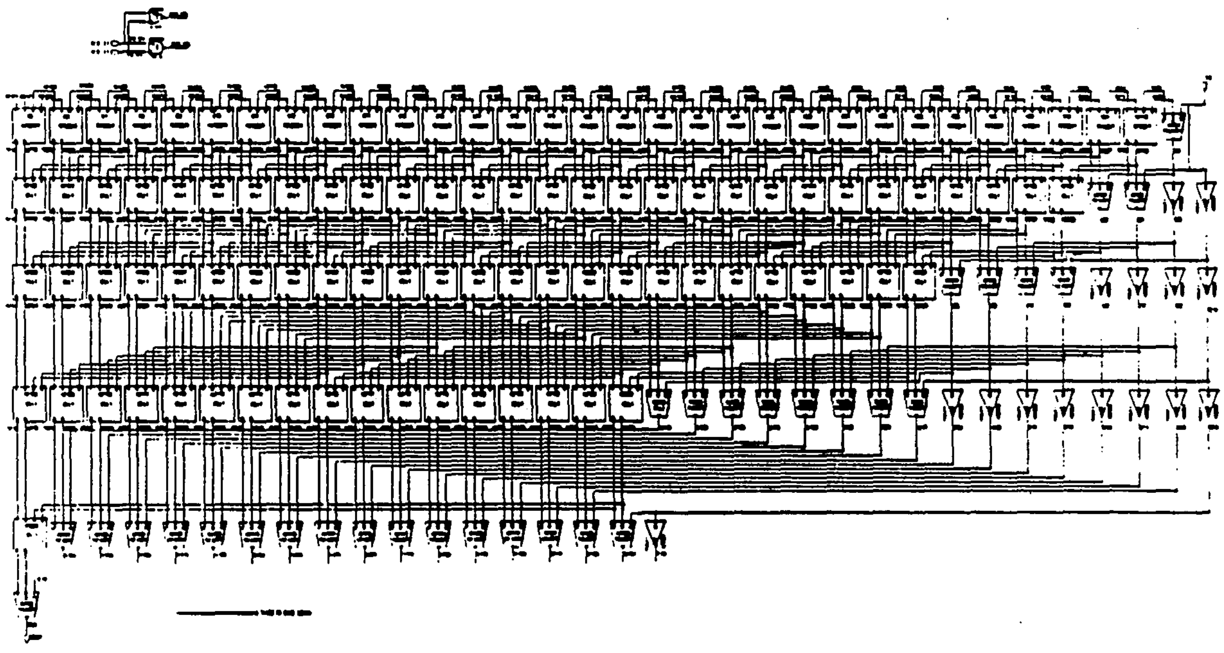
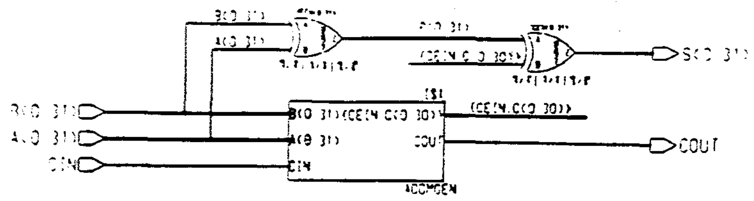


Figure A-5. 32-bit circuit implementation of the superimposed tree CLA adder with carry in incorporated into the tree, according to Fig. 9.



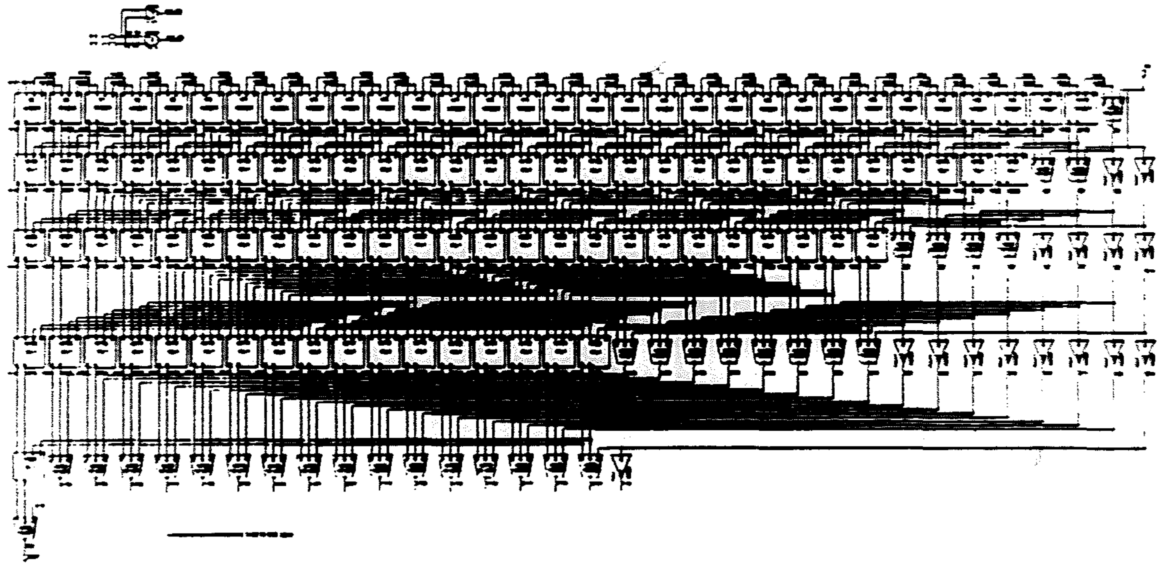


Figure A-5. **32-bit circuit implementation of the superimposed tree CLA adder with carry in incorporated into the tree, according to Fig. 9.**

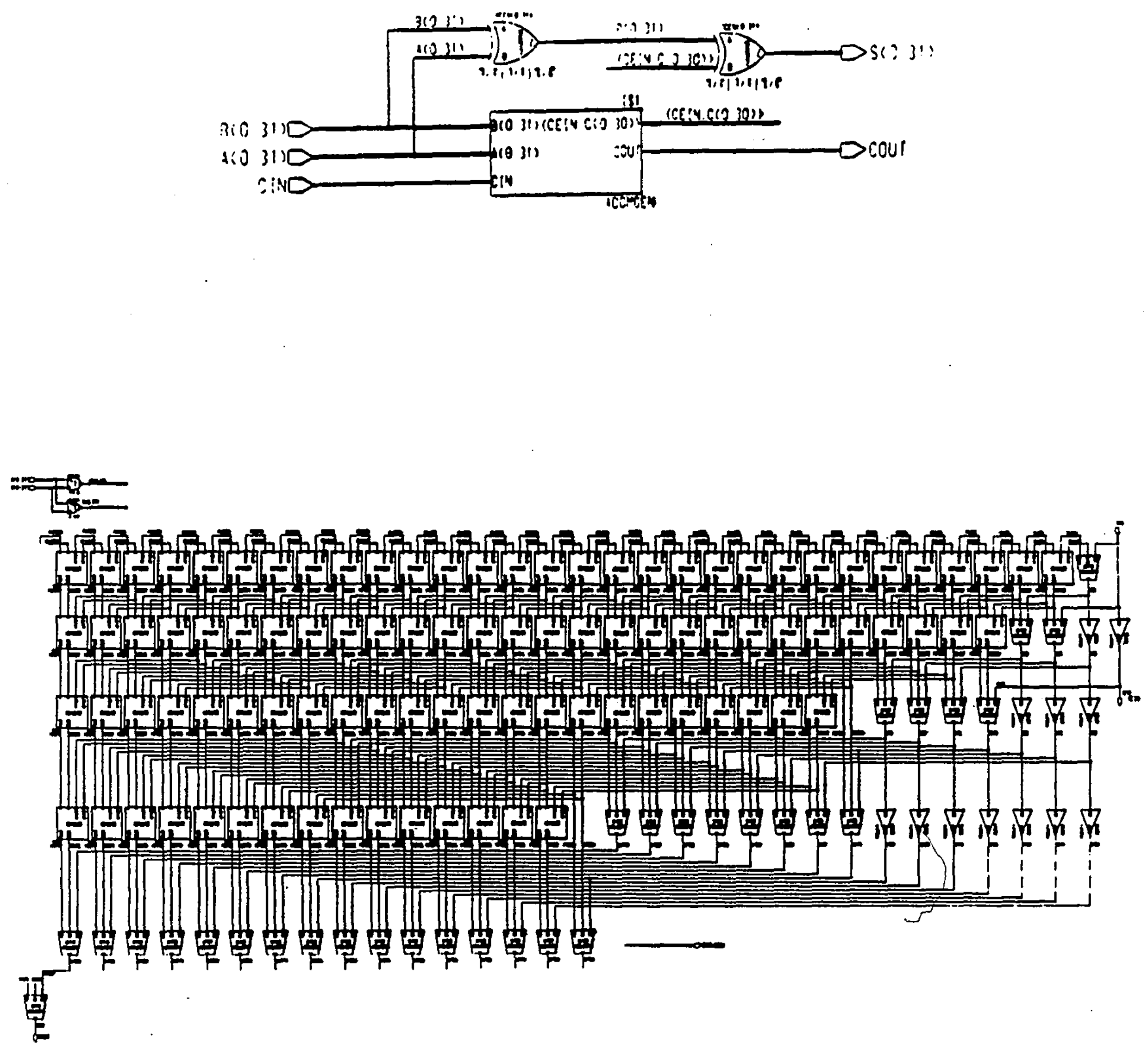


Figure A-6. 32-bit circuit implementation of the Superimposed tree CLA with carry in incorporated into the tree (low power solution), according to Fig. 10.

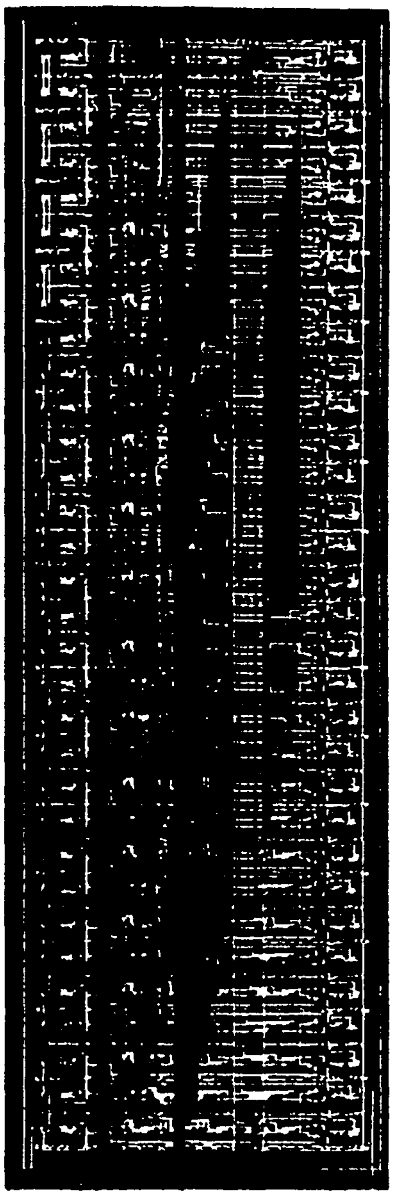


Figure A-7. Layout of the 32 - bit prefix tree adder according to Fig.7.

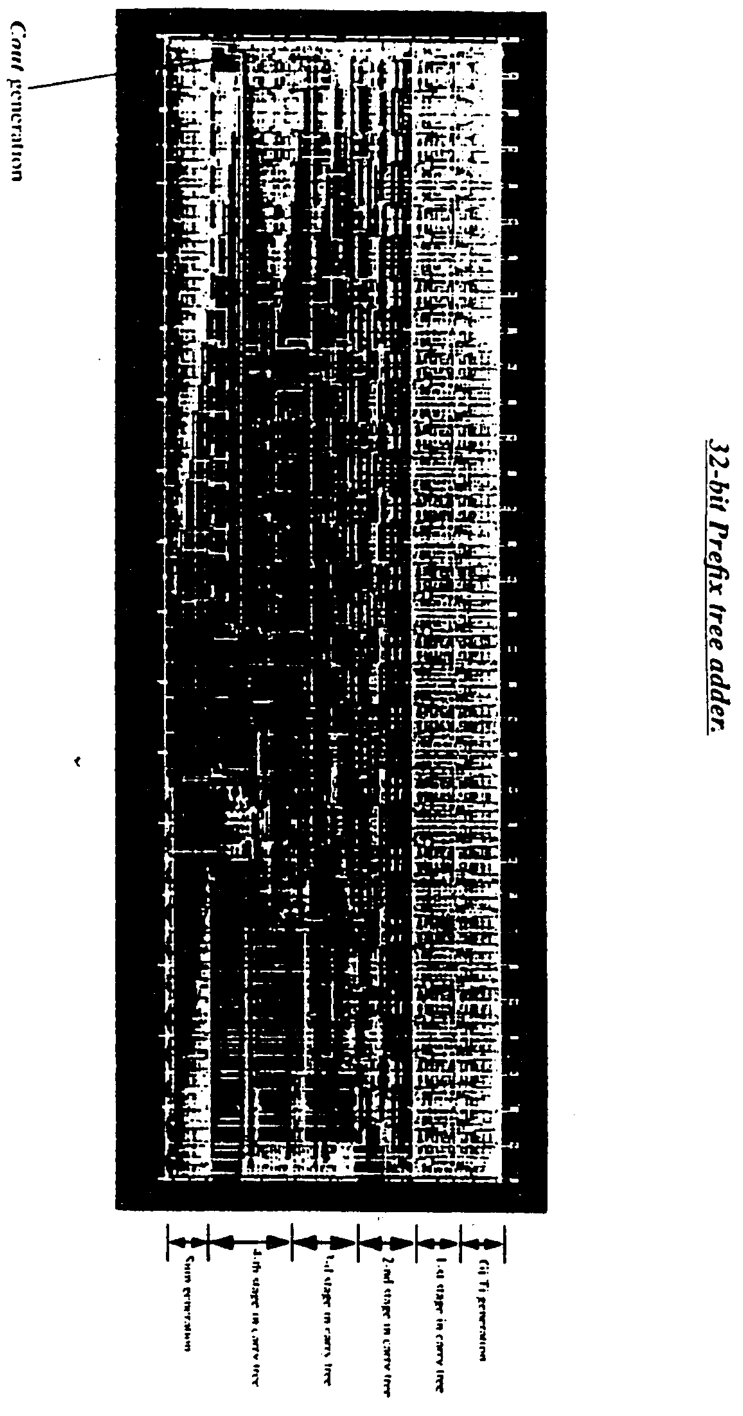


Figure A-8. Layout of the 32 - bit prefix tree adder according to Fig.9.

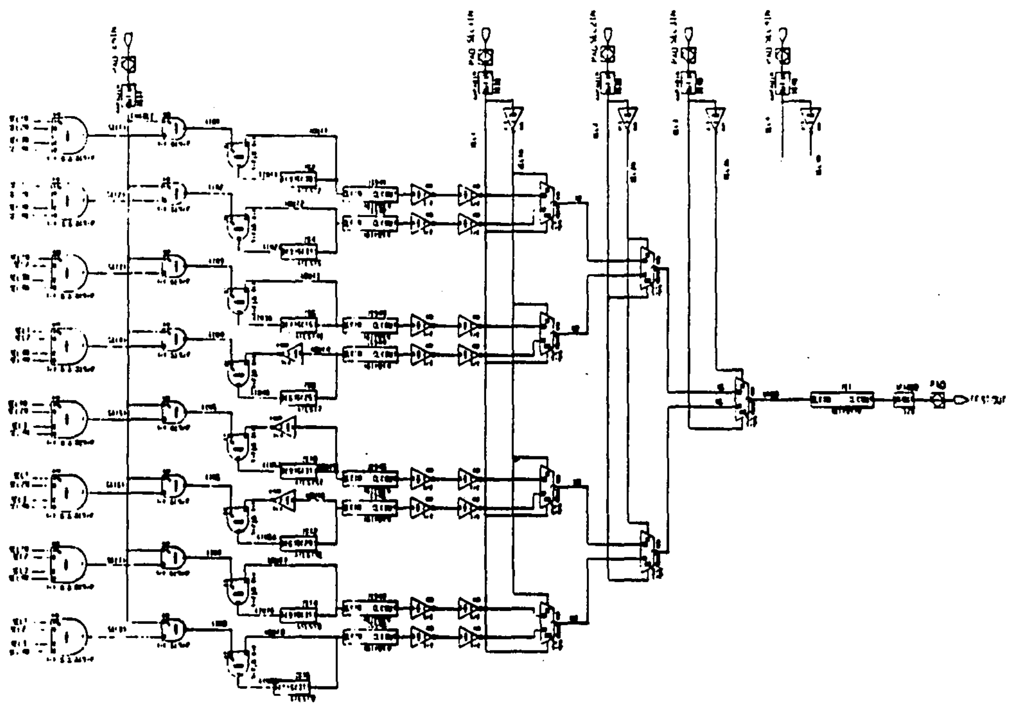


Figure A-9. Circuit diagram for the test structure on the shuttle.

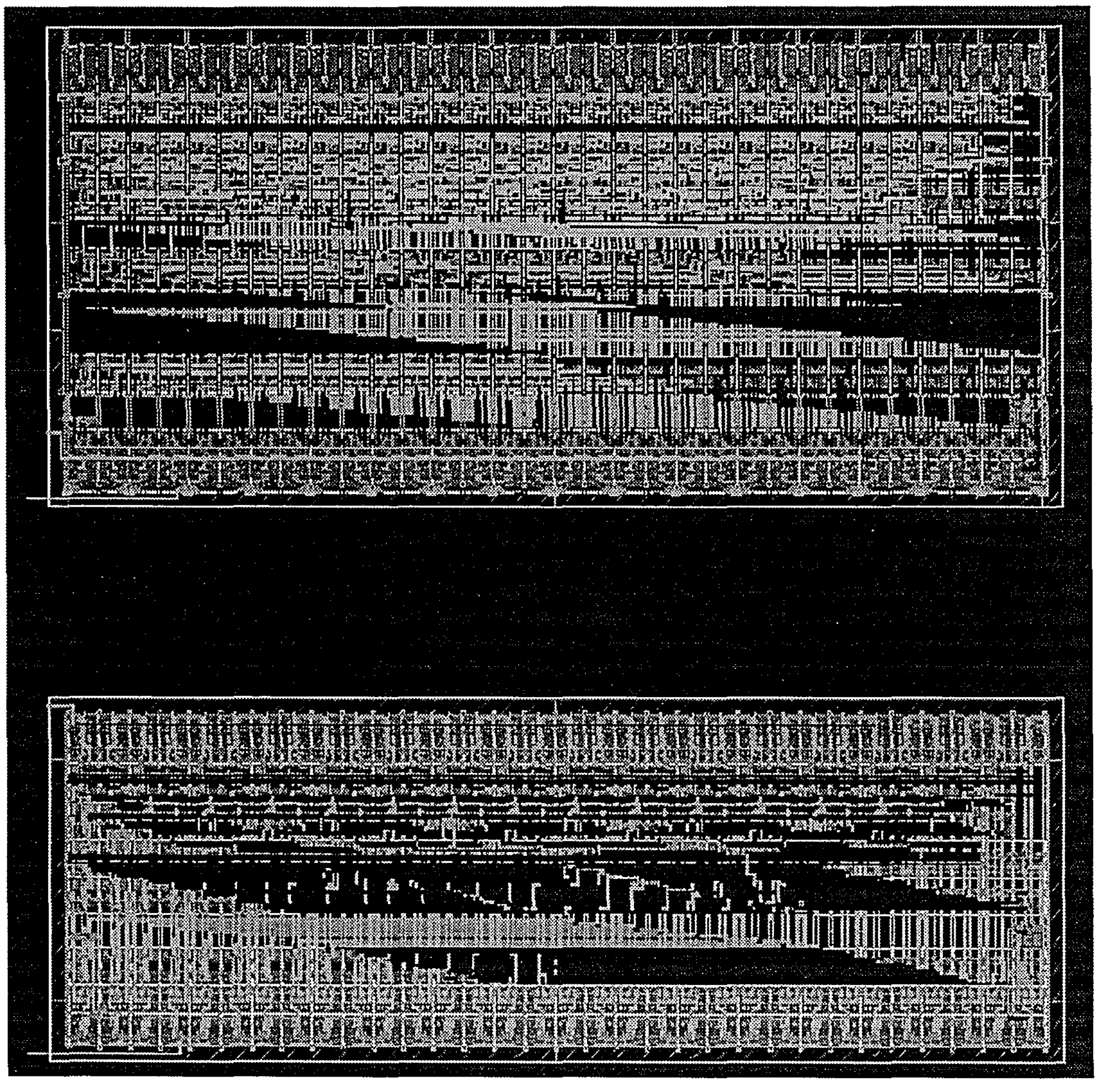


Figure A-10. Two different layout styles according to Table 5, discussed in Section 4.3.

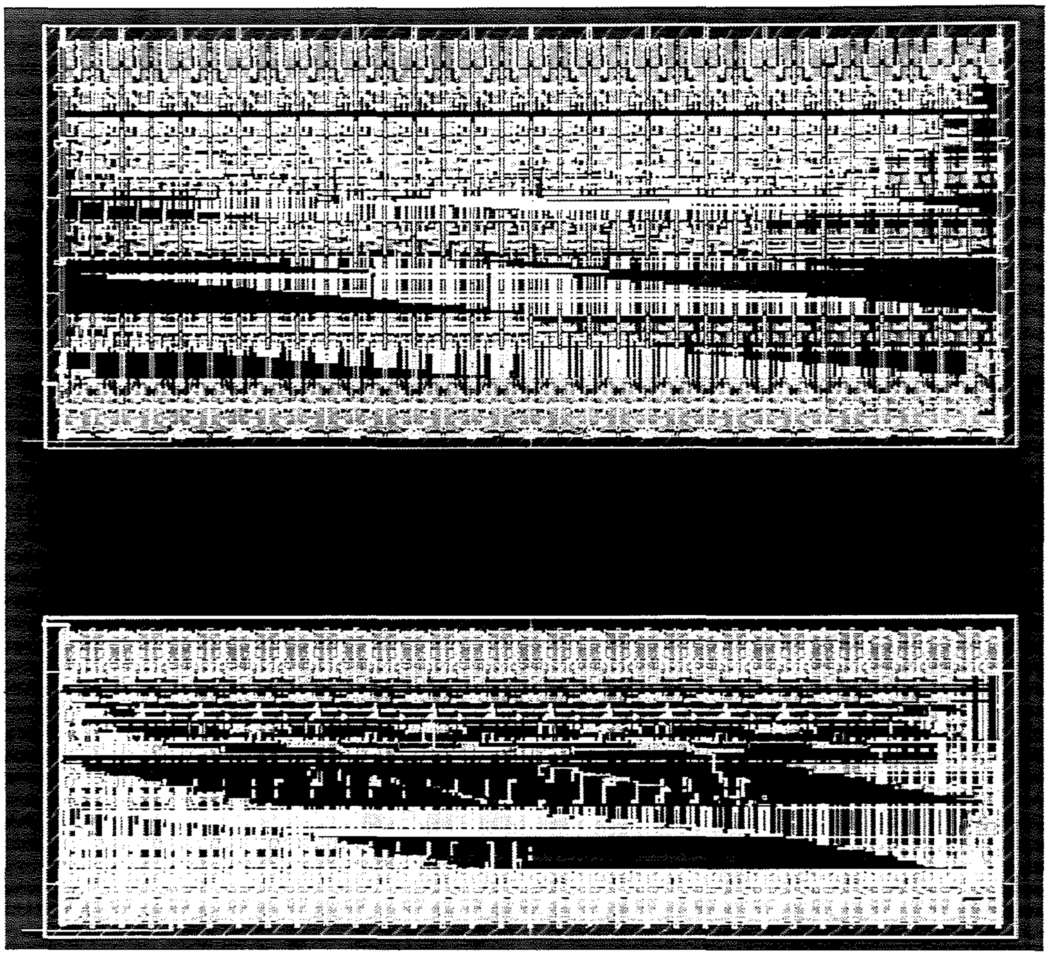


Figure A-10. Two different layout styles according to Table 5, discussed in Section 4.3.

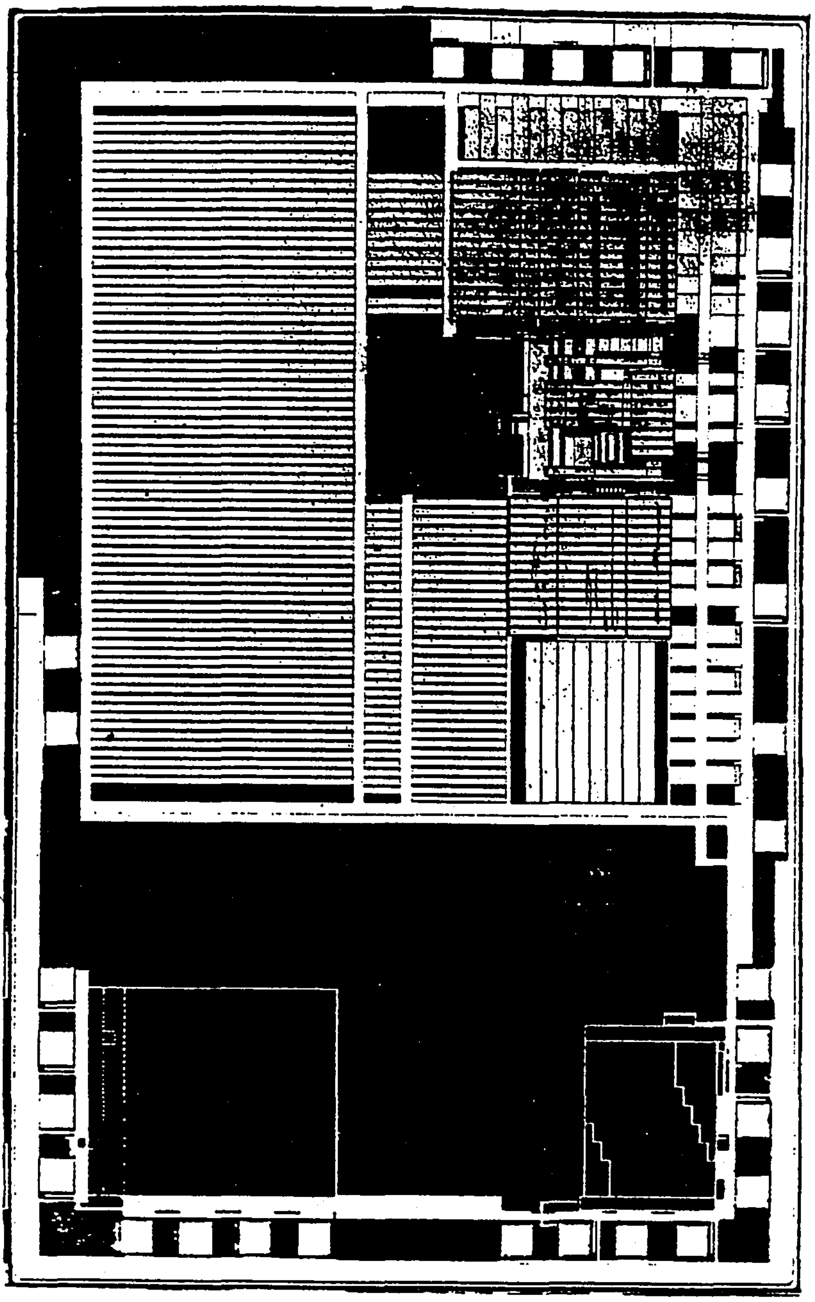


Figure A-11. Photograph of the die.

Vita

Alexander Goldovsky was born in Minsk, Belorussia, on December 5, 1973. He received the B.S. degree in electrical engineering technology from Temple University, Philadelphia, in 1995.

From 1995 he has been employed by the Lucent Technologies. His current research interests include the definition and design of advanced computer systems, with particular emphasis on the introduction of parallelism into computer design and problem solutions.