A COMPARISON OF TWO APPROACHES TO MULTI-OPERAND BINARY ADDITION

D.E. Atkins and S.C. Ong

Program in Computer, Information and Control Engineering
and
Systems Engineering Laboratory
Department of Electrical and Computer Engineering
The University of Michigan
Ann Arbor 48109

INTRODUCTION

This paper presents the results of one phase of a study concerning methods for addition of $P>2$ numbers, each encoded as a vector of digits (digit vector) of length N. Such multi-operand addition has been studied most often in the context of reducing a set of partial products to a single result in the implementation of multiplication. More generalized multi-operand addition, most notably in the form of inner product calculations is, however, central to numerous scientific applications of digital computers. Although multi-operand addition is trivially accomplished by accumulation (iteration in time) in any general purpose machine, demands for very high-speed computation, typified by 2- and 3-D signal processing prompt implementation of dedicated, hardware-intensive structures for multi-operand addition. This study, for example, is motivated in part by requirements for rapid simultaneous addition of up to 100, 16-bit operands in the design of a dedicated processor for real-time reconstruction of 3-D images of the beating heart and breathing lungs [1].

The variety of tradeoffs between iteration in time and hardware, the application of the commutative and transitive properties of addition, and the possibility of using redundant encoding of numbers which restrict carry propagation give rise to numerous published schemes for multi-operand addition. Examples of time-intensive sequential schemes are discussed in [2-8], and arrays of carry-save adders (deferred carry assimililation structures) are cited in [9-15]. If we consider the rows of digit vectors to be reduced by addition to be a matrix of digits, the columns of the matrix may be simultaneously compressed from P digits to a vector of about length log P digits. Subsequent column compressions will lead to a single digit vector representation of the sum of the P operands. These "column compression" ideas are included in [3,10,11,16,17,18]. References [19-21] describe schemes which fall in the middle ground with respect to a time-hardware tradeoff.

Our broad goal is to revisit (especially the elegant work of Dadda [10]) and perhaps extend this literature in light of advances in LSI technology, demands for increased arithmetic computation speed, and innovations in parallel computer architecture. We also intend to study the time-hardware complexity of multi-operand addition across a broad set of finite number representation systems [22], including as a special case, variations in radix. In this paper, however, we restrict ourselves to the specific topic of a cost and performance comparison of multi-operand adders for unsigned, binary encoded operands implemented in the following two ways suggested by commercially available MSI components:

(1) As a tree of (nonredundant, carry-propagate) adders which accept two digit-vectors and produce a one digit-vector result;

(2) As a tree of "carry-save" adders which accept three digit vectors and produce a two digit-vector result.

The analysis will be conducted assuming three variables, namely: number of operands P, length of the operands N (all uniform length), and the maximum fan-in of the device gates, FANIN.

Informally the problem is as follows: carry-save adders produce a result rapidly since they avoid "solving" the carry-propagation recurrence at the expense of reduced operand compression (3-to-2 instead of 2-to-1). Adders which assimilate all carries yield a 2-to-1 operand compression but require auxiliary logic (carry-lookahead logic) in order to "solve" the carry recurrence. Which is "better" for multi-operand addition? Carry-ripple adders avoid the "lookahead" logic but, in general, would be so much slower than a carry-save adder that a comparison would be uninteresting.

In the next section we derive cost and performance measures for a tree of 2-to-1, carry-lookahead structures. In subsequent sections, we perform the analogous task for a tree of 3-to-2 carry-save structures, compare the results, and then recompare with the introduction of restrictions on packaging and chip pin-outs. Since for most of the cost and performance functions we

------

This work was supported by the National Science Foundation, Division of Mathematical and Computer Science under Grant No. MCS-77-03310 and by the U.S. Public Health Service Grants HL-04664, RR-00007 from the National Institutes of Health.

have not found a closed form expression, APL versions of the cost and performance functions are used. A review of the principles of binary carry-lookahead and carry-save addders may be found in [23,24].

COST AND PERFORMANCE OF TWO-TO-ONE STRUCTURES

In this section we first develop a cost and performance function for carry-lookahead adder which accepts two N-bit operands and produces a N+1-bit result. We then apply these results to the analysis of a tree structure which accepts P, N-bit operands to produce one $N+\log_2 P$-bit result.

Two operand carry-lookahead adder

Consider the block diagram in Figure 1 which combines two N-bit operands, A and B, to form a sum S using so-called carry-lookahead techniques to compute the carry vector in the CL box. Assuming that the delay through a logic gate is a uniform time t, then it requires time t to produce carry transmission (T) and generate terms (G). Assuming that an inverter also requires time t, it furthermore requires at best 2t to produce carries C in the CL and 3t more to develop S. For each operand bit, it requires one gate for T, one for G, 3 for P and 4 for S as shown in Figure 2.

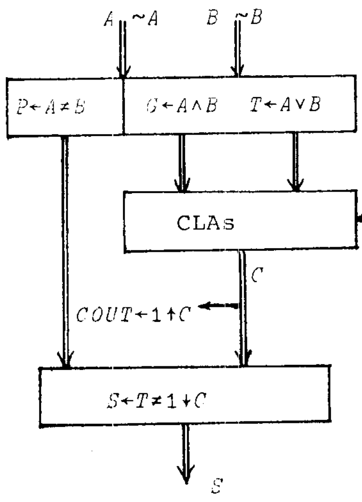

Figure 1. Block diagram of a carry-lookahead adder.

The structure of a CL network which produces 4 carry bits plus a group generate and a group propagate is shown in Figure 3. The dotted lines show a slight variation which is used to produce a true group carry-out at certain boundary positions of both the balanced and optimized adder trees to be presented shortly. The number of positions across which lookahead takes place (i.e. the length of the carry vector produced) is restricted by the fan-in of the constituent gates assuming that operation time does not exceed 2t. If we further impose the restriction that the true group carry must be formed without introducing more delay, then the number of bit positions across which first-level lookahead takes place (NUMBIT) is given by

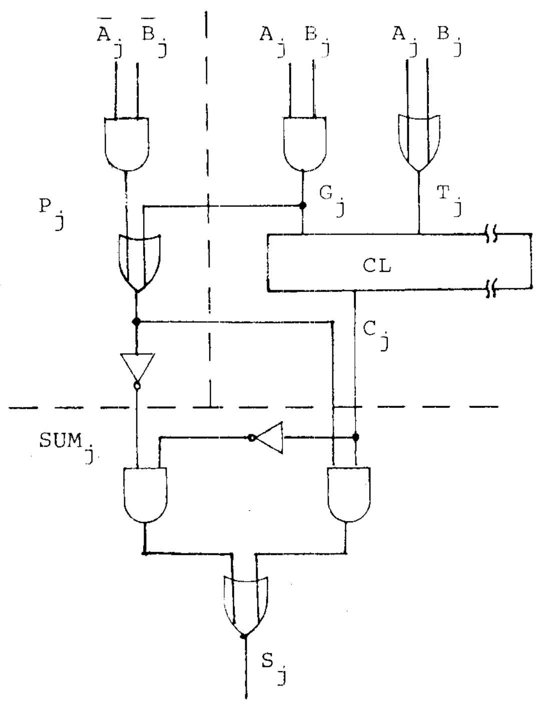

Figure 2. One position of the carry-lookahead adder.

NUMBIT = FANIN-1 (1)

where FANIN is the maximum number of inputs to a primitive gate. In the following equations, for convenience, we use "NUMBIT" as an implicit function of "FANIN".

The number of gates required to implement a CL structure of the type shown in Figure 3 across NUMBIT bits, is the sum of NUMBIT OR gates and a geometric series of AND gates, namely

$$G1'(FANIN) = .5x(NUMBIT^2 + 3xNUMBIT), \quad (2)$$

where NUMBIT is defined in terms of FANIN in Equation (1).

In general, the operand length, N, is greater than NUMBIT and consequently, multi-level carry-lookahead across groups of first level units may be employed. The number of levels of NUMBIT-wide CL units required to produce the carry vector of length N is given by

$$L1(FANIN,N) = \lceil \log_{NUMBIT} N \rceil \quad (3)$$

where $\lceil x \rceil$ is the ceiling function, i.e. the smallest integer $\geq x$.

It follows, under these assumptions, that the time required to produce the sum of two operands of N bits each is

$$T1(L1(FANIN,N)) = 2 \times (2 \times L1(FANIN,N)+1) \times t. \quad (4)$$

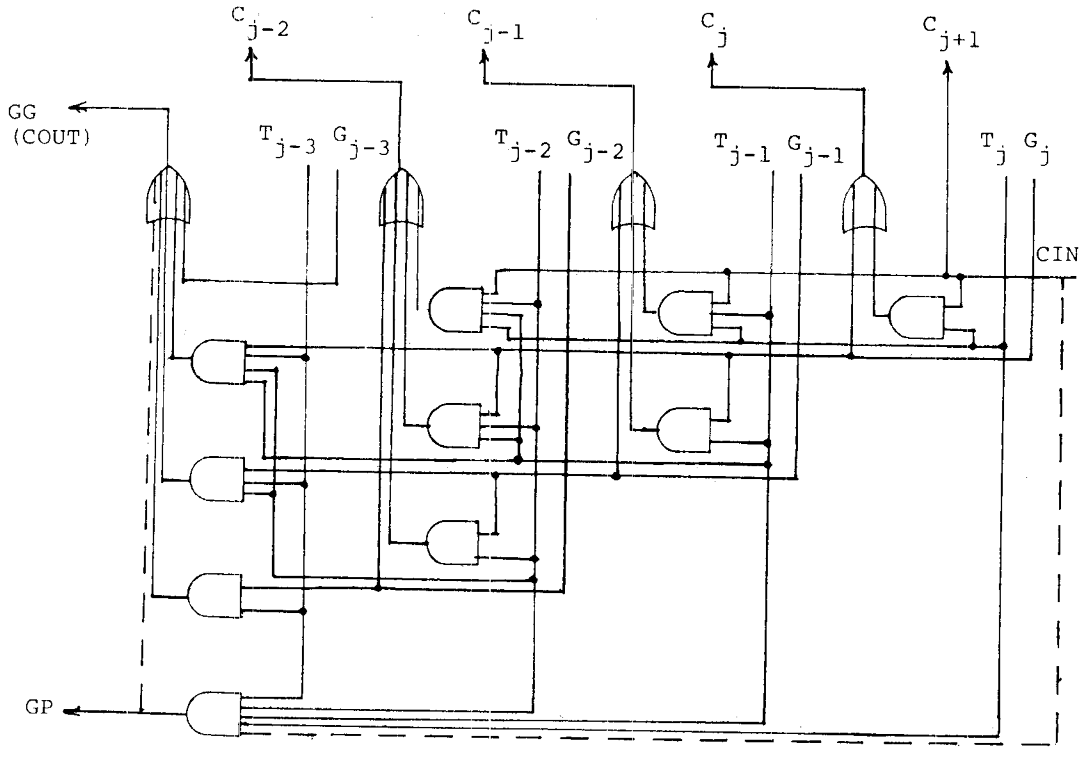

Figure 3. Carry lookahead (CL) unit with NUMBIT=4.

The number of CL units is given by L1 terms of the following

$$U1(N,FANIN) = \lfloor N/NUMBIT \rfloor + \lfloor (\lfloor N/NUMBIT \rfloor)/NUMBIT \rfloor + \ldots \quad (5)$$

where $\lfloor x \rfloor$ is the floor function, i.e. greatest integer $\leq x$.

The number of gates required to implement the two N-bit operand carry-lookahead adder is

$$G1(N,FANIN) = 9 \times N + U1(N,FANIN) \times G1'(FANIN) \quad (6)$$

As an example, the structure consisting of both the solid and dashed nodes shown in Figure 4 is the tree for the case N=32, FANIN=3. In this case, 31 CL units of width 2 are required and the structure produces the final sum in time 22t. The balanced tree of CL units does not, in general, yield the smallest propagation delay. At some point toward the root of the tree it becomes faster to propagate the carries laterally rather than forward and back down from a higher level CL.

Figure 4, minus the dashed boxes together with the lateral carry inter-connections, yields higher performance with less hardware than the full balanced tree. The pruning of the tree is reasoned as follows:

(1) It takes 4t more delay and 1 more CL unit to add a level to the tree, but only 2t more, without increasing the number of CL units, to form the group carry-out. (See Figure 3). Thus the propagation route from module C to B in Figure 4 saves 2t in time and 1 CL in hardware compared to the C to A to B route.

(2) The routes G-F-E-D and G-C-B-D both require delay 6t. However, the former saves two CL units.

(3) It takes the same delay, 4t, for both E-I-H and E-D-H but the former saves 1 CL unit. Similarly, H can be deleted by connecting the carry-out term from I to the carry-in of K.

(4) We need 5t in K to form the sum of the most significant bit, either 2t in the CL plus 3t to form S(sum), or 2t for carry-ripple and 3t for S. In other words, in some parts of the structure, carry-lookahead had no speed advantage over carry ripple.

Following these principles, the minimal gate delay and number of CL units required were computed constructively in the form of APL functions, DELAYNUM and UNIT. These are listed in Figure 5.

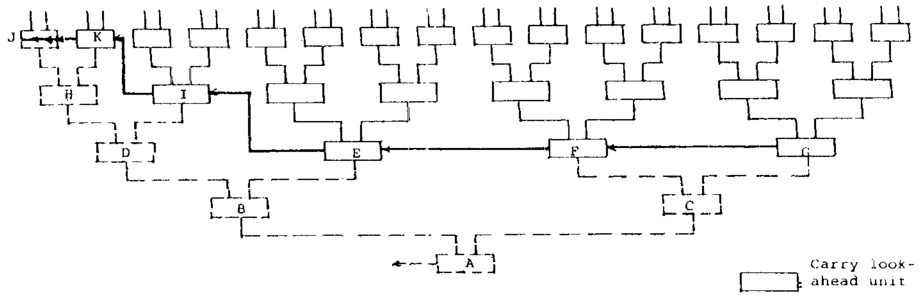

Figure 4. Tree of CL units to produce (N+1)-bit carry vector.

```
     ∇ Z←DELAYNUM N;K
[1]    ⍝ THE NUMBER OF GATE DELAY FOR CARRY LOOK-AHEAD ADDER
[2]    ⍝ NUMBIT IS DEFINED EXTERNALLY
[3]    Z←0
[4]    →(0≥N)/0
[5]    Z←4
[6]    →(1≥N)/0
[7]    Z←6
[8]    →(N≤NUMBIT+1)/0
[9]    K←⌈NUMBIT⍟1+N×NUMBIT-1
[10]   Z←4×K-1
[11]   →(N≤(((NUMBIT*K-1)-1)÷NUMBIT-1)+NUMBIT*K-2)/0
[12]   Z←Z+2
     ∇
```

```
     ∇ Z←UNIT N;TN;I;J;K;L
[1]    ⍝ THE NUMBER OF CLA UNITS
[2]    ⍝ NUMBIT IS DEFINED EXTERNALLY
[3]    ⍝ THE FUNCTION WHICH IS DERIVED CONSTRUCTIVELY IS EFFECTIVE
[4]    ⍝    IF (1) WORDS LENGTH IS LESS THAN 150
[5]    ⍝       (2) 3≤FANIN≤9
[6]    Z←0
[7]    →(N≤2)/0
[8]    →(NUMBIT>2)/THREE
[9]    Z←1
[10]   →(N≤4)/0
[11]   Z←2
[12]   →(N≤5)/0
[13]   I←1
[14]   J←4
[15]   TN←N-5
[16]   Z←N-J
[17]   →(0≥TN←TN-2*I)/0
[18]   Z←Z-1
[19]   →(0≥TN←TN-2*I)/0
[20]   Z←Z+1
[21]   →(0≥TN←TN-2*I)/0
[22]   I←I+1
[23]   J←J+1
[24]   →16
[25]  THREE:→(NUMBIT>3)/MORE
[26]   Z←1
[27]   →(N≤4)/0
[28]   →(N>158)/81
[29]   I←K←1
[30]   J←2
[31]   Z←57
[32]   →(0<TN←N-121)/A1
[33]   Z←30
[34]   →(0<TN←N-67)/A1
[35]   Z←17
[36]   →(0<TN←N-40)/A1
[37]   I←2
[38]   Z←8
[39]   →(0<TN←N-22)/A1
[40]   Z←4
[41]   →(0<TN←N-13)/A1
[42]   K←2
[43]   Z←1
[44]   →(0<TN←N-7)/A1
[45]   TN←N-4
[46]   Z←0
[47]  A1:Z←Z+1
[48]   →(0≥TN←TN-1)/0
[49]   Z←Z+1
[50]   →(0≥TN←TN-2)/0
[51]   →(1≤K←K-1)/A1
[52]   Z←Z+1
[53]   →(0≥TN←TN-1)/0
[54]   Z←Z+1
[55]   →(0≥TN←TN-3)/0
[56]   Z←Z+1
[57]   →(0≥TN←TN-2)/0
[58]   →(1≤I←I-1)/A1
[59]   Z←Z+1
```

Figure 5. Listing of APL functions DELAYNUM and UNIT.

```
[60]  →(0≥TN←TN-1)/0
[61]  Z←Z+1
[62]  →(0≥TN←TN-2)/0
[63]  Z←Z+1
[64]  →(0≥TN←TN-2)/0
[65]  Z←Z+1
[66]  →(0≥TN←TN-3)/0
[67]  Z←Z+1
[68]  →(0≥TN←TN-2)/0
[69]  Z←Z+1
[70]  →(0≥TN←TN-2)/0
[71]  Z←Z+1
[72]  →(0≥TN←TN-1)/0
[73]  Z←Z+1
[74]  →(0≥TN←TN-3)/0
[75]  Z←Z+1
[76]  →(0≥TN←TN-2)/0
[77]  →(1≤J←J-1)/A1
[78] MORE:Z←1
[79]  →(0≥N-NUMBIT+1)/0
[80]  →(0≥TN←N-1+NUMBIT×1+NUMBIT×1+2×NUMBIT)/83
[81]  'OUT OF RANGE'
[82]  →0
[83]  I←J←NUMBIT-2
[84]  Z←1+(NUMBIT+1)*2
[85]  →(0<TN←TN+NUMBIT*3)/A2
[86]  Z←2×NUMBIT+1
[87]  J←J-1
[88]  →(0<TN←N-1+NUMBIT×1+NUMBIT×2)/A2
[89]  Z←NUMBIT+1
[90]  →(0<TN←TN+NUMBIT*2)/A2
[91]  Z←1
[92]  I←I-1
[93]  →(0<TN←N-1+NUMBIT×2)/A2
[94]  Z←0
[95]  TN←TN+NUMBIT
[96] A2:Z←Z+1
[97]  →(0≥TN←TN-1)/0
[98]  Z←Z+1
[99]  →(0≥TN←TN+1-NUMBIT)/0
[100] Z←Z+1
[101] →(0≥TN←TN-1)/0
[102] Z←Z+1
[103] →(0≥TN←TN-NUMBIT)/0
[104] →(0<I←I-1)/102
[105] Z←Z+1
[106] →(0≥TN←TN+1-NUMBIT)/0
[107] Z←Z+1
[108] →(0≥TN←TN-1)/0
[109] Z←Z+1
[110] →(0≥TN←TN+1-NUMBIT)/0
[111] Z←Z+1
[112] →(0≥TN←TN-2)/0
[113] I←NUMBIT-2
[114] Z←Z+1
[115] →(0≥TN←TN-NUMBIT)/0
[116] →(0<I←I-1)/114
[117] Z←Z+1
[118] →(0≥TN←TN+1-NUMBIT)/0
[119] →(0<J←J-1)/109
[120] Z←Z+1
[121] →(0≥TN←TN+1-NUMBIT)/0
[122] Z←Z+1
[123] →(0≥TN←TN-1)/0
[124] I←NUMBIT-2
[125] Z←Z+1
[126] →(0≥TN←TN-NUMBIT)/0
[127] →(0<I←I-1)/125
[128] Z←Z+1
      ∇
```

Figure 5 (cont.). Listing of APL functions DELAYNUM and UNIT.

Figures 6 and 7 are graphs of the results for $N \leq 60$ and $3 \leq$ FANIN ≤ 9. These results are up to 20% less than values predicted by Equations (4) and (5) which assume a balanced full tree, and are up to 50% less than the upper bounds on component count and speed for carry-lookahead adders given by Kuck in [8].

Multi-operand carry-lookahead adder

We are now in the position to analyze a multi-operand structure realized as a tree of two-operand adders each of which employes multi-level carry-lookahead techniques as shown in Figure 8. Note that the operand lengths are N+i-1 at the ith level. The number of levels of 2-to-1 adders to combined $P \geq 2$ operands is given by

$$L2(P) = \lceil \log_2 P \rceil, \qquad (7)$$

where $\lceil x \rceil$ is the ceiling function, i.e. the smallest integer greater than or equal to x.

Now let I21(P,i) and J21(P,i) be the number of input operands and the number of carry-lookahead adders (CLA's), respectively, at the ith level. I21(P,i) equals the number of operand outputs from level i-1 plus the number of operands which have not been combined at a previous level. It follows that

$$I21(P,i) = J21(P,i-1)+I21(P,i-1)-2 \times J21(P,i-1) \qquad (8)$$

with I21(P,1) = P.

$$J21(P,i)=\lfloor I21(P,i)/2 \rfloor \qquad (9)$$

The total number of CLA units is [25]

$$U2'(P) = P-1. \qquad (10)$$

Upper bounds on performance and number of gates may be derived from Equations (1-9). In particular, the total time to compute the P, N-bit operand sum is the total of the delay at each level, namely

$$T2(N,FANIN,P) = \sum_{i=1}^{L2(P)} T1(L1(FANIN,N+i-1)). \qquad (11)$$

A bound on the number of CL units required is the sum of the number of CLA groups times the number of CL units per CLA for each level.

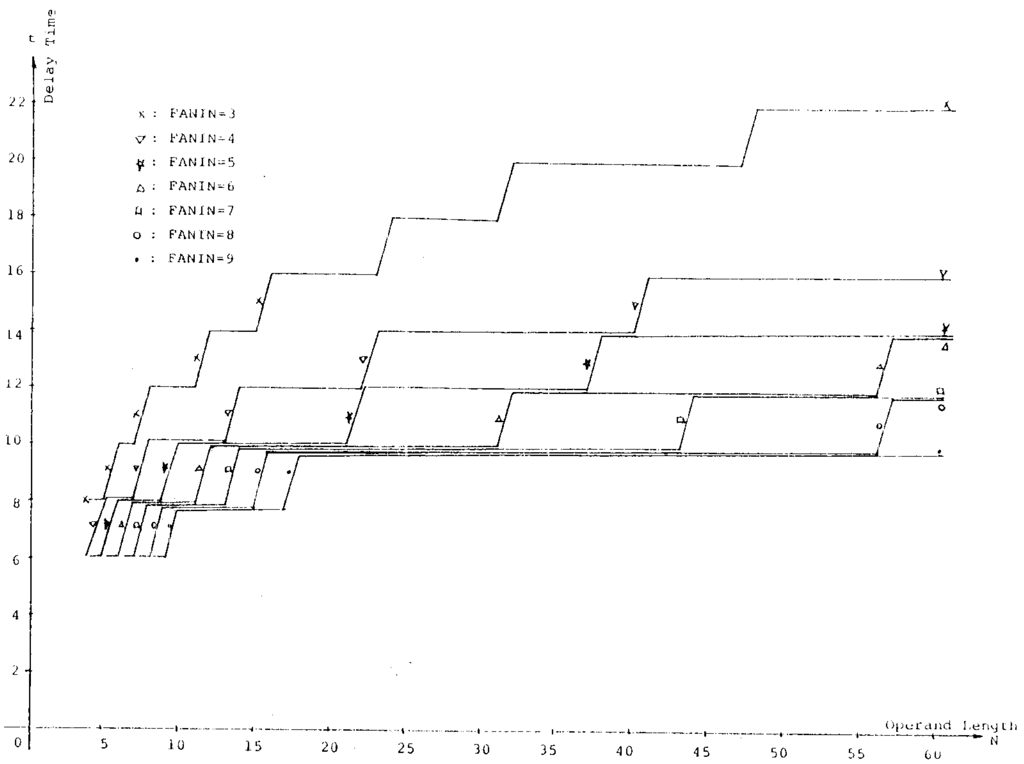

Figure 6. Operational delay vs. operand length and fan-in for carry-lookahead adder.

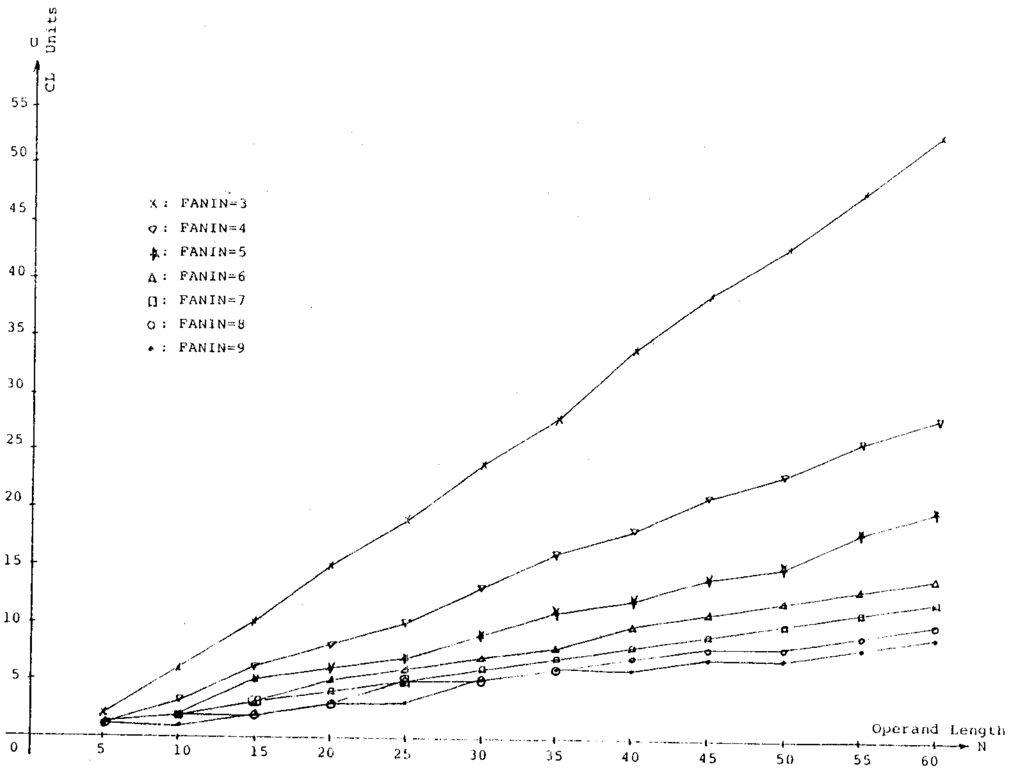

Figure 7. Required number of CL units vs. operand length and fanin for carry-lookahead adder.

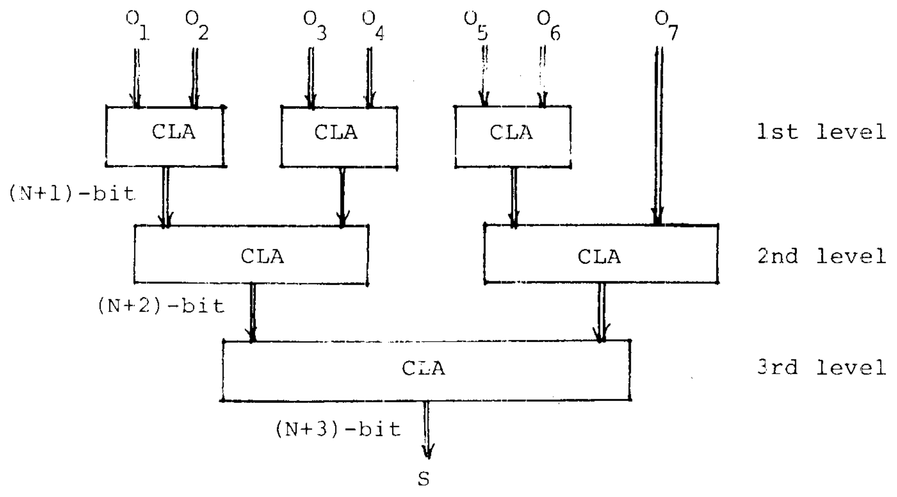

Figure 8. Example of 2-to-1 tree for P=7.

$$U2(N,FANIN,P) = \sum_{i=1}^{L2(P)} J21(P,i) \times U1(N+i-1,FANIN). \quad (12)$$

The total number of gates required for the entire structure is bounded by

$$G2(N,FANIN,P) = \sum_{i=1}^{L2(P)} J21(P,i) \times G1(N+i-1,FANIN). \quad (13)$$

Our numerical results are based, not upon these bounds, but rather upon the optimized CLA′ structures described previously. The functions analogous to T2, U2 and G2 for these optimized structures are denoted T2", U2", and G2", respectively. These functions are evaluated using the APL functions TIME21, CLA21, and GATE21 as listed in Figure 9.

COST AND PERFORMANCE OF THREE-TO-TWO STRUCTURES

Three operand carry-save adder

By "carry-save adder" (CSA) we refer to a logic network which accepts three digit vector operands and produces two, the sum of which is equal to the sum of the three. The truth table for the specific version assumed here is shown in Table 1. This structure is a special case of the carry-save/ borrow-save networks studied by Robertson in [26]. Figure 10 shows a gate- level realization of one position of a CSA for FANIN=4. If FANIN=3 then four levels are required, i.e.

$$T3(FANIN) = 3t \text{ for } FANIN \geq 4 \quad (14)$$

and the number of gates is given by

$$G3(N) = 12 \times N \text{ with } FANIN \geq 4. \quad (15)$$

A B C	PC	PS
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	1

Table 1. Truth table for carry-save adder.

Multi-operand carry-save adder

We now consider a 3-to-2 array sometimes called a Wallace adder tree as shown in Figure 11. Let I32(P,i) be the number of input operands to the ith level and let J32(P,i) be the number of CSA groups required at the ith level. It follows that

$$I32(P,i) = J32(P,i-1)\times 2+I32(P,i-1)-J32(P,i-1)\times 3 \quad (16)$$

with

$$I32(P,1) = P.$$

$$J32(P,i) = \lfloor I32(P,i)/3 \rfloor \quad (17)$$

The total number of CSA groups is

$$U3 = P-2 \quad (18)$$

```
     ∇ Z←TIME21 P;I
[1]    ⍝ THE NUMBER OF GATE-DELAYS FOR MULTIPLE OPERAND ADDITION
[2]    ⍝   OF TWO-TO-ONE TREE STRUCTURE
[3]    ⍝ WORDS LENGTH N, FANIN ARE DEFINED EXTERNALLY
[4]    I←(⌈2⍟P)-1
[5]    Z←0
[6]    Z←Z+DELAYNUM N+I
[7]    →(0≤I←I-1)/6
     ∇


     ∇ Z←CLA21 P;I;J
[1]    ⍝ THE NUMBER OF UNITS FOR MULTIPLE OPERAND ADDITION
[2]    ⍝   OF TWO-TO-ONE TREE STRUCTURE
[3]    ⍝ WORD LENGTH N, FANIN ARE DEFINED EXTERNALLY
[4]    Z←I←0
[5]    J←⌊P÷2
[6]    Z←Z+J×UNIT N+I
[7]    P←J+P-J×2
[8]    I←I+1
[9]    →(P>1)/5
     ∇


     ∇ Z←GATE21 P;I;J
[1]    ⍝ THE NUMBER OF GATES FOR MULTIPLE OPERAND ADDITION
[2]    ⍝   OF TWO-TO-ONE TREE STRUCTURE
[3]    ⍝ WORDS LENGTH N, FANIN ARE DEFINED EXTERNALLY
[4]    Z←I←0
[5]    J←⌊P÷2
[6]    Z←Z+J×NUMGATE N+I
[7]    P←J+P-J×2
[8]    I←I+1
[9]    →(P>1)/5
     ∇
```

Figure 9. Listing of APL functions TIME21, CLA21, and GATE21.

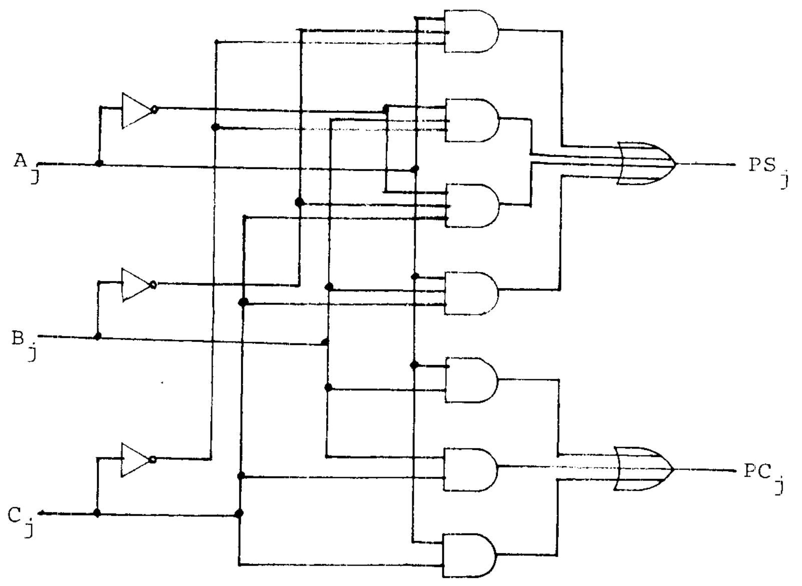

Figure 10. One position of a carry-save adder.

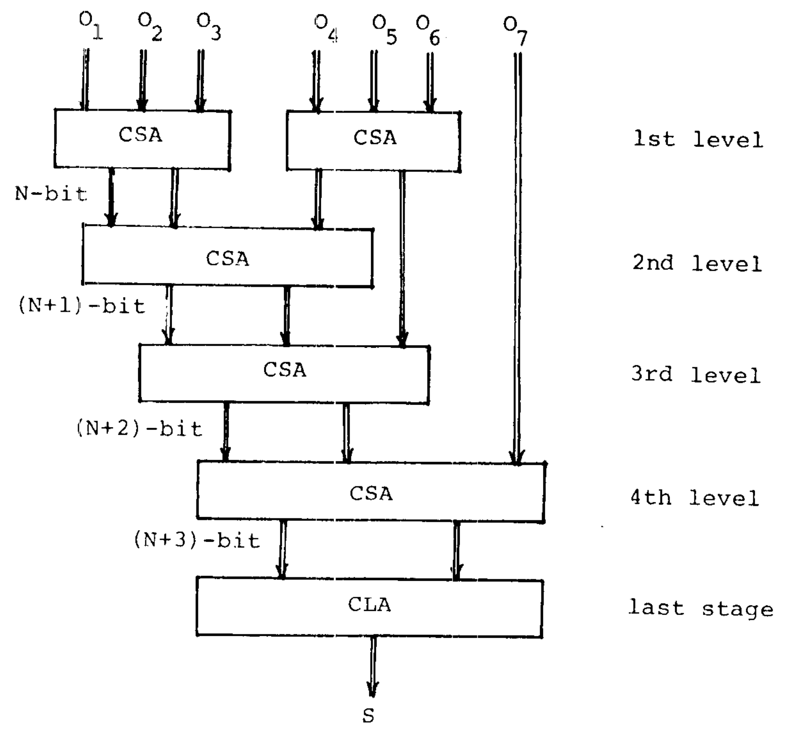

Figure 11. Example of 3-to-2 tree for P=7.

The number of levels of CSA required, L3(P), is computed iteratively as described by the APL function LEVEL32 in Figure 12.

The total time to compute the P operand sum, including carry assimilation as a final step is given by

$$T4(N,FANIN,P) = T1(L1(FANIN,N+L3))+T3(FANIN)\times L3(P). \qquad (19)$$

The number of CSA units required is

$$U4(N,FANIN,P) = \sum_{i=1}^{L3(P)} J32(P,i) \times (N+i-1). \qquad (20)$$

The total number of gates required, including the N+L3 bit carry-lookahead adder at the output end of the tree, is given by

$$G4(N,FANIN,P) = G2(N+L3,FANIN)+G3(N)\times U4(N,FANIN,P). \qquad (21)$$

Figure 12 is a listing of the APL functions which compute L3, T4, U4, G4. These functions are denoted LEVEL32, TIME32, CSA32, and GATE32, respectively.

NUMERICAL EVALUATION AND COMPARISON OF COST/PERFORMANCE FUNCTIONS

Our ultimate concern is with comparison of two pairs of functions of three variables, namely T2", G2", T4 and G4. Our approach has been to evaluate numerical examples and attempt to draw general conclusions. Figure 13 (a and b) shows a comparison of the two methods for fixed operand length and fan-in. In this case the 3-to-2 structure exhibits greater speed with lower gate count. Figure 14 (a and b) compares the structures with fixed values of P and FANIN as N, the operand length, varies. Here the 3-to-2 case is always faster but requires slightly more gates for N<14.

Tables 2 and 3 display some conclusions from trial computations. For FANIN values of 3 through 9, Table 2 lists the lower bound on N such that the time delay and gate count are smaller for the 3-to-2 structures than the 2-to-1 structures for any P in the range $3 \leq P \leq 40$. For example, if FANIN = 8, then for operand lengths of 10 bits or more, a carry-save adder tree is faster and requires fewer gates than a tree of carry-lookahead adders. If

```
      ∇ Z←LEVEL32 P;J
[1]     ⍝ THE NUMBER OF LEVELS FOR THREE-TO-TWO TREE STRUCTURE
[2]     Z←0
[3]     →(P≤2)/0
[4]     J←⌊P÷3
[5]     P←(J×2)+P-J×3
[6]     Z←Z+1
[7]     →3
      ∇


      ∇ Z←TIME32 P;I
[1]     ⍝ THE NUMBER OF GATE-DELAYS FOR MULTIPLE OPERAND ADDITION
[2]     ⍝    OF THREE-TO-TWO TREE STRUCTURE
[3]     ⍝ WORDS LENGTH N, FANIN AND CSAT ARE DEFINED EXTERNALLY
[4]     I←LEVEL32 P
[5]     Z←(CSAT×I)+DELAYNUM N+I
      ∇


      ∇ Z←CSA32 P;I;J
[1]     ⍝ THE NUMBER OF CARRY SAVE ADDERS FOR MULTIPLE OPERAND
[2]     ⍝    ADDITION OF THREE-TO-TWO TREE STRUCTURE
[3]     Z←I←0
[4]     →(P≤2)/0
[5]     J←⌊P÷3
[6]     Z←Z+J×N+I
[7]     P←(J×2)+P-J×3
[8]     I←I+1
[9]     →4
      ∇


      ∇ Z←GATE32 P
[1]     ⍝ THE NUMBER OF GATES FOR MULTIPLE OPERAND ADDITION
[2]     ⍝    OF THREE-TO-TWO TREE STRUCTURE
[3]     ⍝ WORDS LENGTH N, FANIN AND CSAG ARE DEFINED EXTERNALLY
[4]     Z←(CSAG×CSA32 P)+NUMGATE N+LEVEL32 P
      ∇
```

Figure 12. Listing of APL functions LEVEL32, TIME32, CSA32, and GATE32.

N = 9 the delay is less but the gate count may not be. If N is less than 8 we cannot make a general statement about the relative cost and performance, but of course these could be computed for specific choices of N, P, and FANIN.

Table 3 lists the lower bound on P such that the time delay and gate count are smaller for the 3-to-2 structures that the 2-to-1 structures. Again if P is less than the bound, we cannot make a general statement.

COST COMPARISONS UNDER PACKAGING CONSTRAINTS

Our measure of cost or complexity has so far been in terms of number of gates assuming a given limit of fan-in. One of our reasons for concentrating on structures based upon carry-lookahead and carry-save addsers was the fact that off-the-shelf integrated circuit components are available to implement these devices. At this point we revise of cost measure to be the number of circuit packages required under various restrictions on pin-outs.

It requires NUMBIT x 3 + 2 pins for each CL unit and 5 pins for each CSA unit and therefore

$$(PIN - 2) \geq H1 \times (NUMBIT \times 3 + 2) \text{ for CLA and,} \quad (22)$$

$$(PIN - 2) \geq H2 \times 5 \quad \text{for} \quad CSA, \quad (23)$$

where H1 and H2 are the number of CLA and CSA units per package (chips) with PIN external connections. Two pins are assumed to be used for power and ground. Figure 15 is an APL function to calculate the number of chips required for both the CLA and CSA based schemes under restrictions of gate fan-in and package pin-outs. Figure 16 depicts some results from this function. Figure 17 summaries results from multiple evaluations of of the function CHIP. Notice that for a given value of FANIN (a column in Figure 17), the chip count for the 3-to-2 schemes are the smallest for larger values of N. In general the CSA structures do not appear to offer advantage over the CLA structures for as many choices of design parameters when we add the packaging restriction.

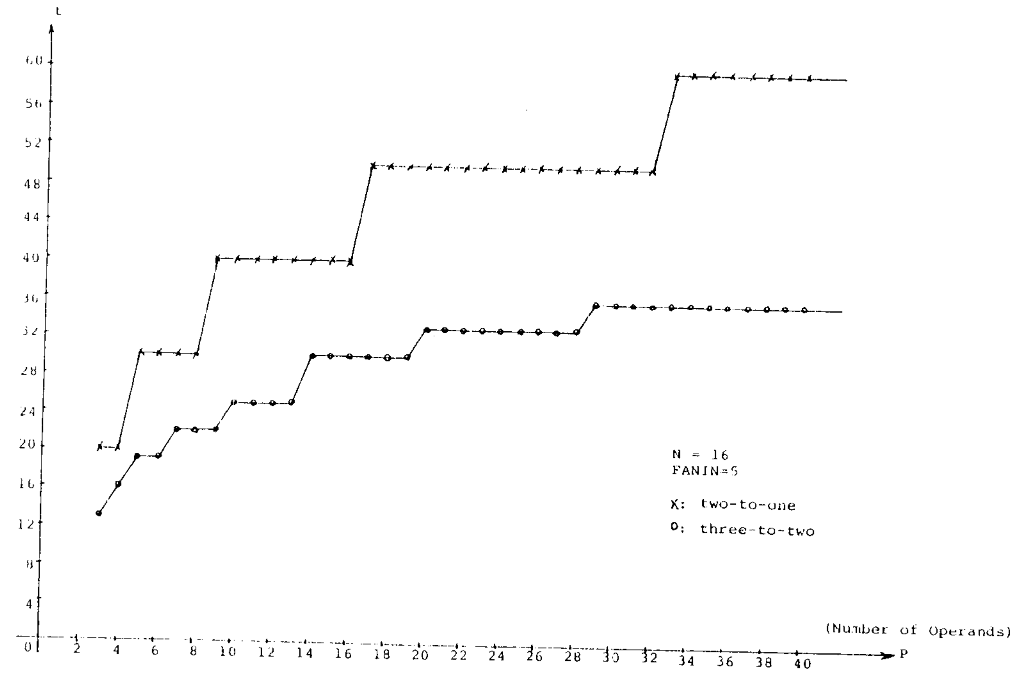

Figure 13(a). Operational delay vs. number of operands for 2-to-1 and 3-to-2 structures with N=16, FANIN=5.

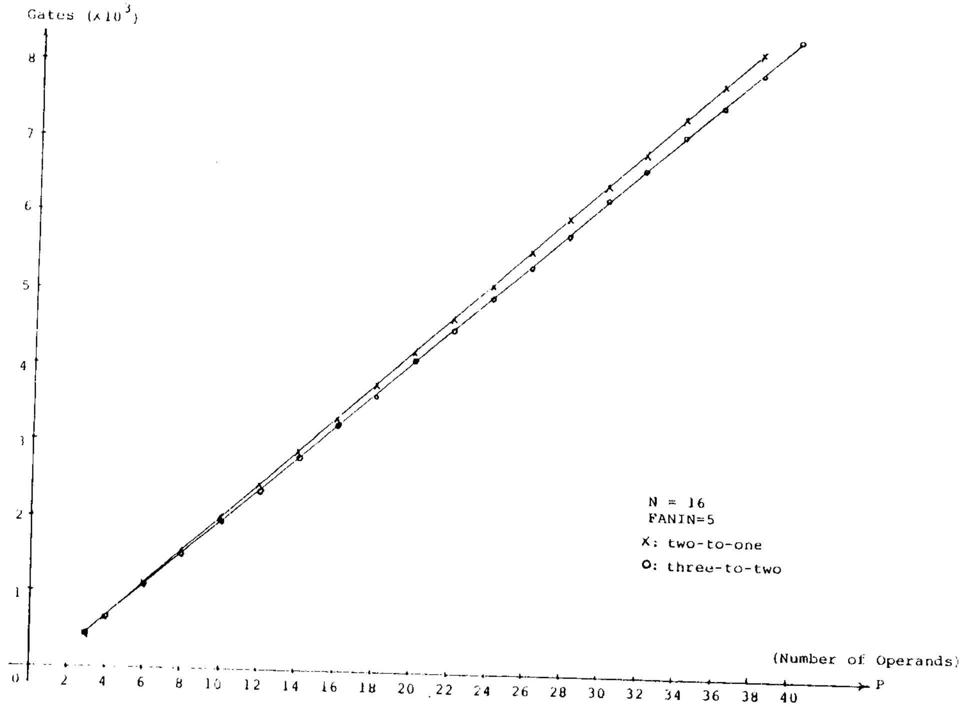

Figure 13(b). Required number of gates vs. number of operands for 2-to-1 and 3-to-2 structures with N=16, FANIN=5.

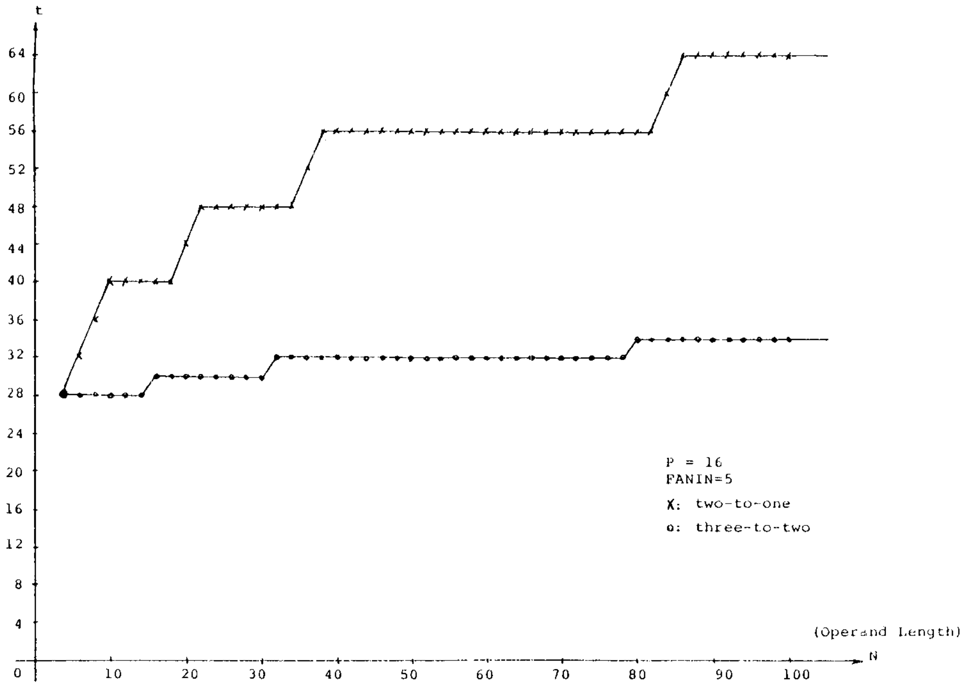

Figure 14(a). Operational delay vs. operand length for 2-to-1 and 3-to-2 structures with P=16 and FANIN=5.

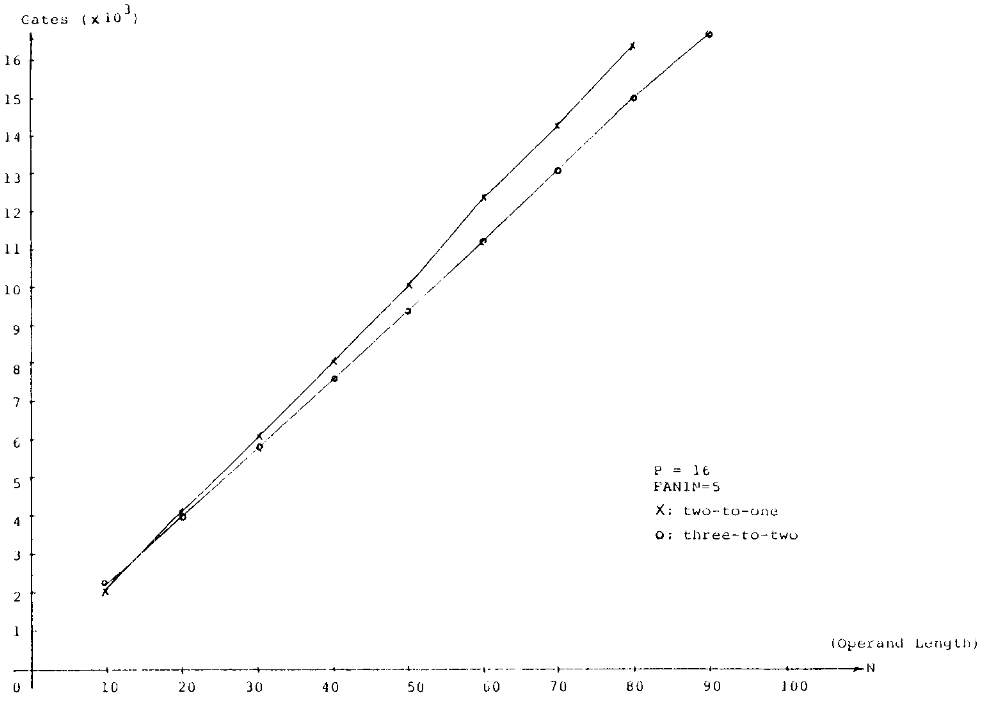

Figure 14(b). Required number of gates vs. operand length for 2-to-1 and 3-to-2 structures with P=16 and FANIN=5.

```
      ∇ Z←CHIP P;DEN;CL21;CL32;CS32;C32
[1]     ⍝ THE NUMBER OF LSI/MSI CHIPS FOR MULTIPLE OPERAND ADDITION
[2]     ⍝   OF TWO-TO-ONE AND THREE-TO-TWO TREE STRUCTURE
[3]     ⍝ WORD LENGTH N, FANIN, PIN ARE DEFINED EXTERNALLY
[4]     DEN←⌊(PIN-2)÷2+NUMBIT×3
[5]     →(0=DEN)/16
[6]     CL21←⌈(CLA21 P)÷DEN
[7]     CL32←⌈(CLA32 P)÷DEN
[8]     CS32←⌈(CSA32 P)÷⌊(PIN-2)÷5
[9]     C32←CL32+CS32
[10]    'CHIPCLA21:  ';CL21
[11]    ' '
[12]    'CHIPCLA32:  ';CL32
[13]    'CHIPCSA32:  ';CS32
[14]    'CHIP32:     ';C32
[15]    →0
[16]    'FAN IN IS TOO LARGE'
      ∇
```

Figure 15. Listing of APL function CHIP.

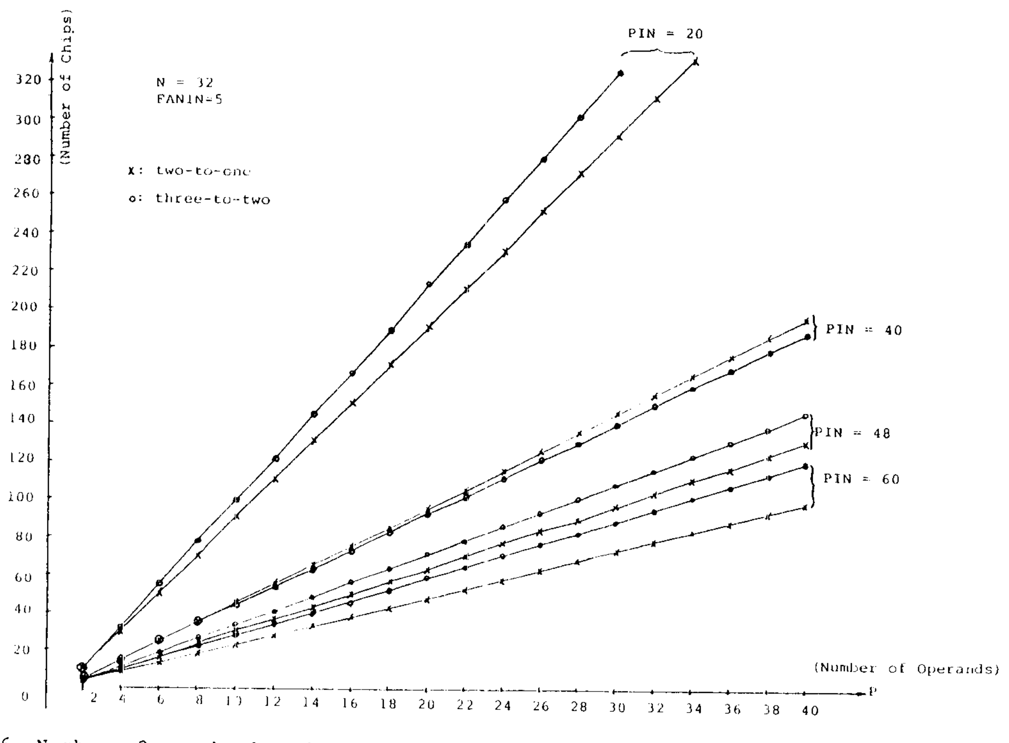

Figure 16. Number of required packages vs. number of operands and pinouts for 2-to-1 and 3-to-2 structures with N=32 and FANIN=5.

	FANIN							
OPERAND LENGTH	3	4	5	6	7	8	9	
8	L	L	L	L	X	X	X	
16	L	S	L	L	X	X	X	PIN=20
32	S	S	L	L	X	X	X	
64	S	S	L	L	X	X	X	
8	L	L	L	L	L	L	L	
16	?	L	?	L	S	L	L	PIN=40
32	S	L	S	L	S	S	S	
64	S	S	S	L	S	S	L	
8	L	L	L	L	L	L	L	
16	L	L	L	L	L	L	?	PIN=48
32	L	L	L	L	L	L	S	
64	S	L	S	L	L	L	S	
8	L	L	L	L	L	L	L	
16	L	L	L	L	L	L	L	PIN=60
32	S	L	L	L	S	L	L	
64	S	L	L	L	S	L	L	

L denotes that the carry-Lookahead scheme requires fewer chips.
S denotes that the carry-Save scheme requires fewer chips.
? Denotes that the relative costs cross over depending upon P.
X indicates that the gate fan-in is limited by number of package connections.

Figure 17. Comparison of package counts for 3-to-2 versus 2-to-1 schemes under fanin and package connection constraints.

FANIN	TIME-DELAY	GATES
3	$N \geq 5$	$N \geq 51$
4	$N \geq 4$	$N \geq 24$
5	$N \geq 5$	$N \geq 14$
6	$N \geq 6$	$N \geq 17$
7	$N \geq 7$	$N \geq 14$
8	$N \geq 8$	$N \geq 10$
9	$N \geq 9$	$N \geq 10$

Table 2. Operand length such that 3-to-2 scheme is faster and/or requires fewer gates than 2-to-1 ($4 \leq N \leq 64$, $3 \leq P \leq 40$).

FANIN	TIME-DELAY	GATES
3	$P \geq 9$	-
4	$P \geq 3$	-
5	$P \geq 5$	-
6	$P \geq 9$	-
7	$P \geq 17$	-
8	$P \geq 33$	$P \geq 21$
9	$P \geq 33$	$P \geq 9$

"-" denotes that the 3-to-2 case may not require less gates.

Table 3. Number of operands such that 3-to-2 scheme is faster and/or requires fewer gates than 2-to-1 ($4 \leq N \leq 64$, $3 \leq P \leq 40$).

REFERENCES

1. B.K. Gilbert, et. al., "Ultra high speed transaxial image reconstruction of the heart, lungs, and circulation via numerical approximation methods and optimized processor architecture," accepted for publication in Computers and Biomedical Research.

2. R. O. Berg and L. C. Kinney, "Serial adders with overflow correction", IEEE Trans. Comp., vol. C-20, pp. 668-671, June 1971.

3. A. Svoboba, "Adder with distributed control", IEEE Trans. Comp., vol. C-19, pp. 749-751, Aug. 1970.

4. H. L. Garner, "Number systems and arithmetic", Advances in Computers, F. L. Alt and M. Rubinoff (ed.), vol. 6, Academic Press, pp. 131-194, 1965.

5. A. J. Atrubin, "A one-dimensional real-time iterative multiplier", IEEE Trans. Electron. Comput., vol. EC-14, pp. 394-399, June 1965.

6. O. L. MacSorley, "High speed arithmetic in binary computers", Proc. IRE, vol. 49, pp. 67-91, Jan. 1961.

7. N. G. Kingsburg, "High speed binary multiplier", Electron. Lett., vol. 7, pp. 277-278, May 20, 1971.

8. D. J. Kuck, The Structure of Computers and Computations, vol. I, class notes (to be soon published as a text), Department of Computer Science, University of Illinois, Urbana, Illinois.

9. C. S. Wallace, "A suggestion for a fast multiplier", IEEE Trans. Electron. Comput., vol. EC-13, pp. 14-17, Feb. 1964.

10. L. Dadda, "Some schemes for parallel multipliers", Alta Freq., vol. 34, pp. 349-356, May 1965.

11. L. Dadda and D. Ferrari, "Digital multipliers a unified approach", Alta Freq., vol. 37, pp. 1079-1086, Nov. 1968.

12. J. A. Gibson and R. W. Gibbard, "Synthesis and comparison of two's complement parallel multipliers", IEEE Trans. Comp., vol. C-24, pp. 1020-1027, Oct. 1975.

13. J. C. Majithia and R. Kitai, "An iterative array for multiplication of signed binary numbers", IEEE Trans. Comp., vol. C-20, pp. 214-216, Feb. 1971.

14. S. D. Pezaris, "A 40-ns 17-bit by 17-bit array multiplier", IEEE Trans. Comp., vol. C-20, pp. 442-447, April 1971.

15. A. Habibi and P. A. Wintz, "Fast multipliers", IEEE Trans. Comp., vol. C-19, pp. 153-157, Feb. 1970.

16. W. J. Stenzel, W. J. Kubitz and G. H. Garcia, "A compact high-speed parallel multiplication scheme", IEEE Trans. Comp., vol. C-26, pp. 948-957, Oct. 1977.

17. I. T. Ho and T. C. Chen, "Multiple addition by residue threshold functions and their representation by array logic", IEEE Trans. Comp., vol. C-22, pp. 762-767, Aug. 1973.

18. N. Kouvaras, D. Lagoyannis and L. Ponticopoulos, "A digital system of simultaneous addition of several binary numbers", IEEE Trans. Comp., vol. C-17, pp. 992-997, Oct. 1968.

19. E. E. Swartzlander, "The quasi-serial multiplier", IEEE Trans. Comp., vol. C-22, pp. 317-321, April 1973.

20. E. E. Swartzlander, "Parallel counters", IEEE Trans. Comp., vol. C-22, pp. 1021-1024, Nov. 1973.

21. S. Singh and R. Waxman, "Multiple operand addition and multiplication", IEEE Trans. Comp., vol. C-22, pp. 113-120, Feb. 1973.

22. D. E. Atkins, "A suggested approach to computer arithmetic for designers of multi-valued logic processors", Proceedings of the 8th International Symposium on Multi-Valued Logic, IEEE No. 78 CH1366-4C, May 1978.

23. F.J. Hill and G.R. Peterson, Digital Systems: Hardware Organization and Design, Chp. 11, John Wiley and Sons, N.Y., 1973.

24. G.A. Blaauw, Digital system implementation, Chp.2. Prentice-Hall, 1976.

25. J.R. Armstrong, "The efficient single output (d,r) circuit," Proceedings of Eighth Annual Southeastern Symposium on System Theory, pp. 63-66, April 1976.

26. J.E. Robertson, "A deterministic procedure for the design of carry-save adders and borrow-save subtracters," Department of Computer Science Report No. 235, University of Illinois, Urbana, July 1967.