A Way to Build Efficient Carry-Skip Adders

ALAIN GUYOT, BERTRAND HOCHET, AND JEAN-MICHEL MULLER

Abstract—**In this paper, we present a way to obtain efficient carry-skip adders, built with blocks of different sizes in VLSI technologies. We give some results about two-level carry-skip adders. We reduce our optimization problem to a geometrical problem, solved by means of an algorithm easily implemented on a microcomputer. Then we present an example of the realization of such an adder.**

Index Terms—**Carry-skip adders, VLSI design.**

I. Introduction

In a classical *ripple-carry* adder, the carry is propagated in a time proportional to the size of the adder, and the sum S of two N-bit binary numbers A and B is obtained by means of the well-known relations

$$C_0 = 0$$

$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = A_iB_i + A_iC_i + B_iC_i = A_iB_i + P_iC_i$$

where

$$\oplus \text{ is the XOR function}$$

$$P_i = A_i \oplus B_i.$$

Since C_{i+1} is dependent upon C_i, one might think that the problem of adding two N bit numbers is intrinsically linear; but it is worth noting that if the terms A_i and B_i are equal ($P_i = 0$), there is no need to know C_i in order to obtain C_{i+1}. (If $A_i = B_i = 0$, then $C_{i+1} = 0$; if $A_i = B_i = 1$, then $C_{i+1} = 1$.) Thus, it is possible to build an adder whose average time of computation would be proportional to the average size of the longest chain of different digits of A and B. It is possible to show that this average size is upper bounded by $\log_2 N$ (this result is due to Burks, Goldstine, and Von Neumann [2]).

For instance, in the following example, all the blocks separated by slashes can be added in parallel:

1010|0001|1001001101|001|101

1101|0110|1110110010|010|110.

Practically, the adder must be based on "worst case" design, thus we have to minimize the *worst time* of addition, instead of the *average time*. In the previous case, we exploit the occurrences $A_i = B_i$, but large chains of consecutive bits i may arise, such that $A_i \neq B_i$. So, we may design an adder divided in blocks, where a special circuit associated with each block detects quickly if all the bits to be added are different ($P_i = 1$ in all the blocks). In this case the carry entering into the block may directly bypass it and so be transmitted to the next block. This is the principle on which carry-skip adders are based.

These adders are actually simple ripple-carry adders with a special speedup carry chain (skip chain). This chain defines the distribution of the blocks. This results in great topological regularity (due to the ripple-carry part), small area, good modularity, and design simplicity. Thus, carry-skip adders are frequently used for adders of datawidth larger than or equal to 32 bits.

In each block, a special circuit compares A_i and B_i for the different cells of the block.

- If *for each cell in the block* $A_i \neq B_i$ ($P_i = 1$), then we shall say that a carry can *skip* over the block. In this case, *the carry must immediately be transmitted to the next block.*
- If $A_i = B_i$ for some i in the block ($P_i = 0$), we shall say that a carry is *generated* in the block (whether 1 or 0).

It is worth noting that if in each block there exists a cell i such that $A_i = B_i$, no block is skipped, but in each block, a carry is generated; thus, the carries are propagated in parallel and the total time of computation is bounded by the time of propagation of a carry in the largest block.

Our purpose here is to find a configuration of blocks which minimizes the worst time of computation, i.e., which minimizes the longest "life" of a carry (we shall call *life* of a carry the time between its generation and the generation of the next carry). In 1960, Lehman and Burla found the best configuration with blocks of *equal size* [9], and showed that with such a configuration, two N bit numbers are added in a time roughly proportional to $\sqrt{N}$, and suggested taking blocks of different sizes.

In 1967, Majerski [10] studied this last case and attempted to reduce the number of skips, but with VLSI techniques, the number of gates has no importance: minimizing the total area and the time of computation is the major goal.

In any case, the models of [9] and [10] are limited in regard to the ratio a between the skip time (the time needed by a carry to skip a block) and the ripple time (the time needed by a carry to pass through a ripple-adder cell); they considered only the case $a = 1$.

In [1], Barnes and Oklobdzija gave a strategy for finding optimal sizes of blocks, but their method is valid only if the ratio a is an *integer* which verifies $2 \leq a \leq 7$. (See Theorem

Manuscript received August 6, 1985; revised September 16, 1986 and March 13, 1987. This work has been supported in part by the GRECO C^3 of the french Centre National de la Recherche Scientifique.

A. Guyot and B. Hochet are with Laboratoire TIM3, Institut National Polytechnique de Grenoble, 38031 Grenoble Cedex, France.

J.-M. Muller is with CNRS, Laboratoire TIM3, 38031 Grenoble Cedex, France.

IEEE Log Number 8715428.

1 of [1, p. 5].) Since the logical gates of a carry skip adder are built with a small number of different transistors, it is not unreasonable to suppose that a is equal to a *rational* number p/q, with q small. However, assuming that a is an integer is only a straightforward approximation; of course it is possible to design the gates in order to obtain $a \in \mathbb{N}$, but generally the delay of such gates is not optimal.

Our intention here is to consider the general case (a may be any nonnegative number). Subsequently, we shall consider two different models. The latter will be more precise than the former, but we shall see that they give approximately the same results.

In this paper, we shall assume that only restoring logic is used in the design of the carry path of the adders. The carry propagation into the blocks (*ripple propagation*) is linear. The time needed to pass through N consecutive cells is $T = k_1 \cdot N$, where k_1 is the delay of one cell.

Concerning the carry propagation around the blocks (*skip propagation*), we shall consider the two following models.

Model 1: The skip propagation is linear, and depends only on the *number* p of skipped blocks, and not on their *size*. Thus, there exists a constant k_2 such that

$$T' = k_2 \cdot p \text{ is the time needed to skip over } p \text{ blocks.}$$

Model 2: The skip propagation depends on the *size* of the skipped blocks. This case occurs in MOS technologies, where delays are a function of the length of the interconnection lines. Let us consider a block of N cells. The path taken by the carry in order to skip over the block is a line in series with a logic gate. The delay of such a path is (see [16, p. 133])

$$T' = t_p + \frac{rc}{2} l^2$$

t_p is equal to the delay due to the logic gate, r and c are the resistance and the capacitance per unit length, l is the length of the line (nearly the length of the block). Thus, we may write

$$T' = k_2 + k_3 N^2$$

with

$$k_2 = t_p \text{ and } k_3 = \frac{rc}{2}\left(\frac{l}{N}\right)^2$$

l/N represents the width of the basic cell. With two-metal layer technologies, the interconnection line may be implemented in one of the metal layers. In this case, k_2 is much greater than k_3 (see [16, p. 136]) with a 2-Alu CMOS technology, the order of magnitude of the ratio k_2/k_3 is approximately 10^5); thus, Model 2 is equivalent to Model 1.

Then, we may consider the building of a carry-skip adder with *two levels of skips*, applying the carry-skip technique to the blocks themselves. The great advantage of such a device is that it only involves a small modification of the one-level carry skip, thanks to its modularity.

At the end of this paper, we shall present the design of a 128 bit carry-skip adder with two levels of skips, built in a 2 μm-gate CMOS technology, with two metal layers. This adder has been implemented in an arithmetic coprocessor which computes elementary functions [4].

II. Definitions

A carry-skip adder with p blocks will be represented by an array L of integers such that $L(i)$ is the length of the block numbered i. We shall note cleft(L) the index of the first block (i.e., cleft(L) = min $\{i | L(i) \neq 0\}$), and cright(L) the index of the last one (cright(L) = max $\{i | L(i) \neq 0\}$. A natural choice would have been to number the blocks starting from a constant value (cleft(L) = 1 for instance), but our notation will simplify the algorithm "transf" presented below. We shall represent such an adder by p adjacent columns of unit squares, with $L(i)$ squares for the column i (Fig. 1).

III. Study of Model 1

We shall introduce the following notations.

If A is a set, $|A|$ will be the number of elements.

$$N \text{ (number of bits of the adder)} = \sum_{i=c\,\text{left}(L)}^{c\,\text{right}(L)} L(i)$$

$$a = \frac{k_2}{k_1}$$

$$b_r(L) = \max_{i \in [c\,\text{left}(L), c\,\text{right}(L)]} \{L(i) + a(i+1)\}$$

$$b_l(L) = \max_{i \in [c\,\text{left}(L), c\,\text{right}(L)]} \{L(i) - a \cdot i\}.$$

$D_l(L)$ is the graph of the line $y = a \cdot x + b_l(L)$. This line is the lowest line of slope a which is above the columns-representation of L.

$D_r(L)$ is the graph of the line $y = -a \cdot x + b_r(L)$. This line is the lowest line of slope $-a$ which is above the columns-representation of L.

$$S(L) = \{i \in [\text{cleft } (L), \text{ cright } (L)] | L(i) = a \cdot i + b_l(L) \text{ OR } L(i) = -a \cdot (i+1) + b_r(L)\}$$

$$S_{\max}(L) = |S(L)|$$

$S(L)$ represents the set of the integers i such that column i is adjacent to $D_l(L)$ or $D_r(L)$.

$$H(L) = \{i \in [\text{cleft } (L) - 1, \text{ cright } (L) + 1] | (L(i) + 1 < a \cdot i + b_l(L)) \text{ AND } (L(i) + 1 < -a \cdot (i+1) + b_r(L))\}$$

$$H_{\max}(L) = |H(L)|.$$

($H(L)$ represents the set of the "holes" of the scheme, i.e., the set of the integers $i \in$ [cleft(L) − 1, cright(L) + 1] such that if $L(i)$ is replaced by $L(i) + 1$, then the ith column is still strictly included in the triangle defined by $D_r(L)$, $D_l(L)$ and the line $y = 0$. We illustrate the preceding notations in Fig. 2.

Let us consider the following problem: a carry generated at the beginning of the block numbered i skips p blocks, and terminates at the end of the block numbered $i + p + 1$. The propagation time obtained is $T_{i,p}(L) = k_l \cdot (L(i) + L(i + p + 1)) + k_2 p$.

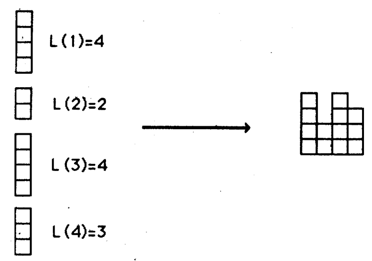


Fig. 1. Columns-representation of a carry-skip adder.

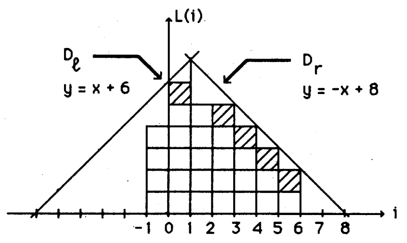


Fig. 2. We have
$a = 1$
c left$(L) = -$, c right$(L) = 5$
$L(-1) = 4, L(0) = 6, L(1) = L(2) = 5, L(3) = 4, L(4) = 3, L(5) = 2$
$b_l(L) = 6, b_r(L) = 8$
$S(L) = \{0, 2, 3, 4, 5\}$, $S\max(L) = 5$, $H(L) = \{-2\}$, $H\max(L) = 1$
The elements of $S(L)$ are darkened.

Since we have

$$L(i) \le a \cdot i + b_l(L)$$

$$L(i+p+1) \le -a \cdot (i+p+2) + b_r(L)$$

we deduce

$$\begin{aligned} T_{i,p}(L) &\le k_1 \cdot (a \cdot i + b_l(L) - a \cdot (i+p+2) + b_r(L)) + k_2 p \\ &= k_1 \cdot (b_l(L) + b_r(L) - a \cdot p - 2 \cdot a) + k_2 p \\ &= k_2 \cdot (b_l(L) + b_r(L) - 2) = T(L). \end{aligned}$$

(This propagation time $T(L)$ is achieved if column i is adjacent to D_l and column $i = p + 1$ is adjacent to D_r.)

Our purpose is to minimize T which is equivalent to minimizing $b_l(L) + b_r(L)$.

Thus, our problem is reduced to a geometric problem: we have to minimize the height of the triangle defined by $D_r(L)$, $D_l(L)$, and the line $y = 0$. It is worth noting that for large values of N we cannot solve this problem efficiently by examining all the possible schemes L; since the total number of schemes with N bits is equal to 2^N.

Now, we shall introduce the concept of H scheme, which is not always optimal, but which gives efficient solutions.

The H Schemes

Definition 1: A scheme L is an *H scheme* if the corresponding triangle contains no holes, i.e., if $H(L) = \emptyset$.

Theorem 1: For each value of $N \ge 1$ and $k_1, k_2 \in \mathbb{R}$, $k_1, k_2 > 0$, there exists an H scheme.

Proof: The proof is given by the following algorithm, *Transf*, which transforms any scheme L_0 into an H scheme. The basic idea of this algorithm is to put *adjacent squares* (elements of S) into the holes, until no hole is left.

```
Transf:
(We suppose that $S(L)(i)$ represents the element numbered $i$ of $S(L)$).
(The same notation is chosen for $H(L)$).
(Any order of numeration may be chosen)

While ($H$max $(L) > 0$) do
  begin
    min := min {$H$max $(L)$, $S$max $(L)$};
    for $j$ := 1 to min do
      begin
        $L(S(L)(j)) := L(S(L)(j)) - 1$;
        $L(H(L)(j)) := L(H(L)(j)) + 1$;
      end;
    Compute (cleft$(L)$,cright$(L)$,$b_l(L)$,$b_r(L)$,$S(L)$,
        $S$max$(L)$,$H(L)$,$H$max$(L)$)
end.
```

This algorithm terminates and gives an H scheme because Hmax decreases strictly at each step. Since this algorithm gives decreasing successive values, we deduce that *at least one optimal scheme is an H scheme;* this result is obtained by starting with an initial solution L_0 which is optimal.

Theorem 2: If L is an H scheme with maximal propagation time $T(L)$, and if the maximal propagation time of an optimal scheme is T^* (with the same number N of bits) then

$$T^* \le T(L) < T^* + 2k_2.$$

This theorem shows that the algorithm *transf* always gives convenient configurations of blocks for designing carry-skip adders.

Proof of Theorem 2: Let $F(n, L)$ (the nth "floor" of L) be the number of columns i such that $L(i) \ge n$. For instance, with the example presented in Fig. 2 $F(1, L) = F(2, L) = 7$ and $F(5, L) = 3$.

Let $\lambda(n, L) = (b_r(L) + b_l(L) - 2n)/a$. ($\lambda(n, L)$ is the distance between the two points of $D_r(L)$ and $D_l(L)$ of ordinate n). We have $F(n, L) \le \lambda(n, L)$ (Fig. 3).

Lemma: If L is an H scheme, then $F(n, L) \ge \lambda(n, L) - 2$.

Proof of the lemma: Let

$$\text{cleft } (L, n) = \min \{i | L(i) \ge n\}$$

$$\text{cright } (L, n) = \max \{i | L(i) \ge n\}.$$

(We have cright(L) = cright$(L, 1)$ and cleft(L) = cleft$(L, 1)$.) L is an H scheme, thus, $H(L) \ne \emptyset$.

Thus,

$$L\,[\text{cleft } (L, n) - 1] + 1 \ge a \cdot [\text{cleft } (L, n) - 1] + b_l(L) \quad \text{(i)}$$

$$L\,[\text{cright } (L, n) + 1] + 1 \ge -a \cdot [\text{cright } (L, n) + 2] + b_r(L) \quad \text{(ii)}$$

Thus, adding (i) and (ii)

$$2n \ge a\,[\text{cleft } (L, n) - \text{cright } (L, n)] + b_l(L) + b_r(L) - 3a \quad \text{(iii)}$$

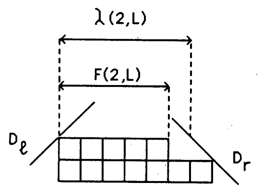


Fig. 3. Notations $F(n, L)$ and $\lambda(n, L)$.

since $L[\text{cleft}(l, n) - 1] \le n - 1$ and $L[\text{cright}(L, n) + 1] \le n - 1$.

By (iii):

$$[\text{cright}\,(L, n) - \text{cleft}\,(L, n) + 1] \ge [b_l(L) + b_r(L) - 2a - 2n]/a.$$

Thus, $F(n, L) \ge \lambda(n, L) - 2$. *The lemma is proved.*

Proof of Theorem 2: Let L be an H scheme, and L' be a better scheme such that $T(L') \le T(L) - 2k_2$. For every n, we have $\lambda(n, L') \le \lambda(n, L) - 2$. Thus, for every n, we have $F(n, L') \le F(n, L)$. Since there is at least one n such that $L[c\,\text{left}(n) - 1] = a[\text{cleft}(n) - 1] + b_l(L)(S \neq \varnothing)$, we can prove that there is at least one n such that $F(n, L) > \lambda(n, L) - 2$, which implies $F(n, L') < F(n, L)$.

Thus, $\Sigma F(n, L') < \Sigma F(n, L) = N$. There is no scheme L' such that $T(L') \le T(L) - 2k_2$. *Thus, Theorem 2 is proved.*

IV. Study of Model 2

In order to find convenient solutions for Model 2, we shall replace our problem by a continuous problem:

Let f be a continuous function such that $f(i) = L(i)$ for every i (see Fig. 4). As in the preceding study, we have to minimize the longest life of a carry. Let us suppose that a carry is generated at the beginning of block i, and that the following carry is generated at the end of block $j (j > i)$. Following the assumptions of Model 2, the life of the carry is

$$k_1(L_i + L_j) + k_2(j - i - 1) + k_3(L_i^2 + L_{i+1}^2 + \cdots + L_j^2).$$

A good approximation of $L_i^2 + L_{i+1}^2 + \cdots + L_j^2$ is $\int_i^j f^2(t)\,dt$.

Thus, our "continuous" problem (P) is to find a *continuous function* f of *finite support* S_f such that f minimizes the quantity

$$T(f) = \max_{\substack{x,y \in S_f \\ x<y}} \Phi_f(x, y)$$

where

$$\Phi_f(x, y) = k_1[f(x) + f(y)] + k_2(y - x) + k_3 \int_x^y f^2(t)\,dt.$$

If we find a family of functions f such that $\Phi_f(x, y)$ does not depend from x and y, thus we can use the same methodology as for Model 1: we have to find in this family the function f which minimizes Φ_f and such that a configuration of N unit squares disposed like in Fig. 1 can be placed under the curve $y = f(x)$. Obviously, similar concepts of *adjacent* squares and *holes* can be defined, and thus, an algorithm similar to *transf* can be given.

First, let us search a function f such that $\Phi_f(0, y)$, $y \in S_f$, $y > 0$, is *constant*. Let us suppose that f has a continuous

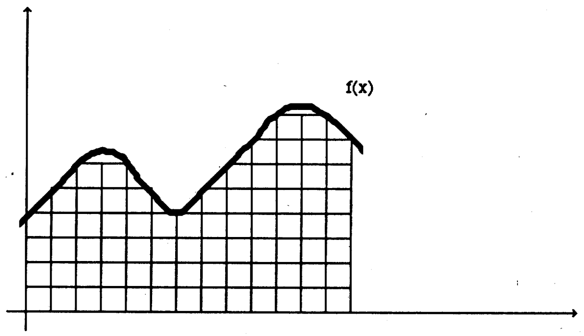


Fig. 4. Function f.

derivative f, we deduce

$$k_1 f'(y) + k_2 + k_3 f^2(y) = 0 \qquad (y \in S_f, y > 0). \qquad \text{(E)}$$

(E) is equivalent to

$$\frac{f'(y)}{\frac{k_3}{k_2}(f(y))^2 + 1} = -\frac{k_2}{k_1}.$$

Thus, the solutions of (E) are

$$f(y) = \sqrt{\frac{k_2}{k_3}} \tan\left(\frac{-y\sqrt{k_2 k_3}}{k_1} + C\right) \qquad C \text{ real.}$$

Now, if we search a function f such that $\Phi_f(x, 0)$ $(x < 0, x \in S_f)$, we obtain

$$f(x) = \sqrt{\frac{k_2}{k_3}} \tan\left(\frac{x\sqrt{k_2 k_3}}{k_1} + C\right) \qquad C \text{ real.}$$

If we examine these two last results, we deduce that a *continuous* function f such that $\Phi_f(x, y) =$ constant *must* verify

$$f(x) = \sqrt{\frac{k_2}{k_3}} \tan\left\{-\frac{\sqrt{k_2 k_3}}{k_1}|x| + C\right\} \qquad C \text{ real.}$$

on the domain

$$S_f = \left[-\frac{Ck_1}{\sqrt{k_2 k_3}}; +\frac{Ck_1}{\sqrt{k_2 k_3}}\right].$$

Now, let us prove that such a function f verifies $\Phi_f(x, y) =$ constant.

$$\begin{aligned}
\Phi_f(x, y) &= k_1[f(x) + f(y)] + k_2(y - x) + k_3 \int_x^y f^2(t)\,dt \\
&= k_1[f(x) + f(0) + k_2(0 - x) + k_3 \int_x^0 f^2(t)\,dt \\
&\quad + k_1[f(0) + f(y)] + k_2(y - 0) + k_3 \int_0^y f^2(t)\,dt \\
&\quad - 2k_1 f(0) \\
&= \Phi_f(x, 0) + \Phi_f(0, y) - 2k_1 f(0). \qquad (1)
\end{aligned}$$

Let us denote K^- the number equal to $\Phi_f(u, 0)$ for every $u \le 0$, and K^+ the number equal to $\Phi_f(0, v)$ for every $v \ge 0$. We have

$$\Phi_f(0, 0) = K^-$$

$$\Phi_f(0, 0) = K^+.$$

Thus, $K^- = K^+ = \Phi_f(0, 0) = K$.

Since we have

$$\Phi_f(0, 0) = 2k_1 f(0) + k_2(0-0) + k_3 \int_0^0 f^2(t)\, dt = 2k_1 f(0).$$

We deduce from (1) that for every (x, y), $\Phi_f(x, y)$ is equal to $2k_1 f(0)$; thus, $\Phi_f(x, y)$ is constant.

The function f associated with the solution of the problem satisfies

$$\int_{S_f} f(t)\, dt \approx N.$$

Thus,

$$N \approx 2\sqrt{\frac{k_2}{k_3}} \int_0^{Ck_1/\sqrt{k_2k_3}} \tan\left(C - x\frac{\sqrt{k_2k_3}}{k_1}\right) dx$$

$$= \frac{2k_1}{k_3}\int_0^C \tan u\, du \text{ with } u = C - x\frac{\sqrt{k_2k_3}}{k_1}.$$

Thus, we can approximate the constant C of the solution, and obtain

$$C \approx \arctan\sqrt{e^{(k_3/k_1)N} - 1}.$$

Thus, function f of the solution is

$$f(y) = \sqrt{\frac{k_2}{k_3}} \tan\left(-\frac{\sqrt{k_2k_3}}{k_1}|y| + \arctan\sqrt{e^{(k_3/k_1)N} - 1}\right).$$

Fig. 5, 6, and 7 present the graph of f for some values of k_1, k_2, and k_3 (N = 32 bits). If $k_3 \ll k_1$, then

$$e^{(k_3/k_1)N} - 1 \approx \frac{k_3}{k_1} N$$

thus,

$$\arctan\sqrt{e^{(k_3/k_1)N} - 1} \approx \sqrt{\frac{k_3}{k_1} N}$$

thus,

$$-\frac{\sqrt{k_2k_3}}{k_1}|y| + \arctan\sqrt{e^{(k_3/k_1)N} - 1}$$

$$\approx \sqrt{k_3}\left(-\frac{\sqrt{k_2}}{k_1}|y| + \sqrt{\frac{N}{k_1}}\right)$$

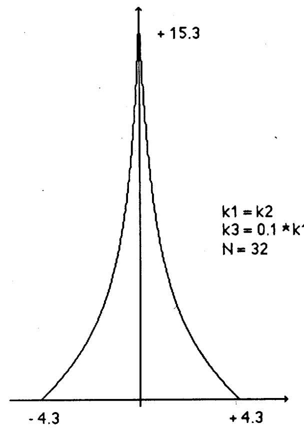


Fig. 5. $k_1 = k_2$; $k_3 = k_1/10$.

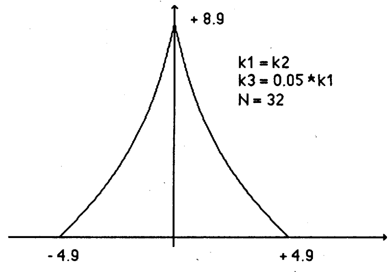


Fig. 6. $k_1 = k_2$; $k_3 = k_1/20$.

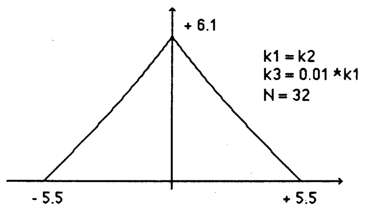


Fig. 7. $k_1 = k_2$; $k_3 = k_1/100$.

thus,

$$\sqrt{\frac{k_2}{k_3}} \tan\left(-\frac{\sqrt{k_2k_3}}{k_1}|y| + \arctan\sqrt{e^{(k_3/k_1)N} - 1}\right)$$

$$\approx -\sqrt{\frac{k_2}{k_3}}\sqrt{k_3}\left(-\frac{\sqrt{k_2}}{k_1}|y| + \sqrt{\frac{N}{k_1}}\right)$$

thus,

$$f(y) \approx -\frac{k_2}{k_1}|y| + \sqrt{\frac{k_2 N}{k_1}}.$$

And Model 2 is equivalent to Model 1, because f is a linear function. Fig. 7 shows that Model 2 becomes equivalent to Model 1 for $k_3 = k_1/100$. Since, with the technology that we used (2-Alu CMOS), $k_3 \approx 5.10^{-5}k_1$, Model 1 is valid; thus we did not apply an algorithm similar to *transf* to Model 2.

V. Two-Level Carry-Skip Adders

As said in the Introduction of this paper, the carry-skip technique may be applied recursively to the blocks themselves.

Thus, the skip chain will have two levels; following a terminology already used [10], we shall say that the adder is divided into *sections* (this division forms the first layer), the sections are themselves divided into *groups* of bits (this subdivision forms the second layer).

We shall assume that Model 1 is valid; thus,

- The time needed to pass through p *cells* included in a same *group* is proportional to p. Thus, this time is equal to $k_1 \cdot p$, where k_1 is the delay of one cell.
- The time needed to skip over q *groups* of the same *section* is proportional to q. Let us call k_2 the constant such that this time is equal to $k_2 \cdot q$.
- The time needed to skip over r *sections* of the adder is proportional to r. Let us call k_3 the constant such that this time is equal to $k_3 \cdot r$.

Now, let us build, as in Section II of this paper, a geometric problem associated with our optimization problem. Let us consider N parallelepipeds like that presented in Fig. 8 (N is the number of bits of the adder):

And let us consider the pyramid $P(x_h, y_h, z_h)$ of vertex (x_h, y_h, z_h) and of equation

$$\begin{cases} x+y+z \le x_h+y_h+z_h \\ x-y+z \le x_h-y_h+z_h \\ x+y-z \le x_h+y_h-z_h \\ x-y-z \le x_h-y_h-z_h \\ x \ge 0 \\ y \ge 0 \\ z \ge 0. \end{cases}$$

If we take four points A, B, C, D of the pyramid such that $x_A = x_B$ and $x_C = x_D$ ((x_A, y_A, z_A) are the coordinates of A, (x_B, y_B, z_B) are those of B, and so on), and if we compute the number

$$L = z_A + (y_B - y_A) + z_B + 2(x_D - x_B) + z_D + (y_D - y_C) + z_C.$$

We obtain $L = 4z_h$; L does not depend on A, B, C, and D.

But if we dispose (as in Figs. 9 and 10) the N parallelepipeds under the pyramid, such that each plane of equation $x =$ constant represents a *section,* which is similar to the representation of a carry-skip adder given in Fig. 1, L represents the *propagation time* of a carry generated in the section of ordinate x_A and relayed by the following carry generated in the section of ordinate x_B.

Therefore, our optimization problem is equivalent to a geometric problem similar to that of Section I of this paper: we have to minimize the height z_h of the lowest pyramid $P(x_h, y_h, z_h)$ which contains all the parallelepipeds. Thus, an algorithm similar to *transf* can be built.

Transf2:

(We start from an initial configuration of N parallelepipeds similar to that of Fig. 9)

```
stop := false;
while not stop do
    begin
    1. Compute the parameters $(x_h, y_h, z_h)$ of the lowest
       pyramid $P(x_h, y_h, z_h)$ which contains all the parallelepipeds.
    2. Detect the set $H$ of the ``holes'' of the configuration,
       i.e., the set of the columns of parallelepipeds
       such as if a parallelepiped is added to the column, it
       remains under the pyramid.
    3. Detect the set $S$ of the columns of parallelepipeds
       adjacent to the pyramid.
    4. If $H \neq \emptyset$ then
           begin
       $m := \min \{|S|, |H|\}$;
       move $m$ parallelepipeds from $m$ different
                  columns of $S$ to $m$ different columns of $H$
           end
       else stop := true
end.
```

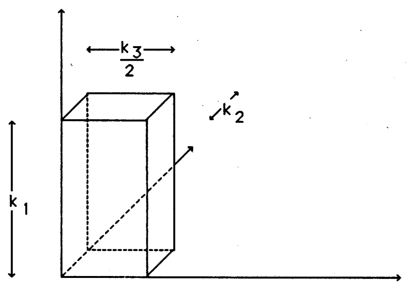


Fig. 8. The parallelepipeds.

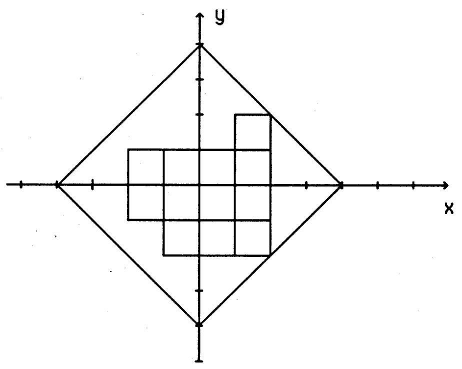


Fig. 9. Top view of the disposition.

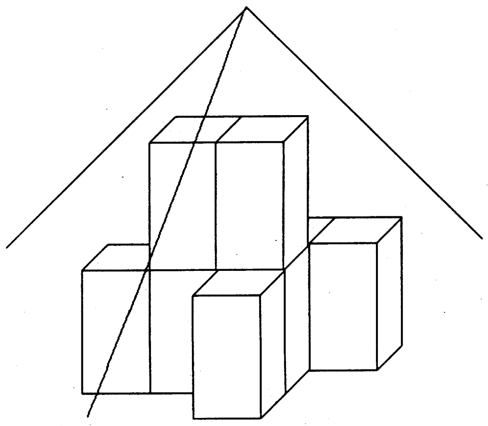


Fig. 10. The pyramid.

VI. Example of a Two-Level Carry-Skip Adder

We describe here the design of a 128 bit carry-skip adder included in an arithmetic coprocessor designed at our laboratory: the FELIN chip [4]. The technology used is a 2 μm gate CMOS with two metal layers, so the distribution follows Model 1. The skip chain has two levels; the sections form the first layer, and the groups form the second layer. In order to simplify the basis cells and to linearize and speed up the ripple propagation, we adopt the schemes shown in Fig. 11.

The XOR gates are designed as shown in Fig. 12.

The layout of these two cells is nearly the same. The tristate gate of the ripple-carry part of the circuit introduces a delay t_r. The groups are built by alternating the even and odd cells, and they always contain an even number of cells. So, the carry entering or leaving the groups is either C or its complement. In order to skip over a group, the carry has to pass through a logic gate which introduces a delay T, which has actually nearly twice the value t_r. This gate is as shown in Fig. 13. The pair of pass transistors at the output of the gate are actually included in a multiplexor. If the propagate signal of the section I is P_I, and the propagate signals of the groups of the section I are P_{jI} ($j \in \{1, \cdots, k\}$, where k is the number of groups in section I): the multiplexor is driven by a function of P_I and P_{kI}. The circuitry for the carry paths is presented in Fig. 14.

The sizing of the groups is guided by a straight line, whose slope is equal to $T/t_r = 2$. For the sections, the slope is 1. The distribution of the complete adder is given here:

2(242)(2442)(24642)(246642)(246642)(24642)(2442)(242)2.

The total propagation time of the carry in this 128 bit adder corresponds to the propagation through 12 gates of delay T. Due to the simplicity of these gates, T is within 3–5 ns. Thus, the computing time of this adder is around 50 ns.

We give here the distribution of a 66 bit two-level carry-skip adder:

(22)(242)(2442)(24642)(2442)(242)(22).

The total propagation time of the carry in this adder is about 40 ns ($9T$). We can see that the difference between the speed of both adders is small despite their difference in size, but differing results would be obtained with other slopes. Thus, it is difficult to make general comparisons between carry-skip techniques and other speedup techniques. However, for the case presented here (66 bit adder) a comparison is possible. In [3], a 64 bit carry-look ahead adder is described. If D is the delay of a gate for the technology being used, it gives a result in $13D$. The 66 bit carry-skip described previously completes it in

$$\begin{array}{rll} & 9T & \text{(total propagation time)} \\ + & T & \text{(computation of } p_i = a_i \oplus b_i) \\ + & 2T & \text{(computation of } P_{jI} \text{ (groups) and } P_I \text{ (sections))} \\ + & T & \text{(computation of } S_{63} = P_{63} \oplus C_{63}) \\ \hline & 13T & \end{array}$$

Actually, T corresponds to the propagation through two inverters. But D is the delay of gates with two inputs or more, so T and D are roughly equivalent. In [16, p. 181–183], Weste and Eshraghian give the delay ratio between an inverter and more complex gates. Hence, the 66 bit two-level carry-skip adder is comparable in speed to the CLA.

The 128 bit carry-skip adder presented here is highly modular. Its main component cells are

- the basis pair cells (odd and even) of the ripple part,
- dynamic NOR's for the computation of the propagate signals for the groups and the sections.

The whole adder has been assembled [14] automatically by a Pascal program. A part of the layout is given in Fig. 15. We can see that the carry-skip part about doubles the size of the basis ripple-carry adder.

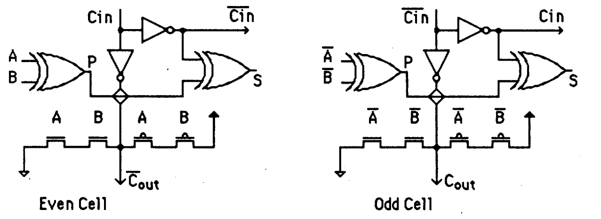


Fig. 11. The two kinds of ripple cells.

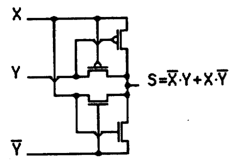


Fig. 12. The XOR gates.

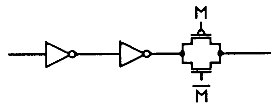


Fig. 13. The skip-over-groups gate.

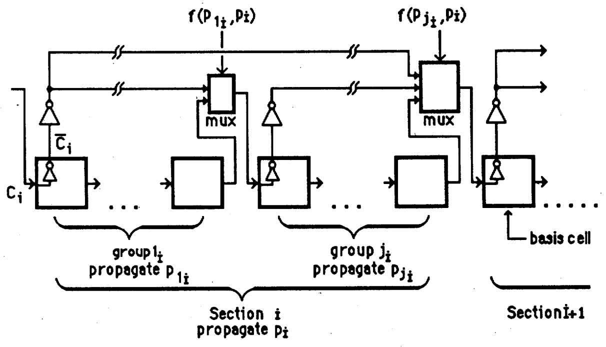


Fig. 14. Circuitry for the carry path.

VII. Conclusion

The carry-skip adders seem to be a good compromise between ripple-carry adders (very simple, but slow) and sophisticated adders (like carry-look ahead adders, for instance), which involve both large silicon areas and design problems. Since the gates of the skip part are very simple, their propagation time is interesting.

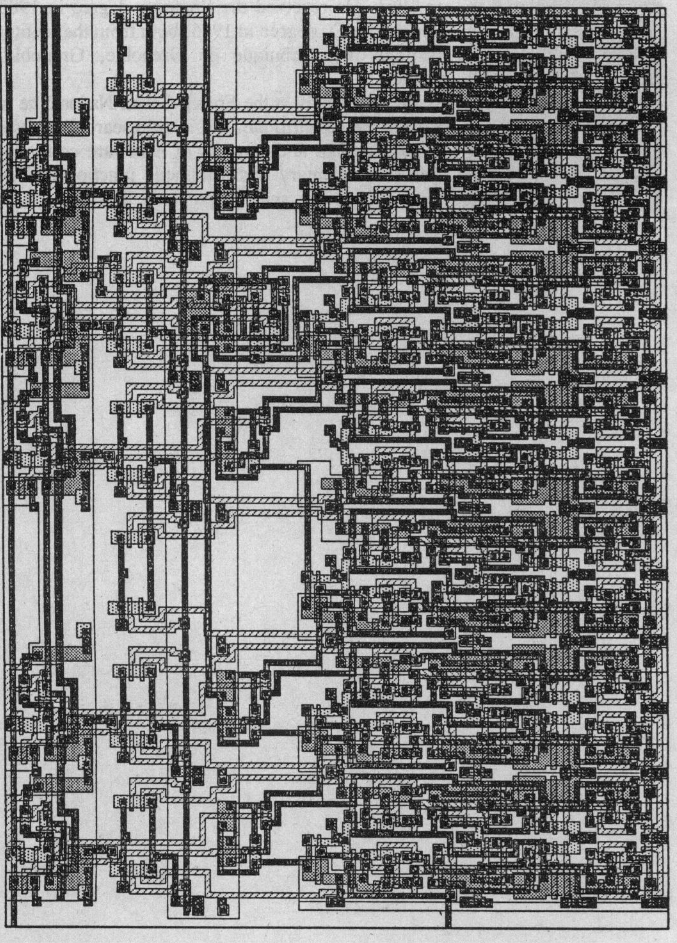


Fig. 15. A portion of the two-level carry-skip adder of FELIN. The upper aluminium layer is not shown.

N = 64 bits

a	L
0.4	1 1 1 2 2 3 3 3 4 4 5 5 5 4 4 4 3 3 2 2 1 1 1
0.45	1 1 2 2 3 3 4 4 5 5 5 4 4 4 3 3 2 2 1 1
0.55	1 1 2 2 3 4 4 5 5 6 6 5 4 4 3 3 2 2 1 1
0.65	1 2 2 3 4 4 5 6 6 6 5 5 4 3 3 2 2 1
0.75	1 1 2 3 4 4 5 6 7 6 6 5 4 4 3 2 1
0.85	1 2 3 4 5 5 6 7 7 6 5 4 3 3 2 1
0.95	1 2 3 4 4 5 6 7 7 6 5 4 4 3 2 1
1.0	1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
1.1	1 2 3 4 5 6 7 8 8 7 6 5 3 2 1
1.2	1 2 3 5 6 7 8 8 7 6 5 3 2 1
1.3	1 2 3 5 6 7 8 8 7 6 5 3 2 1

N = 128 bits

a	L
0.4	1 1 2 2 3 3 3 4 4 5 5 5 6 6 7 7 7 7 6 6 5 5 5 4 4 3 3 3 2 2 1 1
0.45	1 1 2 2 2 3 3 4 4 5 5 6 6 6 7 7 7 7 6 6 6 5 5 4 4 3 3 2 2 2 1 1
0.55	1 2 2 3 3 4 4 5 5 6 6 7 8 8 8 8 7 6 6 5 5 4 4 3 3 2 2 1
0.65	1 2 2 3 4 4 5 6 6 7 8 8 9 9 8 7 7 6 5 5 4 4 3 2 1
0.75	1 2 2 3 4 5 6 6 7 8 9 9 10 9 8 7 7 6 5 4 4 3 2 1
0.85	1 2 2 3 4 5 6 7 7 8 9 10 10 9 8 7 7 6 5 4 3 2 2 1
0.95	1 2 3 4 5 6 7 8 9 9 10 10 9 9 8 7 6 5 4 3 2 1
1.0	2 3 4 5 6 7 8 9 10 11 11 10 9 8 7 6 5 4 2 1
1.1	2 3 4 5 6 7 8 9 10 12 11 10 9 8 7 6 5 3 2 1
1.2	1 2 3 5 6 7 8 9 11 12 12 11 9 8 7 6 5 3 2 1
1.3	1 2 3 5 6 7 8 10 11 12 12 11 10 8 7 6 4 3 2

Fig. 16. Some configurations of blocks given by the algorithm *trans*.

We have given a way to find efficient carry-skip adders. The algorithm *transf*, presented here, was written in Pascal on a microcomputer (IBM PC), and gave us fine results for different values of N (from 32 to 1024 bits). Some of them, tabulated for different values of N and a, are given in Fig. 16.

Acknowledgment

We thank the referees for their suggestions, which helped us to improve the original manuscript.

References

[1] E. R. Barnes and V. G. Oklobdzija, ''Some optimal schemes for ALU implementation in VLSI technology,'' in *Proc. 7th Symp. Comput. Arithmetic.*

[2] A. Burks, H. H. Goldstine, and J. Von Neumann, ''Preliminary discussion of the logic design of an electronic computing instrument,'' Instit. Adv. Study, Princeton, NJ, 1946 (reprinted in C. G. Bell, *Computer Structures: Readings and Examples.* New York: McGraw-Hill, 1971).

[3] J. J. F. Cavanagh, *Digital Computer Arithmetic, Design and Implementation.* New York: McGraw-Hill, 1984.

[4] M. Cosnard, A. Guyot, B. Hochet, J. M. Muller, H. Ouaouicha, and E. Zysman, ''FELIN: An elementary function cruncher,'' *Comput. Computing, Proc. Symp. Future Trends Computing,* Grenoble, France, 1985, Masson, ed. New York: Wiley, 1986.

[5] J. B. Gosling, ''Review of high speed addition techniques,'' *Proc. IEEE,* vol. 118, pp. 29–35, Jan. 1971.

[6] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits.* Reading, MA: Addison-Wesley, 1985.

[7] K. Hwang, *Computer Arithmetic Principles, Architecture and Design.* New York: Wiley, 1979.

[8] M. Lehman, ''A comparative study of propagation speed-up circuits in binary arithmetic units,'' in *Information Processing.* Amsterdam, The Netherlands: Elsevier-North Holland, 1963, p. 671.

[9] M. Lehman and N.Burla, ''Skip techniques for high-speed carry propagation in binary arithmetic units,'' *IRE Trans. Electron. Comput.,* p. 691, Dec. 1961.

[10] S. Majerski, ''On determination of optimal distributions of carry skips in adders,'' *IEEE Trans. Electron. Comput.,* vol. EC-16, Feb. 1967.

[11] C. Mead and L. Conway, *Introduction to VLSI Systems.* Reading, MA: Addison-Wesley, 1980, p. 25.

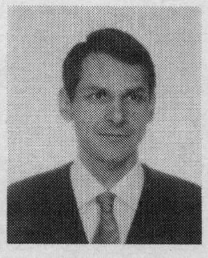
[12] S. Ong and D. E. Atkins, ''A comparison of ALU structures for VLSI technology,'' in *Proc. 6th Symp. Comput. Arithmetic,* Aarhus University, Aarhus, Denmark, June 1983.

[13] M. Pomper *et al.,* ''A 32 bit execution unit in an advanced NMOS technology,'' *IEEE J. Solid-State Circuits,* vol. SC-17, June 1982.

[14] J. P. Schoellkopf, ''Lubrick: A silicon assembler and its application to data-path design for FISCA,'' in *Proc. VSLI'83,* G. Anceau and E. J. Aas, Eds. Amsterdam, The Netherlands: Elsevier Science, IFIP, 1983, pp. 435–445.

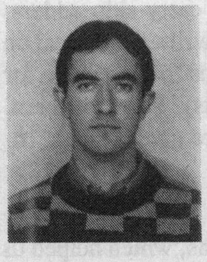
[15] N. R. Scott, *Computer Systems and Arithmetic.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

[16] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A Systems Perspective.* Reading, MA: Addison-Wesley, 1985.



Alain Guyot was born in Trifouilly-Les-Oies, France, in 1945. After teaching mathematics in high school, he received the Ph.D. degree in 1975 from the University of Grenoble, Grenoble, France.

He then joined the staff at ENSIMAG (Ecole National Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble) where he has been teaching computer architecture and VLSI design ever since, except for one year spent at Stanford University. He has helped starting the french MPC project in 1981.



Bertrand Hochet was born in France in 1961. He received the Engineer degree from Ecole Nationale Supérieure d'Electronique et de Radioélectricité de Grenoble, Grenoble, France, in 1984, and the Ph.D. degree from Institut National Polytechnique de Grenoble, Grenoble, France, in January 1987, on the architecture and design of an elementary functions processor.

His main research interests are computer architecture, electronics, and VLSI design.

Jean-Michel Muller was born in Grenoble, France, in 1961. He received the Engineer degree in 1983 and the Ph.D. degree in 1985, both from the Institut National Polytechnique de Grenoble, Grenoble, France.

He is presently at the French Centre National de la Recherche Scientifique. His main research interests include design and analysis of hardware computation of elementary and arithmetic functions, computer arithmetic, and computer architecture.