EFFICIENT SERIAL-PARALLEL ARRAYS FOR MULTIPLICATION AND ADDITION

L.Ciminiera and A.Serra

Dipartimento di Automatica e Informatica- Politecnico di Torino-
Torino, Italy

Abstract

Three new arrays for unsigned and signed multiplication, and for multiplication/addition are presented. It is assumed that the factors are axpressed in 2's complement, while the addend (in the latter array only) and the result are expressed in a redundant notation.
The arrays operate in serial-parallel way, since one factor is input in parallel, while the second factor and the addend (in the case of multiplication/addition) are entered digit by digit starting from the most significant one; the result is also produced serially with the most significant digit first.
Hence, the arithmetic unit presented is suitable to be used as basic block of special purpose processors performing functions such as non-recursive digital filtering, signal correlation and matrix multiplication. Indeed, they have the same speed improvements as other similar units using redundant representations for the result, with a cost equivalent to their counterparts based on full 2's complement representation.

1. Introduction

New opportunities for implementing special-purpose processors and for exploiting the potential advantage of redundant number systems are given by the recent advances of the integration technology.
Higher speed can be attained by using suitable redundant representations specially when they are used for arithmetic units where at least one input operand and the result come digit-by-digit.
Furthermore, serial-serial and serial-parallel units are often a good solution for implementing complex processors, since full parallel units are area and pin consuming.
Ammong others redundant number systems applied to serial arithmetic units, an important class is represented by signed digit representation[1] and the on-line algorithms for basic and complex arithmetic operations[2-4], developed for this representation.
The advantages of these algoritms over the conventional ones have been shown in the literature[5-6]; the major reason for this higher speed lies in the ability of on-line units to produce the most significant digit first. However, the implementation of these units is not as simple as for the conventional ones[7].
In the field of non-redundant number representation too, the arithmetic units for serial-parallel and serial-serial operations have been extensively studied in the past. In particular, several works have appeared on the implementation of serial-parallel multipliers[8-9], which basically differs in the sequence used for generating the elementary products. The resulting design is very simple, since it is based on an adder with the some input logic for addition/subtraction and elementary product generation.
The arrays presented in this paper try to combine the simplicity of the design of the arithmetic units based on non-redundant number systems with the speed advantages offered by the redundant ones, for an important class of application, at least.
The basic idea is that there is an important class of algorithms such as convolution and matrix multiplication, which do not require the multiplication of any intermediate result. Hence, it is not necessary to use a multiplier accepting factors in redundant form. Since this is one of the most important factors causing the non simple implementations of arithmetic units based on uniform redundant representation of inputs and outputs, a careful non-uniform choice of the different representations produces the two characteristies mentioned above.
Three arrays of this kind are presented in this paper: a serial- parallel multiplier for unsigned numbers and two units for serial-parallel signed multiplication and

CH2146-9/85/0000/0028$01.00

multiplication/addition. The two input factors are always represented in 2's complement notation, while the result is represented by using 2 bits per digit in the non assimilated carry-sum form[11]. For the multiplication addition unit, the addend number is input serially and it is represented in the same way as the result. All the units presented generate the output digits starting with the most significant one; hence, they retain all the speed advantages of other units based on different redundant representation.

In section 2, the algorithm and the array for unsigned multiplication are introduced. In section 3, the array for signed multiplication is obtained as a modification of the previons one; furthermore, the unit for signed multiplication/addition is discussed. In section 4, speed and cost comparisons with both conventional and on-line implementations are shown.

2. Unsigned number multiplication

The array for unsigned number multiplication is presented first, because it is the kernel of the other units able to process signed numbers, which will be presented in the following sections.

The representation assumed for the operands is the following one:

$$X = \sum_{i=0}^{n-1} x_i 2^i \quad , \quad Y = \sum_{i=0}^{n-1} y_i 2^i \qquad (1)$$

The representation of the result, P, is not the conventional binary representation, bacause the need for obtaining the most significant digit first cannot be met, if non-redundant representation is used. In our case, the redundant representation chosen gives the result P, by means of a pair of numbers, each represented in the conventional binary form, whose sum gives the value of the result. Formally, the result P is given by:

$$P = \sum_{i=0}^{2n-1} (c_i + s_i) 2^i = \sum_{i=0}^{2n-1} c_i 2^i + \sum_{i=0}^{2n-1} s_i 2^i = C+S \qquad (2)$$

Since the result is expressed in a redundant form, each pair of bits (c_i, s_i) will be referred to as a digit of the result.

Multiplication algorithm

We assume that all the bits of the operand Y are available when the multiplication begins, while only the most significant bit of the operand X is available.

The other bits of X become available one at a time in descending order of significance. Hence the algorithm is organized in successive steps, with a new bit of X available at the beginning of each step. In formulas, the multiplication algorithm is given by the following recursive procedure:

$$R_0 = 0 \qquad (3.a)$$

$$R_i = R_{i-1} + x_{n-i} 2^{n-i} Y \qquad (3.b)$$

$$P = R_n \qquad (3.c)$$

The novelty of the algorithm presented lies in the method used for performing the addition of the induction step shown in (3.b), and in the representation used for the values of R_i. We choose to represent the partial sum R_i in the following form:

$$R_i = M_i + Q_i = M_{i-1} + Q_{i-1} + x_{n-i} 2^{n-i} Y \qquad (4)$$

$$M_i = \sum_{k=2n+2-i}^{2n} (c_k + s_k) 2^k \qquad (5)$$

$$Q_i = q'_{i,2n+1-i} + \sum_{k=0}^{2n-i} (q'_{i,k} + q''_{i,k}) 2^k \qquad (6)$$

Hence, each step of the algorithm is reduced to the computation of M_i and Q_i, starting from the values of M_{i-1}, Q_{i-1} and $x_{n-i}Y$. From equation (4) and (5), we derive the following relation:

$$M_i = M_{i-1} + (c_{2n+2-i} + s_{2n+2-i}) 2^{2n+2-i} \qquad (7)$$

This equation simply states that M_i is the collection of the most significant bits of the product, with a new digit produced at the end of each step of the algorithm. The real computation is performed to obtain Q_i from Q_{i-1} and $x_{n-i}Y$, and to obtain the new pair of c and s bits. From (4) and (7) we obtain:

$$Q_i + (c_{2n+2-i} + s_{2n+2-i}) 2^{2n+2-i} = Q_{i-1} + x_{n-i} 2^{n-i} Y \qquad (8)$$

$$Q_i + x_{n-i} 2^{n-i} Y = \sum_{j=0}^{n-i} (q'_{i-1,j} + q''_{i-1,j}) 2^j +$$
$$+ \sum_{k=n-i+1}^{2n-i} (q'_{i-1,k} + q''_{i-1,k} + x_{n-i} y_{n-i-k}) 2^k +$$
$$+ (q'_{i-1,2n-i+1} + q''_{i-1,2n-i+1}) 2^{2n-i+1} +$$
$$+ q'_{i-1,2n+2-i} 2^{2n+2-i} \qquad (9)$$

So far, only formula manipulations have been carried out; at this point, it is necessary to perform some real arithmetic operation, which will transform the current representation of the numbers into a new one. The basic arithmetic function used in our algorithm is

that performed by a full adder, which is able to perform the following operation:

$$\alpha 2^1 + \beta\, 2^0 = (a+b+c)\, 2^0 \qquad (10)$$

where a, b and c are the input bits, while and β, α are the output ones.
By applying the transformation to each term of the second summation in the right end of (9), we obtain:

$$\sum_{j=0}^{n-i} (q'_{i-1,j}+q''_{i-1,j})2^j + \sum_{j=n+1-I}^{2n-i} (q'_{i,j}+q''_{i,j})2^j + \\ +2^{2n+1-i}(q'_{i-1,2n+1-i}+q''_{2n-i+1}+\alpha_{2n-i}) + \\ +q'_{i-1,2n+2-i}\,2^{2n+2-i} \qquad (11)$$

where α_{2n-i} is the most significant bit produced by the trasformation (10) applyed to the term with j = 2n-1. By applyng (10) once again to the third addend of (11), we get:

$$\sum_{j=0}^{n-i}(q'_{i-1,j}+q''_{i-1,j})2^j + \sum_{j=n-i+1}^{2n-i}(q'_{i,j}+q''_{i,j})2^j + \\ +q'_{i,2n+1-i}\,2^{2n+1-i} + (c_{2n+2-i}+s_{2n+2-i})2^{2n+2-i} \qquad (12)$$

In (12) the new digit of the most significant part of the result has been expressed in order to underline that it is the new digit to be included in M_i. From (12) and (8), it is possible to obtain Q_i, whose representation complies with (6), hence at the next step the same trasformations can be made to compute Q_{i+1}.
At the beginning of the operation the values of R_o and Q_o are set to 0.
At the end of step n, both R_n and Q_n are represented by at most 2 bits per digit, and their concatenation gives the final result P, R_n will contain the most significant part of the representation of P, and Q the least significant.

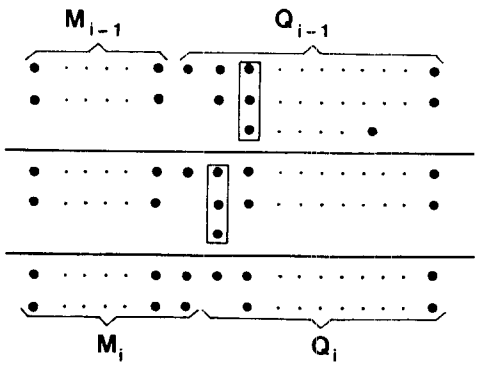


Fig. 1. Dot diagram of the arithmetic function required for performing the recursion step of the unsigned multiplication algorithm.

Array

The algorithm presented above requires arithmetic operation only for transforming (9) into (11), and then (11) into (12). First, it is worth noting that $q'_{i,j} = q'_{i-1,j}$ and $q''_{i,j} = q''_{i-1,j}$ for $0 \le j \le n-1$, hence no operation is performed on these least significant digits and they are all set to 0, given the initial condition; furthermore, the digits of R_i are not involved in any computation. Therefore, the implementation of the algorithm presented requires an arithmetic unit to perform the operations described by the dot diagram of Fig. 1.
The array in Fig. 2 implements that operation for a parallel operand Y rapresented by 4 bits. The input x is the serial input for the second operand, while on the outputs c and s the digits of R are produced, serially, one per clock cycle, starting from the most significant one. The memory elements, represented by bars in Fig. 2, store the values of the digits of Q_i; hence, after n clock cycles, R_n has been output serially from the outlets c and s, while the value of Q_n is still stored within the array. In order to output the full representation of the result, the array operations have to be continued until n more clock cycles have been performed; in this second phase, the serial input x must be set to 0.
The dots shown in Fig. 2 mark the position of two optional memory elements, which lead to two different implementations.
If they are not included, each clock cycle of the array cannot be shorter than twice the delay of a single full adder plus the delay of one memory elements and one AND gate. If the

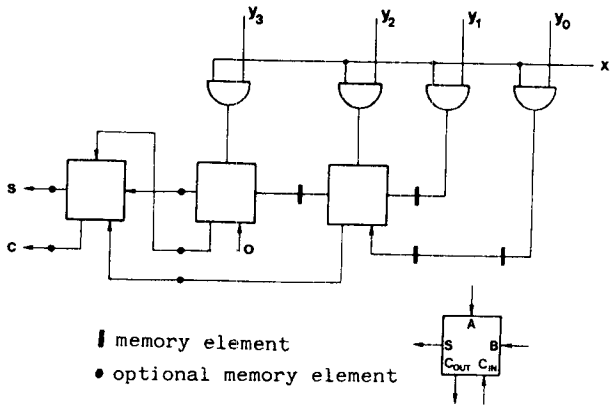


Fig. 2. Array for unsigned multiplication, with a parallel factor of 4 bits.

memory elements are introduced, the cycle time can be shortened by the delay of a single full adder, however an additional delay of one clock cycle will be required before the most significant digit of the result is produced. A third implementation option is obtained by substituting the two leftmost full adders in Fig. 2 with a fast 2x2 bit binary adder; in this case both the clock cycle and the latency time are kept short, while the complexity of the implementation is slightly increased.
Finally, if the full adders are modified so that they include the function performed by the AND gates, the cycle time can be shortened, with an increasing complexity of the implementation.

3. Signed multiplication/addition

The 2's complement is the most widely used representation of signed numbers in the digital system. Hence, this representation is assumed for the input factors of the array presented in this section.
The two factors are given in the following form:

$$X=\sum_{j=0}^{n-2} x_j 2^j - x_{n-1} 2^{n-1} \quad , \quad Y=\sum_{j=0}^{m-2} y_j 2^j - y_{m-1} 2^{m-1} \tag{13}$$

while the result P is produced in the following form:

$$P = (c_{m+n-1}+s_{m+n-1})2^{m+n-1} + \sum_{j=0}^{m+n-2} (c_j+s_j)2^j \tag{14}$$

In order to take advantage of the algorithm developed in the previous section, we must use a 2's complement multiplication algorithm based solely on the addition of positive numbers as the unsigned number multiplication algorithm.
This characteristic holds for the Baugh and Wooley algorithm[12], which is now briefly recalled.
Given the representations in (13), the product is given by:

$$\begin{aligned} XY = {} & x_{n-1}y_{m-1}2^{m+n-2} - x_{n-1}2^{n-1}\sum_{j=0}^{m-2} y_j 2^j - \\ & - y_{m-1}2^{m-1}\sum_{j=0}^{n-1} x_j 2^j + \sum_{j=0}^{n-2} x_j 2^j \sum_{k=0}^{m-2} y_k 2^k = 2^{m+n-1} + \\ & + (1+\overline{x}_{n-1}\overline{y}_{m-1})2^{m+n-2} + \sum_{j=0}^{m-2} x_{n-1}\overline{y}_j 2^j + \sum_{k=0}^{n-2} y_{m-1}\overline{x}_k 2^k + \\ & + \sum_{j=0}^{n-2} x_j 2^j \sum_{k=0}^{m-2} y_k 2^k + y_{m-1}2^{n-1} + x_{n-1}2^{m-1} \end{aligned} \tag{15}$$

The additions of (15) can be performed, by means of a recursive procedure, where a new bit of X is entered at each step. The algorithm is as follows.

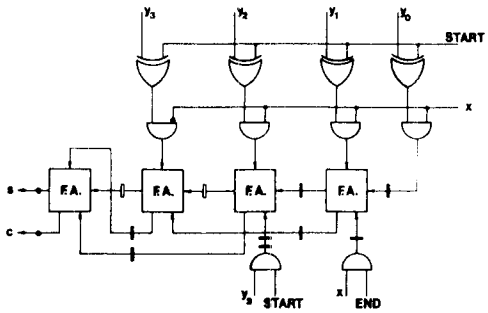
$$R_0 = 3 \cdot 2^{m+n-2} \tag{16.a}$$

$$R_1 = R_0 + \overline{x}_{n-1}\overline{y}_{m-1}2^{m+n-2} + x_{n-1}\sum_{j=0}^{m-1}\overline{y}_j 2^{n-1+j} + y_{m-1}2^{n-1} \tag{16.b}$$

$$R_i = R_{i-1} + \overline{x}_{n-i}y_{m-1}2^{m+n-i-1} + x_{n-i}\sum_{j=0}^{m-2} y_j 2^{n-i+j} \tag{16.c}$$

$$R_i = R_{i-1} + \overline{x}_{n-i}y_{m-1}2^{m+n-i-1} + x_{n-i}\sum_{j=0}^{m-2} y_j 2^{n-i+j} + x_{n-1}2^{m-1} \qquad \begin{matrix} i\neq n-m+1 \\ i=n-m+1 \end{matrix} \tag{16.d}$$

It is worth noting that this algorithm is quite similar to the algorithm presented in section 2. They differ only in the third operand at the right end of the first and last step, and in some complementations of the operand bits. Hence it is possible to perform the same manipulation of the formulas (16) as those of the previons section, assuming that the values of R_i are represented as in (4), (5) and (6).
The resulting array is shown in Fig. 3. It is essentially the same as that of Fig. 2, with a line of exclusive-or gates used to perform the complementations of the factor bits required by the formulas (16). Furthermore, one extra full adder is used to accomodate the addition of the last term in the first step and in the step n-m+1 of the algorithm. Since the initial step is slightly different, an additional signal, START, has been introduced in Fig. 3; it is set to 1 when the first digit of the serial operand is applied to the X input, then START is reset when the second digit is input. Hence, using



▯ memory element (in.value 1)
▮ memory element (in.value 0)
▮ optional memory element

Fig. 3. Array for signed multiplication, with parallel factor of 4 bits.

START, all the different complementations and additions required by the first step are generated. Another signal, END, is activated to add the term x_{n-1}, required in step n-m+1.
The same tradeoffs shown in section 2 arise for the implementation of this second array; in addition, since the logic between the array inputs and the full adder inputs has now been increased, a significant reduction of the clock cycle can be achieved by pipelining the operations of this input logic and those of the full adders, this can be achieved by introducing memory elements on the outputs of the AND gates.

Signed multiplication/addition

The array for signed multiplication can be used as the kernel of an arithmetic unit able to perform the following computation

$$G = X\,Y + A \tag{17}$$

where X and Y are represented in 2's complement, with the notation shown in (13), A is represented as P in (14) and G is represented as follows:

$$G=-(e_{m+n}+f_{m+n})2^{m+n}+\sum_{j=0}^{m+n-1}(c_j+s_j)\,2^j \tag{18}$$

Note that the presentation of G is basically the same as for A and the product XY, the only difference is given by the larger number of digits used for G.
The function (17) can be performed as follows:

$$R_0 = 3\cdot 2^{m+n-1} \tag{19.a}$$

$$R_1=R_0+(c_{m+n-1}+s_{m+n-1})2^{m+n-1}+2^{m+n-2} \tag{19.b}$$

$$R_2=R_1+(c_{m+n-2}+s_{m+n-2})2^{m+n-2}+\bar{x}_{n-1}\bar{y}_{m-1}2^{m+n-2}+$$
$$+x_{n-1}\sum_{j=0}^{m-2}\bar{y}_j 2^{j+n-1}+x_{n-1}2^{m-1} \tag{19.c}$$

$$R_i=R_{i-1}+(c_{m+n-i}+s_{m+n-i})2^{m+n-i}+x_{n-i+1}\sum_{k=0}^{m-2}y_k 2^{k+n-i+1}+$$
$$+\bar{x}_{n-i+1}y_{n-1}2^{m-1-i} \tag{19.d}$$

$$R_i=R_{i-1}+(c_{m+n-i}+s_{m+n-i})2^{m+n-i}+\bar{x}_{n-i+1}y_{m-1}2^{m+n-i}+$$
$$+x_{n-1}2^{m-1}+x_{n-i}\sum_{k=0}^{m-2}y_k 2^{n-i+k+1}\quad ;\quad i=n-m-2 \tag{19.e}$$

In this algorithm the first step is introduced only to perform the sign extention of the operand A (if needed), then from step 2, it is similar to the signed multiplication algorithm, with a new addend given by the input digit of the operand A.
Once again the partial result R_i is represented by the addition of M and Q, however in this case we have:

$$M_0 = M_1 = M_2 = 0$$

$$M_i=M_{i-1}+(e_{m+n-i+3}+f_{m+n-i+3})2^{m+n-i+3} \tag{20}$$

$$Q_i=q'_{i,m+n+3-i}+q''_{i,m+n+2-1}+\sum_{k=0}^{m+n-i+3}(q'_{i,k}+q''_{i,k})2^k \tag{21}$$

Note that the new difinition of Q_i is slightly different from that given in section 2. The recursion step can be written as follows.

$$Q_{i-1}+\bar{x}_{n-i+1}y_{m-1}2^{m+n-i+1}+\sum_{j=0}^{m-2}x_{n-i+1}y_j 2^{j+n-i+1}-$$
$$-(c_{m+n-i+1}+s_{m+n-i+1})2^{m+n-i+1}=\sum_{k=m+n-i+1}^{m+n+3-i}q'_{i-1,k}2^k+$$
$$+(c_{m+n-i+1}+s_{m+n-i+1}+q''_{i-1,m+n-i+1})2^{m+n-i+1}+$$
$$+(x_{n-i+1}+y_{m-1}+q'_{i-1,m+n-i}+q''_{i-1,m+n-i})2^{m+n-i}+$$
$$+\sum_{j=n-i+1}^{m+n-i-1}(q'_{i,j}+q''_{i,j}+x_{n-i+1}y_{j-n+i-1})2^j+$$
$$+\sum_{j=0}^{n-i}(q'_{i-1,j}+q''_{i-1,j})2^j \tag{22}$$

By applying the usual transformation (10) to the previons equation we get:

$$Q_{i+1}+\bar{x}_{n-i+1}y_{m-1}2^{m+n-i}+(c_{m+n-i+1}+s_{m+n-i+1})2^{m+n-i+1}=$$
$$=q'_{i-1,m+n-i+3}2^{m+n-i+2}+(\alpha_{m+n-i+1}+$$
$$+q'_{m+n-i+2})2^{m+n-i+2}+(\alpha_{m+n-i}+\beta_{m+n-i+1}+$$
$$+q'_{i-1,m+n-i+1})2^{m+n+1-i}+\sum_{j=0}^{m+n-i}(q'_{i,j}+q''_{i,j})2^j \tag{23}$$

A double application of (10) leads to:

$$Q_{i-1}+x_{n-i+1}y_{m-1}2^{m+n-i}+(c_{m+n-i+1}+s_{m+n-i+1})2^{m+n-i+1}=$$
$$=(q'_{i-1,m+n-i+3}+\alpha_{m+n-i+2})2^{m+n-i+3}+$$
$$+\beta_{m+n-i+2}2^{m+n-i+2}+\beta_{m+n-i+1}2^{m+n-i+1}\sum_{k=0}^{m+n-i}(q'_{i,k}+$$
$$+q''_{i,k}) \tag{24}$$

which gives the new digit of the result to be included in M_j and the new value of Q_i as well.
The array implementing this algorithm is shown in Fig. 4.
It is basically the same as that in Fig. 3 with some additional full adders used to add the c and s inputs, representing the serial digit of A.

However, some difference in the operation exists. First, the most significant digit of A must be applied one clock cycle before the most significant bit of the factor X; then, in the following clock cycle, the most significant digit of A is repeated and the most significant digit of X is entered, with START set to 1.

4. Discussion

Since most of the computer systems in use today represent the numbers in 2's complement notation, the advantages of the arithmetic units manipulating numbers in "non standard" form are more evident when computation intensive applications are considered. In these cases, the time spent for performing the conversion from and to the 2's complement are more than compensated by the higher speed achieved by using "non standard" representation.

The algorithms which can be solved by means of systolic arrays are, in general, computational intensive; hence in order to show the effectiveness of the solutions proposed, we shall now be examining the performance of a linear systolic array for matrix-vector multiplication[6].

First, it is worth noting that the algorithms presented are on-line[5] respect to the serial operand(s), since they produce the result serially starting from the most significant one, and to produce each digit it is not necessary to know all the digits of the serial operand(s). Formally, it can be seen from (20) that when the third digits of both the serial factor and the serial addend are input in the array then the output digit with weight m + n is generated. This is the most significant digit of the result, since the whole operation includes a mxn bit multiplication, whose most significant digit has a weight m + n-1, and an addition of this product with another number, which leads to a weight of m + n for the most significant digit.

Hence, the on-line delay for the addition/multiplication algorithm is 2. However, this is only a theoretical delay, because the length of each clock cycle has to be taken into account.

If we want to obtain a real latency time equal to 1 clock cycle, all the optional memory elements of Fig. 2 should be omitted. In this way the clock cycle length will be

$$t_c = 3\sigma + t_m \qquad (25)$$

where σ is the delay of a single full adder and t_m is the delay of a memory element. Thus, we have shortened the latency time, but we have lessened the steady-state throughput of the pipelined unit.

Grnarov and Ercegovac[6] have analyzed the performances of a linear array for band matrix-vector multiplication, when on-line arithmetic units are used.

The structure considered here is the same as in their paper[6] with the difference that the matrix coefficients are entered in parallel 2's complement form, the multiplication/addition units used are those shown in the previons section, and the vector components are input serially in 2's complement form. Since these units produce the result on-line respect to their serial inputs, the same formulas[6] also hold in this case. Hence the total operation time is:

$$T_G = t_c[\delta + 1 + (p-1)n + nN] \qquad (26)$$

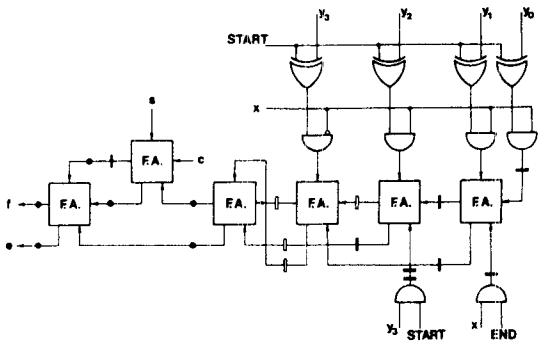
where p is number of non-zero coefficients in the first now, N is the matrix and vector size and $\delta + 1$ is the latency time of each multiplication/addition unit including the delay introduced by the pipelining scheme.

As shown before, both δ and t_c depend on the degree of pipelining chosen for the implementation. One of the two extreme solutions is that obtained by omitting all the optional memory elements of Fig. 4.

With this solution, the total operation time becomes:

$$T_G = (3\sigma + t_{in} + t_m)\,(3 + (p-1)n + nN) \qquad (27)$$

The opposite solution uses all the optional memory elements; this leads to:



▯ memory element (in.value 1)
▮ memory element (in.value 0)
• optional memory element

Fig. 4. Array implementing the addition/multiplication, with parallel factor of 4 bits.

$$T_G = (\sigma + t_{in} + t_m)\ (5+(p-1)n+N) \qquad (28)$$

where t_{in} is the delay of the input addition/subtraction logic.

Of course, the selection of the best solution depends on the values of p, n, N and on the relative values of σ, t_{in} and t_m.

Bit-level pipeling is also possible when conventional serial/parallel arithmetic is used. However, in this case, if we want to obtain only the n most significant bits of the result, the first bits output by each unit have to be discarded.

Assumming the operands are represented by n bits, we have to wait until the (n + 1)-th output bit in order to start the subsequent operation. Hence the latency time becomes:

$$T_L = \lceil n+1+(p-1)n \rceil\ t_c \qquad (29)$$

and the total operation time is:

$$T_c = \lceil n+1+(p-1)n+2nN \rceil\ t_c \qquad (30)$$

The term 2nN is derived from the fact that, for each result, each unit produce at least n least significant bits to be discarded. The clock cycle is the same as in (28), since the implementation of a conventional serial/parallel multiplier leads to an array based on the same cells as in Fig. 4, but with different interconnections[8-10].

The only difference in cost between the proposed adder/multiplier and the conventional ones derives from two extra full adders used in the array proposed in this paper. Hence, with a minimal additional cost, it is possible to achieve the speed advantages illustrated by Fig. 5, where the values of T_c and T_G are plotted.

On-line arithmetic is another possible competitor of the solutions proposed, since the latter share some characteristics introduced by the former.

The performance of a systalic array for band matrix-vector multiplication has been previously studied[6], where it has been shown that the total operation time is:

$$T_{on} = \lceil 2+(p-1)+nN \rceil\ t_{on} \qquad (31)$$

It can be seen that the number of clock cycles is smaller tham for T_G. However, the greater complexity of the implementation leads to longer clock cycles, or, if we want to shorten the clock cycle, to a larger number of clock cycles for T_{on}.

Our evaluation is based on the implementation of a multiplication division unit presented in the literature[7].

The unit is composed by n processing elements plus additional control logic. Each PE requires, with radix 2, 344 gates, leading to a cost of at least 344n gates.

Or the other hand, assuming that 9 gates are required by a full adder, 6 gates are required by a memory element and 3 gates are required by an EX-OR gate, the total complexity of the array for addition/multiplication is only 29 n.

This higher complexity leads to slower operations since on algorithm step can be performed in 29 times the single gate delay; while the solutions proposed here, can have a clock cycle equal to 7 times the gate delay, assumming $2 = \sigma$, $t_{in} = 3$, $t_m = 2$.

In the light of the real values of the clock cycle, can be seen from (31) and (27), that the solution proposed is considerably faster than the corresponding on-line implementation.

This comparison with the on-line arithmetic could not be considered fair, without mention of the advantage of the on-line units. Namely, the uniform representation of the input and the

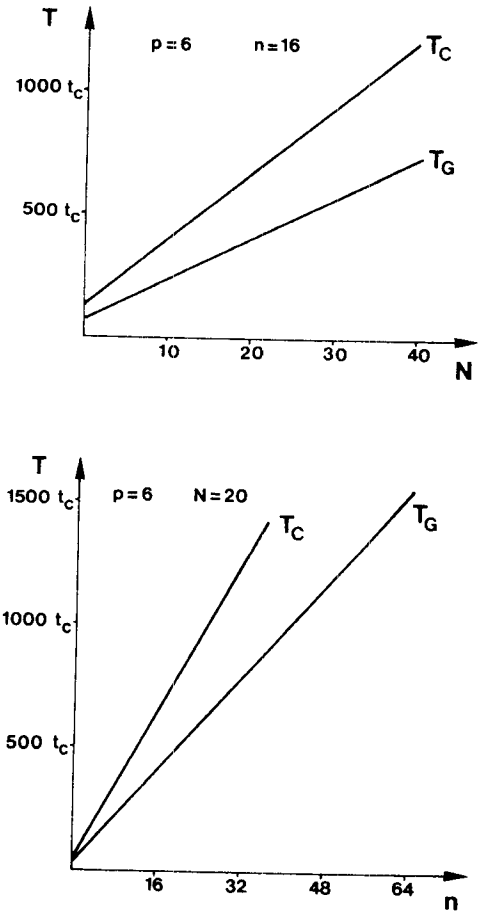


Fig. 5. Evaluation of the operation time T for the conventional, T_c, and the proposed, T_G, implementations.

outputs, which on the one hand causes cost increase and slower operation, but on the other hand allows the use of these units for implementing any possible arithmetic algorithm.
In conclusion the above discussion confirms the major claim of this paper. The units presented here are a very good solution for implementing special purpose processors for on important, though restricted, class of arithmetic algorithms.

5. Conclusions

Non redundant number systems are an attractive solution for implementing special purpose processors devoted to the excution of some arithmetic algorithm(s). Their major feature is the high speed often achievable by using redundant representations, which becomes cost-effective when a large number of computations is required, so that the final conversion of the result requires only a small fraction of the total operation time.
Three arrays have been presented in this paper, they achieve the same speed as other similar units based on different number representations, with a cost substantially identical to the units processing number in non-redudant representation.
First an array for positive integer multiplication has been introduced.
Then two arrays for signed number multiplication ad multiplication/addition have been presented. It is assumed that one factor is input in parallel, while the other(s) enter(s) the array digit by digit, with the most significant one first.
The result is also generated serially starting with the most significant digit, and it is represented in redundant form using 2 bits per digit. Both factors are assumed to be represented in 2's complement, while the addend input (for multiplication/addition only) is represented in the same way as the result.
The non-uniform choice for the input and output numbers restricts the applications of these arrays to a limited, though important, class of algorithms, including signal correlation, non-recursive digital filtering and matrix multiplication.
However, with this choice, the arrays achieve the same advantages as other units based on redundant number representation, with a significant reduction of cost.

REFERENCES

1. Avizienis, A. "Signed Digit Number Representation for Fast Parallel Arithmetic, IRE Trans. Electron. Comput., vol. EC-10, 10(1961) pp. 389-400.

2. Ercegovac, M., D. "An On-Line Square Rooting Algorithm", Proc. 4th Sym. on Computer Arithmetic, October 1978, pp. 183-189.

3. Trivedi, K., Ercegovac, M., D. "On Line Algorithms for Division and Multiplication", IEEE Trans. Comp., vol. C-26, 7 (1977), pp. 681-687.

4. Ercegovac, M.D., "A General Method for Evaluation of Functions and Computations in a Digital Computer", PhD. Thesis, Report No. 750, Department of Computer Science, University of Illinois, Urbana, August 1975.

5. Ercegovac, M.D. and Grnarov, A.L., "On the Performance of On-Line Aritmetic", Proc. 1980 International Conference on Parallel Processing, August 1980.

6. Grnarov, A., L., Ercegovac, M., D. "VLSI Oriented Iterative Networks for Array Computations", Proc. ICCC, October 1980, pp. 60-64.

7. Gorij-Sinaki A., Ercegovac M.D. "Design of a digit-slice on-line arithmetic unit", Proc. 5th Sym. on Computer Arithmetic, April 1981, pp.72-80.

8. L. Dadda, D.Ferrari: Digital multipliers a unified approach, Alta Frequenza, vol. 37, 11(1968), pp. 1079-1089.

9. E.E. Swartzlander Jr., "The quasi serial multiplier", IEEE Trans. Comp., vol. C-22, 4(1973) pp. 317-321, April 1973.

10. Am 25LS14 and Am 25LS15 data sheets, in AMD Bipolar microprocessor logic and interface data Book, 1981.

11. Garner H.L. "Number systems and arithmetic", in Advances in Comp., vol. 6, 1965, pp. 131-195.

12. Baugh C.R., Wooley B.A., "A two's complement parallel array multiplication algorthm", IEEE Trans. on Comp., vol. C-22, (1973) pp. 1045-1047.