$n \times n$ Carry-Save Multipliers without Final Addition

Paolo Montuschi *Luigi Ciminiera*
Dipartimento di Automatica e Informatica,
Politecnico di Torino, corso Duca degli Abruzzi 24,
10129 Torino (Italy)

Abstract

Carry-save multipliers require an adder at the last step to convert the carry-sum representation of the most significant half of the result into an irredundant form. This paper presents a multiplication scheme where this conversion is performed with a circuit operating in parallel with the carry-save array.

The resulting implementation, when a radix-2 adder array is used, produces a result on $2n$ bits with a delay comparable to the multiplier proposed by Ercegovac and Lang in [13]. When a radix-4 array is employed, the proposed unit is almost twice as faster as the units proposed by Nakamura in [18] and Jullien *et al.* in [27].

Keywords: carry-save addition, multiplication, on-the-fly conversion, redundant number representations.

1 Introduction

The design of high speed multipliers has always played a primary role in computer arithmetic. In the literature, interesting multiplication schemes have been presented in [4], [5], [6], [10], [17], [21], [26], and more recently in [3], [7], [8], [13], [16], [19], [22], [23], [24], [25].

Many of the proposed multipliers use, internally, a carry-sum representation of the partial accumulated products. The carry-sum is one of the early types of redundant representation introduced in order to conveniently address the problem of carry propagation. The signed digit redundant representation is ideal for representing results produced both by iterative algorithms for SRT [20] division and square root [14] and by on-line algorithms [12]. On the other hand, the carry-sum form is more popular than signed digit and well suited for the representation of results coming from successive accumulations, (such as the partial results during a multiplication) and, in general, is easy to implement when the operands are in non redundant form. The main reason of this, is that a carry save adder is simpler than a signed digit adder [2], [3], [16]. However, the former accepts operands in irredundant form, while the latter uses redundant operands. Therefore, once the product of negative numbers has been transformed into a sequence of additions, (like the Baugh and Wooley algorithm [4], or by encoding of the partial products [24] or by means of other techniques [19], [23]), the carry-sum representation of the partial accumulated products is cheaper to handle.

Multipliers based on the signed digit representation of the products have been presented in [13]. By carrying out an on-the-fly conversion procedure based on the scheme presented in [11], Ercegovac and Lang present in [13] a $n \times n$ multiplier not requiring the carry propagated addition in the last step of the "classical" multipliers, but providing a result only on the most significant n bits.

On the other hand, our proposal consists in two schemes of carry-sum-adder-based $n \times n$ multipliers, for either binary or two's complement numbers, one with radix-2 and the other with radix-4 adder array. The proposed multipliers can be considered as variations of the "classical" carry-save solution [15], [22], and they feature all the $2n$ bits of the result. Actually, they produce the least significant n'' bits of the result in irredundant form, and the most significant $n' = 2n - n''$ digits in carry-sum form, but the latter are produced starting with the most significant one. The digits in redundant form are converted in parallel with the computation of the least significant ones, so that, when the last redundant digit is produced, the whole product value in irredundant form can be obtained after a constant (i.e. independent of n) delay.

The design of $n \times n$ two's complement and binary multipliers is presented in section 2, where we discuss also the problem of on-the-fly converting the result in irredundant form. The proposed multipliers are then evaluated in section 3, considering both their hardware requirements and their speed of operation.

2 Design of binary and two's complement multipliers

We propose a multiplier which produces a result not requiring the carry propagate addition in the last step of some conventional schemes [15], [22], and with respect to the multiplier proposed by Ercegovac and Lang in [13] (which is based on a signed representation), providing all the $2n$ bits of the result. It performs the multiplication producing the least significant n'' bits in carry assimilated representation and the most significant $n' = 2n - n''$ in carry-sum form, which are on-the-fly (and in parallel) converted in non-redundant representation.

Our unit generates all the elementary product terms by operating radix-2 and, in the case of two's complement multiplier, implements the algorithm by Baugh and Wooley [4] with the extension by Blankenship [5].

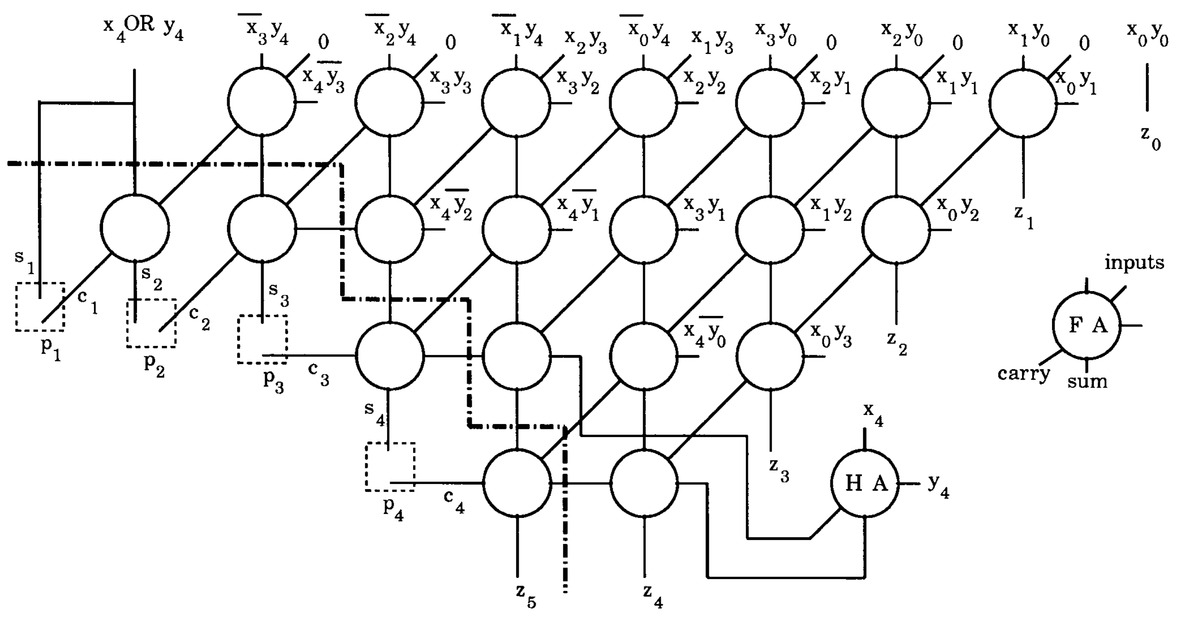

Figure 1: Proposed array of adders for $n = 5$

As with many other parallel multiplying arrays, the multiplication can be considered as being performed in 2 phases:

1. all the elementary products are generated;

2. the bits obtained in the phase 1 are added by a suitable array.

The first phase requires a constant time, and it is not of interest in our case, because, once the radix has been selected, its complexity and delay are common to different algorithms. Extension to higher radices of the multiplier for the phase when all the elementary products are generated, can be performed by using algorithms based on the encoding of partial products [24] or on other techniques [19], [23], in order to eliminate the need for sign extension for the negative terms.

Several solutions have been proposed in the literature for the array assimilating the bits produced in the first phase. Of particular interest are the "classical" radix-2 assimilation array [15], and the radix-4 array proposed by Nakamura in [18] and by Jullien *et al.* in [27]. The "classical" array for radix-2 assimilation considers the full adder as the basic assimilating element. On the other hand, Nakamura uses for his radix-4 assimilation a $(5,3)$ counter as the basic assimilating element, while Jullien *et al.* employ a 2-bit full adder. We start by considering the radix-2 and then we present the radix-4 assimilation array.

2.1 Radix-2 adder array

Our attention is focused only on the second phase; the array used for two's complement numbers is shown in Fig. 1 for $n = 5$. Globally, the whole unit is basically a carry save multiplier, where some additions are anticipated in order to obtain the digits in redundant form as soon as possible. This also implies the introduction of a diagonal line of full adders (the one below the dashed line in Fig. 1). In Fig. 1 we have denoted with

$$X = -x_4 \cdot 2^4 + \sum_{i=0}^{3} x_i \cdot 2^i$$

$$Y = -y_4 \cdot 2^4 + \sum_{i=0}^{3} y_i \cdot 2^i$$

$$Z = X \cdot Y = -z_9 \cdot 2^9 + \sum_{i=0}^{8} z_i \cdot 2^i$$

the two operands and the result, respectively.

In Fig. 1 the $(n+1)$ least significant bits of the product have been denoted with the symbols z_i (with $i = 0, \cdots, n$), since they are produced in irredundant form and they are equal to the $(n+1)$ least significant bits of the result. Therefore, in such a case the subscript i of z_i is related to the weight of the bit of the final result. On the other hand, the $(n-1)$ most significant digits of the result are produced as carry and sum

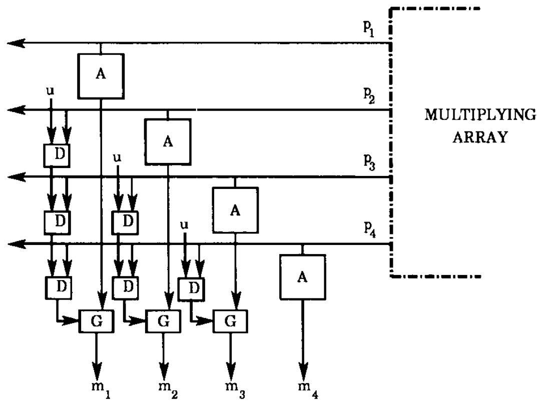

Figure 2: On-the-fly conversion hardware for $n = 5$

pairs, i.e. in non-assimilated carry-sum form. In Fig. 1 they have been denoted with the symbols p_i (with $i = 1, \cdots, n-1$), where the subscript i of p_i is related to the order of availability of the pair p_i. Therefore, p_1 is the first available carry and sum pair, p_2 is the second, and so on; each digit p_i in effect comes from the assimilation of one carry and one sum bit c_i and s_i, respectively. To summarize, in Fig. 1 the $(n-1)$ most significant digits are labeled in the order they are produced, while the $(n+1)$ least significant bits are labeled with index corresponding to the bit weight inside the final result.

The $(n+1)$ least significant bits are produced after a delay of n full adders. The $(n-1)$ most significant digits are also produced after a delay of n full adders. In particular, it should be noted that the most significant carry and sum pair of the final product p_1 is produced after a delay of 2 full adders. The successive carry and sum pairs follow thereafter, with delays of one full adder between one and the next. In this computation, the half adder used to sum the additional inputs required by the Baugh and Wooley algorithm does not contribute to the total delay, as it operates in parallel with the rest of the array.

A radix-2 multiplier for positive numbers can be immediately obtained from the one of Fig. 1, by eliminating all the additional inputs required by the Baugh and Wooley algorithm, as well as the extra half adder, while at the same time generating the appropriate elementary products.

2.2 On-the-fly conversion and output of the result for the radix-2 adder array

By using the hardware of Fig. 2, the redundant digits are produced by the array starting with the most significant one. The hardware of Fig. 2 is used to implement a sort of on-the-fly conversion algorithm which is similar to the methodology presented in [13]. However, in [13] the most significant digits are in signed digit form, while in the proposed algorithm they are in carry save

Table 1: Conversion control signals

$D_k[i]$	Decision at level i about value of product digit m_k
u	undecided
g	decided: no change ($m_k = A_k$)
t	decided: increment modulo the operating base (($m_k = A_k + 1$) *mod* R)

representation. Observe that the hardware for the on-the-fly conversion of Fig. 2 operates in parallel with the computation of the following digits.

The carry-sum representation of the product is converted into the corresponding conventional representation m. During the on-the-fly conversion, we produce both the assimilations of the bits p_k using the blocks A and the control signals $D_k[i]$ associated with each bit A_k output from blocks A. This is done so as to determine whether the final bit m_k is A_k or $\bar{A}_k$ (i.e. $(A_k + 1)$ *mod* 2). The meaning of the control signals is given in Table 1. In Fig. 1 we observe that, in correspondence with each stage of the array, one pair of carry and sum bits is output from the array. By considering that p_i has the value of the assimilation of the bits c_i and s_i, a high level description of the process is:

$$A_i = \begin{cases} 0 & if \;\; p_i = 2 \\ 1 & if \;\; p_i = 1 \\ 0 & if \;\; p_i = 0 \end{cases} \tag{1}$$

From (1) it follows that $A_i = c_i \oplus s_i$, where the symbol $\oplus$ stands for the EXOR operator. At the beginning we have $D_i[i] = u$, and for $k < i$ we have

$$D_k[i+1] = \begin{cases} u & if \;\; D_k[i] = u \; and \; p_i = 1 \\ g & if \;\; D_k[i] = u \; and \; p_i = 0 \; or \; D_k[i] = g \\ t & if \;\; D_k[i] = u \; and \; p_i = 2 \; or \; D_k[i] = t \end{cases} \tag{2}$$

It is interesting to note that the rules in Table 1 implement the same addition algorithm as carry-select [15]; however, this implementation differs from the previous one, because it takes into account the different availability in time of the digits to be added.

We denote with $\gamma_k[i]$ and $\delta_k[i]$ the most and least significant bits of $D_k[i]$. We choose to represent the condition $D_k[i] = g$ with the pair $(\gamma_k[i] = 1, \delta_k[i] = 0)$, the condition $D_k[i] = t$ with the pair $(\gamma_k[i] = 0, \delta_k[i] = 1)$, and the condition $D_k[i] = u$ with the pair $(\gamma_k[i] = 0, \delta_k[i] = 0)$. The boolean expressions used to obtain the bits of $D_k[i+1]$ are derived from (2) and are

$$\begin{aligned} \gamma_k[i+1] &= \overline{\delta_k[i]}\bar{c}_i\bar{s}_i + \gamma_k[i] \\ \delta_k[i+1] &= \overline{\gamma_k[i]}c_i s_i + \delta_k[i] \end{aligned} \tag{3}$$

From Table 1 we observe that at the final level, (i.e. when $i = n-1$), only when $D_k[n-1] = t$ it is necessary to complement the value of A_k, that is $m_k = \bar{A}_k$. If $D_k[n-1] = u$ or $D_k[n-1] = g$, we will have $m_k = A_k$. Therefore, at the final level only the signal $\delta_k[n-1]$ is necessary. The bit m_k is obtained as

$$m_k = A_k \oplus \delta_k[n-1] = c_k \oplus s_k \oplus \delta_k[n-1] \quad (4)$$

With reference to the general scheme which can be derived from Figures 1 and 2, we observe that the "on-the-fly-converted" bit m_k corresponds to the bit z_{2n-k} of the result, i.e. $z_{2n-k} = m_k$.

2.3 Radix-4 adder array

The scheme of the proposed radix-4 adder array used for two's complement numbers is shown in Fig. 3 for $n = 8$, where we have denoted with

$$X = -x_7 \cdot 2^7 + \sum_{i=0}^{6} x_i \cdot 2^i$$

$$Y = -y_7 \cdot 2^7 + \sum_{i=0}^{6} y_i \cdot 2^i$$

$$Z = X \cdot Y = -z_{15} \cdot 2^{15} + \sum_{i=0}^{14} z_i \cdot 2^i$$

the two operands and the result, respectively.

In Fig. 3 have been used the blocks whose definition is reported in Fig. 4. In particular: Fig. 4a depicts an EXOR gate; Fig. 4b depicts a basic processing element (BPE), where the inputs are given on $\alpha, \beta, \gamma, \delta, \varepsilon$ and the outputs are on μ, η, ξ, being μ and ξ the most and least significant bits, respectively; Fig. 4c depicts a 2-bit full adder (CLA_2), where $2(\alpha+\beta)+\gamma+\delta+\varepsilon = 2^2\mu+2\eta+\xi$; Fig. 4d depicts a full adder, where $\alpha+\beta+\gamma = 2\mu+\eta$; Fig. 4e depicts a binary full adder (BFA), where $\alpha+\beta+\gamma\cdot\delta = 2\mu+\eta$.

In Fig. 3 the basic processing element (BPE) used for the "compression" of the product terms, is the $(5,3)$ counter of [18]. A similar array can be used with the 2-bit full adder CLA_2 introduced in [27] (which implements the addition of two 2-bit operands with a carry-in). In fact, both the $(5,3)$ counter and the 2-bit full adder produce a result on the weights $2^2, 2^1$ and 2^0; as for the inputs, 2 inputs of the same weight are used in both cases to add 2 new elementary products, while the remaining 3 inputs are used to add carries and partial results from neighbor cells. Thus, the difference between the use of the two types of cells is only in the weights of the latter 3 inputs, that are the same as those of elementary products in [18], and different as in [27]. Observe that the number of rows of BPEs is $n/2$.

Surrounding the adder array, we can identify two different portions of hardware. In the lower right corner (below the right dashed line) of Fig. 3, we have the hardware which is dedicated to the determination of the least significant digits of the result in non redundant form. These circuits are based on the use of binary full adders (BFA) and EXOR gates and do not substantially differ from the solutions proposed by Nakamura in [18] and Jullien *et al.* in [27], and hence will not be discussed.

On the other hand, the hardware for the determination of the most significant digits of the result, is identified in the lower left corner of Fig. 3. As the radix-2 adder array of Fig. 1, some additions are anticipated in order to obtain the most significant digits in carry-sum form as soon as possible, thanks to the introduction of a diagonal line of full adders FA's and other 2-bit full adders CLA_2's (the one below the left dashed line in Fig. 3). In order to achieve the maximum advantages in terms of parallelism, it is implicitly assumed that the "assimilation" operated by one CLA_2 is carried out in a time no longer than the delay for the "compression" of some product terms operated by one line of BPEs. This assumption implies that a $(5,3)$ counter should not be faster than a 2 bit full adder.

In Fig. 3 we have used a similar notation as what adopted in Fig. 1, for the most and least significant digits, respectively. In fact, in Fig. 3 too, the $(n+2)$ least significant bits are labeled with index corresponding to the bit weight inside the final result. On the other hand, the $(n-2)$ most significant bits are produced from $(n/2-1)$ radix 4 units, and the corresponding most significant $(n/2-1)$ radix 4 (redundant) digits are labeled in the order they are produced. The $(n+2)$ least significant bits are produced after a delay of $(n/2+3)$ BPEs. The $(n/2-1)$ most significant digits are also produced after a delay of $(n/2+3)$ BPEs. In Fig. 3 the most significant digits are produced in groups of three bits, i.e. for the $i-th$ digit two of weight $2^{2n-2i+1}$ and one of weight 2^{2n-2i}. In particular, it should be noted that the most significant (redundant) digit of the final product p_1 is produced after a delay of 5 BPEs. The successive digits follow thereafter, with delays of one BPE between them.

2.4 On-the-fly conversion and output of the result for the radix-4 adder array

The on-the-fly conversion of the most significant digits coming out from the array of Fig. 3 is based on similar operating principles as these outlined in section 2.2 for the radix-2 adder array. Again, we produce both the assimilations of the bits p_k using the blocks A and the control signals $D_k[i]$ associated with each digit A_k output from blocks A, used to determine whether the final radix-4 digit m_k is A_k or (A_k+1) *mod* 4 (see Table 1). The high level scheme of the on-the-fly conversion hardware is again this of Fig. 2 where, however, the digits A_k and m_k are composed by 2 bits: $A_{k,1}$ and $A_{k,0}$ (most and least significant, respectively).

In Fig. 3 we observe that in correspondence with each stage of the array, one (redundant) digit composed by three bits is output from the array: the i-th digit p_i has two bits of weight $2^{2n-2i+1}$ and one of weight 2^{2n-2i}.

Let us denote with $c_{i,1}$ and $s_{i,1}$ the two bits of weight $2^{2n-2i+1}$ and with $c_{i,0}$ the bit of weight 2^{2n-2i} of the i-th most significant (redundant) digit output from the array. By considering that p_i has the value of

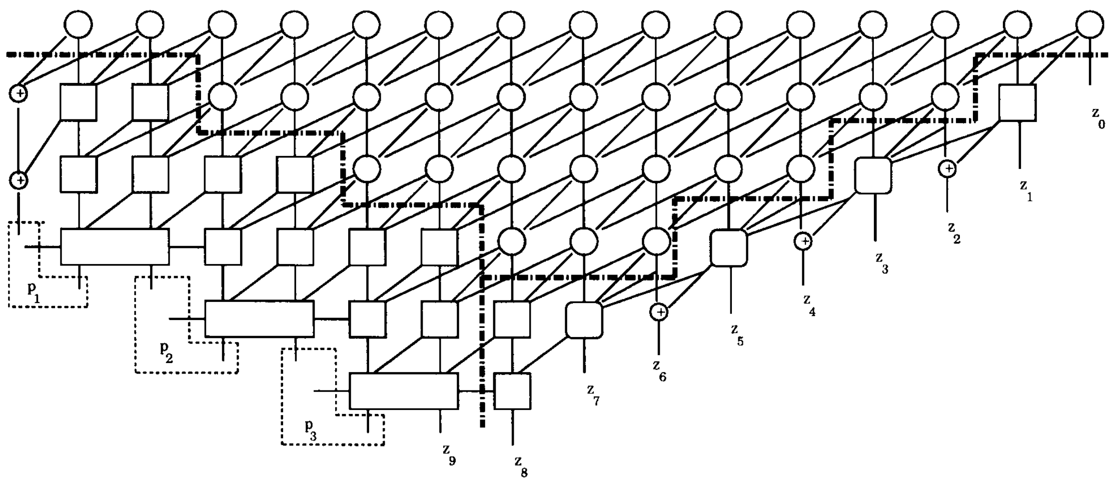

Figure 3: Proposed radix-4 array of adders for $n = 8$

the assimilation of the bits $c_{i,1}, s_{i,1}$ and $c_{i,0}$ (i.e. $p_i = 2(c_{i,1} + s_{i,1}) + c_{i,0}$), a high level description of the process is:

$$A_i = \begin{cases} 0 & if\ \ p_i = 0\ or\ p_i = 4 \\ 1 & if\ \ p_i = 1\ or\ p_i = 5 \\ 2 & if\ \ p_i = 2 \\ 3 & if\ p_i = 3 \end{cases} \tag{5}$$

From (5) it follows that $A_{i,1} = c_{i,1} \oplus s_{i,1}$ and $A_{i,0} = c_{i,0}$. At the beginning we have $D_i[i] = u$, and for $k < i$ we have

$$D_k[i{+}1] = \begin{cases} u & if\ D_k[i] = u\ and\ p_i = 3 \\ g & if\ D_k[i] = u\ and\ p_i \leq 2\ or\ D_k[i] = g \\ t & if\ D_k[i] = u\ and\ p_i \geq 4\ or\ D_k[i] = t \end{cases} \tag{6}$$

Again, we denote with $\gamma_k[i]$ and $\delta_k[i]$ the most and least significant bits of $D_k[i]$, and we use the same coding introduced in section 2.2 to represent the conditions $D_k[i] = g, t, u$. The boolean expressions used to obtain the bits of $D_k[i+1]$ are derived from (6) and are

$$\begin{aligned} \gamma_k[i+1] &= \overline{\delta_k[i]}(\bar{c}_{i,1}\bar{c}_{i,0} + \bar{c}_{i,1}\bar{s}_{i,1} + \bar{s}_{i,1}\bar{c}_{i,0}) + \gamma_k[i] \\ \delta_k[i+1] &= \overline{\gamma_k[i]}c_{i,1}s_{i,1} + \delta_k[i] \end{aligned} \tag{7}$$

At the final level, (i.e. when $i = n/2 - 1$), only when $D_k[n/2 - 1] = t$ it is necessary to add 1 to the value of A_k, that is $m_k = (A_k + 1)\ mod\ 4$. Therefore, at the final level only the signal $\delta_k[n/2 - 1]$ is necessary. With reference to the general scheme which can be derived from Figures 2 and 3 we observe that the "on-the-fly-converted" bit m_k corresponds to the pair $z_{2n-2k+1}$, z_{2n-2k} of the result, i.e. $m_k = 2z_{2n-2k+1} + z_{2n-2k}$.

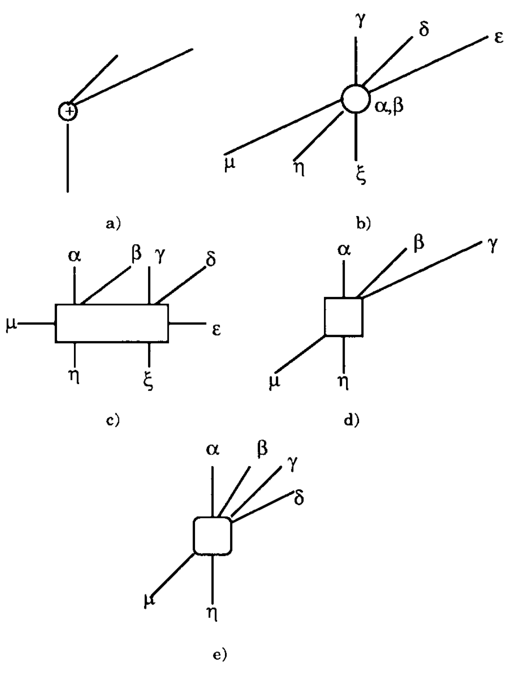

Figure 4: Proposed radix-4 array of adders for $n = 8$

3 Evaluation

There are two possible parameters to compute for evaluating the proposed multipliers: execution time and hardware requirements. In particular we will compare our unit featuring the radix-2 adder array on both aspects with the multiplier proposed by Ercegovac and Lang [13] since these multipliers use similar schemes. On the other hand, we will compare only the performances of the unit employing the radix-4 assimilation array with the units by Nakamura [18] and Jullien *et al.* [27], since the latter architectures do not require any supplementary hardware for the on-the-fly conversion of the most significant part of the result as it is done in [13].

3.1 Execution time of the unit with radix-2 assimilation

We assume that the blocks D of Fig. 2 which generate control signals $D_k[i]$'s, operate in parallel with the lines of the carry save adder array. Our assumption is realistic, since the complexity of the boolean functions (3) is comparable with the complexity of the boolean functions of a stage of the carry save adder array (i.e. a full adder). As can be observed in Figures 1 and 2, the critical path passes through the generation of the elementary product terms, through the n stages of the carry save adder array, through the last line of blocks D generating the function $\delta_k[n-1]$, and through the blocks G for the generation of the bits m_k of the result. It is important to note that the bottom rows of the conversion array (see Fig. 2) require a large fanout for the signals produced by the multiplying array. This may impose an execution time dependent on n; however, this problem can be overcame by using larger drivers, whose additional cost is negligible with respect to the complexity of the whole array.

Let us denote with t_{FA} the delay of a full adder; the delay of the proposed unit becomes

$$T_{pr} = t_{gen,pr} + nt_{FA} + t_{\delta_k[n-1]} + t_{m_k} \qquad (8)$$

where with $t_{gen,pr}, t_{\delta_k[n-1]}$ and t_{m_k} we have indicated the delay in generating the elementary product terms, and in generating $\delta_k[n-1]$ and m_k, respectively.

We assume that the generation of the function $\delta_k[n-1]$ according to (3) requires a delay of one full adder, and that the last EXOR required to determine the generic bit m_k according to (4) has also a delay of one full adder. In such a case equation (8) becomes

$$T_{pr} = (n+2)t_{FA} + t_{gen,pr} \qquad (9)$$

Since we implement a similar on-the-fly conversion mechanism, we compare our unit with the left-to-right multiplier proposed by Ercegovac and Lang in [13]. The multiplier of [13] implements the radix-4 modified Booth algorithm, and uses a signed digit adder array consisting of $[(n/2)-1]$ stages of signed digit adders. According to [13], the delay of the multiplier is

$$\begin{aligned} T_{EL} &= t_{rec,mul} + t_{gen,EL} + [(n/2)-1]t_{SDA} + \\ &\quad + t_{rec,add} + t_{ctrl} + t_{inc} \qquad (10) \end{aligned}$$

where $t_{rec,mul}$ is the delay for the recoding of the multiplier, $t_{gen,EL}$ is the delay in generating the elementary product terms, t_{SDA} is the delay of a signed digit adder, $t_{rec,add}$ is the delay in recoding and adding one digit which is output from the adder array, t_{ctrl} is the delay of the generation of the last control signals, and t_{inc} is the delay in incrementing a radix-4 digit. Reasonable assumptions for these delays are:

- $t_{rec,mul} \geq 0$;
- $t_{gen,EL} \geq t_{gen,pr}$; in fact, in the proposed architecture, the delay in generating the elementary product terms approximately equals the delay of an AND gate; on the other hand, in the architecture of [13] the generation is slightly more complex since it is necessary to make a selection of the $\pm 2, \pm 1$, and 0 multiples of the multiplicand;
- $t_{SDA} = 2t_{FA}$; because a radix-4 signed digit adder requires the propagation of the carry through one bit weight [2], [3], [16];
- $t_{rec,add} \geq 2t_{FA}$; because the recoding of the digit has the delay of t_{FA} and the addition requires at least t_{FA};
- $t_{ctrl} = t_{FA}$;
- $t_{inc} \geq t_{FA}$; because the last increment of a radix-4 digit has a delay of at least one full adder.

Therefore, from (10) we obtain

$$T_{EL} \geq (n+2)t_{FA} + t_{gen,pr} \qquad (11)$$

A comparison of (9) and (11) shows that under the previous assumptions regarding the delays, our architecture provides a multiplication scheme comparable to the one of [13]. However, our multiplier computes a result on $2n$ bits, while the multiplier proposed by Ercegovac and Lang in [13] provides only the most significant n bits.

3.2 Hardware requirements of the unit with radix-2 assimilation

It is not possible to reduce the size of the adder array as in the case of [13], and as in the Cray X-MP processor [1], since the final product is provided on all the $2n$ bits. From Fig. 1, we observe that the adder array requires $(n-1)^2$ full adders, 1 half adder, 1 OR gate (to perform the inclusive OR between the signs of the two multiplicands X and Y), and n^2 AND gates (to generate the elementary product terms). Therefore, the complexity of the adder array is approximately

$$K_{array} \approx (n-1)^2 K_{FA} + n^2 K_{AND} \qquad (12)$$

where with K_{FA} and K_{AND} we have denoted the area of a full adder, and of an AND gate, respectively. The hardware used to determine the most significant $n' = n-1$ bits of the result has a complexity of

$$K_{ms,pr} = \frac{(n-1)(n-2)}{2} \cdot K_D + (n-1)K_{FA} + (n-1)K_G$$

where with K_D and K_G we have denoted the complexity of the hardware required to implement the blocks D and G (to generate the m_k's according to (4)), respectively. Reasonable assumptions are that $K_D \approx 2K_{FA}$ (because of relations (3)), and $K_G \approx K_{FA}$ (because of relation (4)). In such a case, the complexity of the the hardware used to determine the most significant n bits is

$$K_{ms,pr} \approx (n^2 - n) \cdot K_{FA} \Rightarrow O(n^2) \qquad (13)$$

On the other hand, the complexity of the hardware used by the unit of [13] to convert the most significant n bits is

$$\begin{aligned} K_{ms,EL} &= n/2 \cdot K_{rec,add} + n/2 \cdot K_{SDA} + \\ &+ \frac{n/2(n/2-1)}{2} \cdot K_{ctrl} \end{aligned}$$

where $K_{rec,add}$, K_{ctrl} and K_{SDA} express the complexity of the module for recoding and adding one digit output from the adder array, of the module for generating the control signals, and of a signed digit adder, respectively. If we assume that $K_{rec,add} \approx 2K_{FA}$, $K_{ctrl} \approx K_{FA}$ and $K_{SDA} \approx 2K_{FA}$ [2], [3], [16], we get

$$K_{ms,EL} \approx \frac{n^2 + 7n}{8} \cdot K_{FA} \Rightarrow O(n^2) \qquad (14)$$

Finally, the classical carry save multiplier [15] requires a carry propagate adder so that the most significant bits can be obtained, and this multiplier has the complexity of $O(n \log(n))$, with a delay of $O(\log(n))$, as against to constant addition time in our algorithm. A comparison between (13), (14) and the complexity of the classical carry save multiplier, shows that the hardware in the proposed architecture to determine the most significant n bits is of the same order of complexity as the hardware required by the unit proposed by Ercegovac and Lang in [13], but is more complex than the carry propagate adder used in the classical carry save multiplier.

The extra hardware required by the proposed multiplier with respect to the classical solution [15] is the price paid for a faster multiplication scheme which avoids the final addition, but still provides all the $2n$ bits of the result in non redundant form.

3.3 Execution time of the unit with radix-4 assimilation

We assume that the blocks D of Fig. 2 which generate control signals $D_k[i]$'s, operate in parallel with the lines of the adder array, since the complexity of the boolean functions (7) is comparable with these implementing a BPE. In this case the critical path passes through the generation of the elementary product terms, through the $(n/2+3)$ stages of the adder array, through the last line of blocks D generating the function $\delta_k[n/2-1]$, and through the blocks G for the generation of the bits m_k of the result. The delay of the proposed unit becomes

$$T_{pr,radix\ 4} = t_{gen,pr} + (n/2+3)t_{BPE} + t_{\delta_k[n/2-1]} + t_{m_k} \qquad (15)$$

where with t_{BPE} we have indicated the delay of a BPE (the other symbols have already been defined in section 3.1). We assume that the generation of the function $\delta_k[n/2-1]$ according to (7) requires a delay of one full adder, and that the last increment required to determine the generic bit m_k has a delay of two full adders. In such a case equation (15) becomes

$$T_{pr,radix\ 4} = (n/2+3)t_{BPE} + t_{gen,pr} + 3t_{FA} \qquad (16)$$

According to the schemes of the multipliers of of [18] and [27], the delay is

$$T_{NJ} = t_{gen,NJ} + (n-1)t_{BPE} + t_{BFA} + t_{XOR} \qquad (17)$$

where $t_{gen,NJ}$ is the delay in generating the elementary product terms, t_{BFA} is the delay of binary full adder, and t_{XOR} is the delay of an EXOR gate. Reasonable assumptions for these delays are $t_{gen,NJ} = t_{gen,pr}$ and $t_{BFA} = t_{XOR} = t_{FA}$. Therefore, from (17) we obtain

$$T_{NJ} = (n-1)t_{BPE} + t_{gen,pr} + 2t_{FA} \qquad (18)$$

A comparison of (16) and (18) shows that under the previous assumptions regarding the delays, our architecture provides a multiplication scheme almost twice as faster as the units proposed in [18] and [27].

4 Conclusions

We have presented the design of both binary and two's complement $n \times n$ multipliers, where the addition required for the conversion of the result into conventional representation is implemented in overlapping with the elementary product reduction. In the proposed multiplication schemes, the computation of the least significant bits is carried out in parallel with the determination of the most significant digits, where the latter are represented in redundant (i.e. carry-sum) form. The conversion from carry-sum to irredundant form is also carried out in parallel with the generation of the other digits of the product.

Therefore, the additional delay required to obtain all the product digits in irredundant form once the carry-save operation is completed, is short and independent from n, while addition performed in cascade with elementary product reduction are larger with a delay dependent on n. The resulting implementation when a radix-2 adder array is used, produces a result on $2n$ bits with a delay comparable to the multiplier proposed by Ercegovac and Lang in [13] that produces only the n most significant ones. When a radix-4 array is employed, the proposed unit is almost twice as faster as the units proposed by Nakamura in [18] and Jullien *et al.* in [27].

References

[1] Annon, "Cray X-MP Computer Systems," Four-Processor Mainframe Reference Manual, HR-0097, Cray Research, Inc., 1985.

[2] A. Avižienis, "Signed-Digit Number Representation for Fast Parallel Arithmetic," IRE Trans. Electron. Comput., Vol.EC-10, September 1961, pp.389-400.

[3] W. Balakrishnan and N. Burgess, "Very-High-Speed VLSI 2s-Complement Multiplier Using Signed Binary Digits," IEE Proceedings, Part E, Vol.139, No.1, January 1992, pp.29-34.

[4] C. R. Baugh and B. A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," IEEE Trans. Comput., Vol.C-22, No.12, December 1973, pp.1045-1047.

[5] P. E. Blankenship, "Comments on 'A Two's Complement Parallel Array Multiplication Algorithm'," IEEE Trans. Comput., Vol.C-23, pp.1327, 1974.

[6] A. D. Booth, A Signed Binary Multiplication Technique," Quart. J. Mech. Appl. Math., Vol.4, Part 2, pp.236-240, 1951.

[7] P. R. Cappello and K. Steiglitz, "A VLSI Layout for a Pipelined Dadda Multiplier," ACM Trans. Comput. Systems, Vol.1, No.2, pp.157-174, May 1983.

[8] L. Ciminiera and A. Serra, "Fast Iterative Multipliying Array," Proc. 6th IEEE Symposium on Computer Arithmetic, Aarhus, Denmark, pp.60-66, June 1983.

[9] L. Ciminiera and P. Montuschi, "Higher Radix Square Rooting," IEEE Trans. Comput., Vol.C-39, No.10, October 1990, pp.1220-1231.

[10] L. Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, Vol.34, March 1965, pp.349-356.

[11] M. D. Ercegovac and T. Lang, "On-the-Fly Conversion of Redundant into Conventional Representations," IEEE Trans. Comput., Vol.C-36, pp.895-897, July 1987.

[12] M. D. Ercegovac and T. Lang, "On-Line Arithmetic: A Design Methodology and Applications in Digital Signal Processing" VLSI Signal Processing III, pp.252-263, 1988.

[13] M. D. Ercegovac and T. Lang, "Fast Multiplication Without Carry Propagate Addition," IEEE Trans. Comput., Vol.C-39, pp.1385-1390, November 1990.

[14] J. Fandrianto, "Algorithm for High Speed Shared Radix 8 Division and Radix 8 Square-Root," Proc. 9th IEEE Symposium on Computer Arithmetic, Santa Monica, CA, pp.68-75, September 1989.

[15] K. Hwang, "Computer Arithmetic: Principles, Architecture and Design," John Wiley and Sons, New York, 1978.

[16] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taninguchi and N. Takagi, "Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation," Proc. 8th IEEE Symposium on Computer Arithmetic, Como, Italy, pp.80-85, May 1987.

[17] A. R. Meo, "Arithmetic Networks and Their Minimization Using a New Line of Elementary Units," IEEE Trans. Comput., Vol.C-24, pp.258-280, March 1975.

[18] S. Nakamura, "Algorithms for Iterative Array Multiplication," IEEE Trans. Comput., Vol.C-35, pp.713-719, August 1986.

[19] M. Putrino, S. Vassiliadis and E. M. Schwarz, "Array Two's Complement and Square Function," Electron. Lett., Vol.23, No.22, pp.1185-1187, October 1987.

[20] J. E. Robertson, "A New Class of Digital Division Methods," IRE Trans. Electron. Comput., Vol.EC-7, pp.218-222, September 1958.

[21] E. E. Swartzlander, "The Quasi-Serial Multiplier," IEEE Trans. Comput., Vol.C-22, pp.317-321, April 1973.

[22] M. Uya, K. Kaneko and J. Yasui, "A CMOS Floating-Point Multiplier," IEEE J. Solid-State Circuits, Vol.SC-19, No.5, pp.697-701, October 1984.

[23] S. Vassiliadis, M. Putrino and E. M. Schwarz, "Parallel Encrypted Array Multiplier," IBM Journal of Research and Development, Vol.32, No.4, pp.536-551, July 1988.

[24] S. Vassiliadis, E. M. Schwarz and B. M. Sung, "Hard-Wired Multipliers with Encoded Partial Products," IEEE Trans. Comput., Vol.C-40, pp.1181-1197, November 1991.

[25] J. Vuillemin, "A Very Fast Multiplication Algorithm for VLSI Implementation," Integration: the VLSI Journal, No.1, pp.39-52, 1983.

[26] C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Trans. Electron. Comput., Vol.EC-13, pp.14-17, February 1964.

[27] Z. Wang, G. A. Jullien and W. C. Miller, "An Architecture for Parallel Multipliers," Proc. 25th IEEE Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, pp.403-407, November 1991.