

# Asynchronous Multipliers with Variable-Delay Counters<sup>\*</sup>

Gianluca Cornetta  
Computer Architecture Dept.  
Universitat Politècnica de Catalunya  
08034 Barcelona—Spain  
E-mail: cornetta@ac.upc.es

Jordi Cortadella  
Software Dept.  
Universitat Politècnica de Catalunya  
08034 Barcelona—Spain  
E-mail: jordic@lsi.upc.es

## Abstract

*Although multiplication is an intensely studied arithmetic operation and many fast algorithms and implementations are available, it still represents one of the major bottlenecks of many digital systems that require intensive and fast computations. This paper presents a novel design approach based on the well-known Baugh and Wooley algorithm, particularly appealing for asynchronous implementations and that may be easily mapped into a VLSI circuit. This technique has been applied to the design of a high-speed variable-delay multiplier that resulted to be faster than other synchronous and asynchronous implementations.*

## 1 Introduction

Multiplication is an intrinsically slow operation since a large number of partial products have to be added in order to obtain the final result. The most common techniques to speedup multiplication aim at reducing the number of partial products to decrease the execution time. This may be achieved by encoding the multiplier [4], or by using parallel counters [10]. The algorithm proposed in this paper has an implementation inspired by [7] but, unlike [7], it has a data-dependent execution time.

The idea of datapaths with variable execution time has already been applied to the realization of several arithmetic operations such as division and square root [18, 9], addition [11, 15] and multiplication [13], while a general method for synthesizing variable-delay pipelined datapaths has been described in [3]. For example the multiplier described in [13] is a bidimensional array of full adders with two possible delays. What determines the delay of a row of the array is the corresponding bit of the multiplier. This design approach limits the implementation only to radix-2 arrays.

The data-dependent computation times are determined by a speculation function as proposed in [9]. However, in this case it is not necessary the retiming of the algorithm in order to perform speculation and error detection and correction in parallel. In addition, unlike [13] the architecture described in this paper does not need an adder at the last stage, since digits are generated in redundant form starting from the most significant ones and conversion into non-redundant form may be performed on-the-fly [12].

Synchronization is achieved by means of a dual-rail encoding of the data bits [6]. This implies the use of differential logic. We choose to implement the basic cells using CPL gates [2]. The use of complementary pass-transistor gates is particularly appealing for low-power applications. Moreover, CPL can be faster than conventional CMOS logic. Nevertheless, the reduced output voltage swing requires the use of buffers to obtain a full-swing output voltage. In addition the complementary output necessary to implement the dual-rail protocol requires extra transistors and leads to a larger area occupation when compared with standard CMOS.

<sup>\*</sup>This work has been partially supported by the Ministry of Education of Spain under CICYT, TIC 98-0410, by ACiD-WG (ESPRIT-21949) and by Intel Corporation.

Like [7], the proposed multiplication scheme is based on a triangular array of counters with a variable delay. The variable execution time is obtained by making the computation to be data-dependent. Data dependency is obtained by means of a speculation function that tries to predict the result by assimilating a reduced number of input bits. This leads to a faster execution time. However, since the output of a counter is a speculation, it may be wrong and a correction of the result may be necessary. What makes the proposed approach better than other similar implementations is that, in our case, prediction and error detection and correction run in parallel and since the error-detection function is faster than the speculation function we may activate the correction phase before the speculated value is issued. As a consequence, in case of prediction error the correction logic does not introduce any overhead in the execution time. This leads to high speedups compared to other implementations.

The rest of the paper is organized as follows: in Section 2 we describe the implementation of a standard multiplier based on parallel counters and outline the differences between this standard implementation and the proposed multiplication scheme. Section 3 deals with the implementation details. Section 4 compares our design with other synchronous and asynchronous designs. Finally, in Section 5 we draw up some conclusions.

## 2 Design of Array-Multipliers Based on Parallel Counters

In this section we deal with a multiplication scheme based on parallel counters. The use of counters or compressors permits to decrease the execution time of a multiplication since the overall number of partial products to be added is reduced. We first describe the general architecture of an array-multiplier based on parallel counters and implementing the Baugh-Wooley algorithm. Next we will deal with the design approach we propose to achieve data-dependent computation times. Hence, we will introduce the performance metrics used to evaluate the design and focus our attention on some important design issues as well as on the design of counters with variable execution time.

### 2.1 Basic Multiplication Scheme

Figure 1 shows the architecture of a  $8 \times 8$  multiplier [7] that implements the product  $A \times B$ , with  $A = (a_7, a_6, \dots, a_0)$ ,  $B = (b_7, b_6, \dots, b_0)$  and  $a_i, b_i \in \{0, 1\}$ , using the Baugh-Wooley algorithm [1]. The array topology is triangular so that the most and the least significant halves of the product can be computed in parallel. The most significant digits  $p_i$ 's of the result are generated in redundant carry-save form and fed into the on-the-fly conversion (OTFC) unit that operates in parallel with the multiplier. The OTFC unit converts the result from redundant into conventional representation [12]. The dashed line separates the variable-delay part of the multiplier from the interface logic to the OTFC unit and from the logic that generates the least significant part of the result. Each element of the array is an  $(m, n)$  parallel counter [10], that is an arithmetic circuit whose inputs are  $m$  bits of

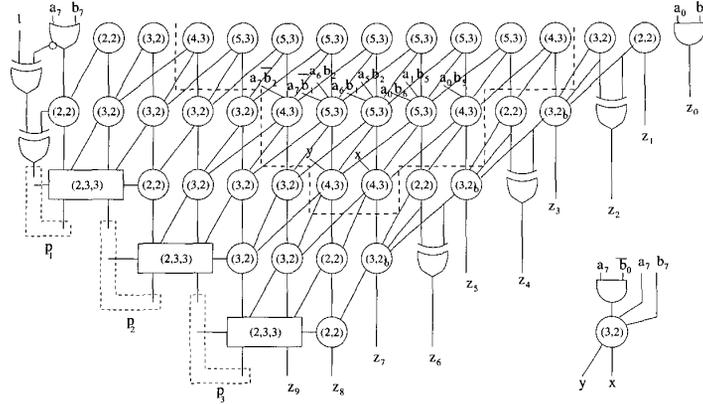


Figure 1. Array Implementation of the  $8 \times 8$  Multiplier.

weight  $2^w$  and whose  $n$  outputs ( $n = \lceil \log_2(m+1) \rceil$ ) are the bits with weights from  $2^w$  to  $2^{w+n-1}$  and represent the arithmetic sum of the input bits. Counters are widely used in array-multiplication since they permit to reduce the overall number of partial products to be summed, thus reducing the execution time of the algorithm. Referring to Figure 1,  $(3,2)_b$  and  $(2,3,3)$  denote a *binary full-adder* and a *2-bit full-adder* respectively. Each redundant digit  $p_i$  of the  $b \times b$  multiplier is composed of  $\log_2 r + 1$  bits. The  $i$ -th radix-4 redundant digit  $p_i$  is formed by three bits: we denote by  $s_{i,1}$  and  $c_{i,1}$  the two of them with weight  $2^{2b-2i+1}$  and by  $c_{i,0}$  the one with weight  $2^{2b-2i}$ . The assimilation of these bits produces a  $p_i \in [0, 5]$ , that must be converted on-the-fly into a non-redundant radix-4 digit. The OTFC algorithm is described in detail in [8] and is derived from the one described in [7].

## 2.2 Area and Time Performance Criteria

The designs are implemented using a set of full-custom CMOS CPL cells. Delays and areas are represented as multiples of the delay and area of a two-input XOR-XNOR gate. To make simulation realistic, parasitic resistances and capacitances were extracted and worst-case RC load due to wiring was estimated by assuming a draft layout plan [14]. Routing area was estimated as well by assuming a pitch of  $5\lambda$  between two adjacent METAL I wires and taking into account that a wire width is  $2\lambda$ .

## 2.3 Achieving Data-Dependent Delays

We propose a multiplier that produces a result with a telescopic delay [3]. Data-dependency is achieved by means of a prediction function. Let us consider, for the sake of simplicity, the case of a  $(5,3)$  counter; the extension to other types of counters is straightforward. A  $(5,3)$  counter has five inputs and three outputs; let  $\underline{y} = (y_4, \dots, y_0)$  be the input vector and  $\underline{z} = (z_2, z_1, z_0)$  the output vector. To shorten the execution time, we choose to compute the sum by assimilating a reduced number of bits. In the case of a  $(5,3)$  counter we use bits  $(y_4, y_3, y_2)$  to predict the result and bits  $(y_1, y_0)$  for error detection and correction. We may define the prediction function as the function  $F^P(y_4, y_3, y_2) = \sum_{i=2}^4 y_i + \sigma$  where  $\sigma$  is prediction of the value of the sum  $y_1 + y_0$  of the discarded bits. The speculated sum  $\sigma$  is derived by taking into account the statistical distribution of the values of the sum  $y_1 + y_0$ . Since  $0 \leq y_1 + y_0 \leq 2$ , our goal is finding a small interval  $[\sigma_1, \sigma_2]$  so as to:

1. maximize  $P(\sigma_1 \leq y_1 + y_0 \leq \sigma_2)$ ;
2. obtain a fast and simple implementation.

Unfortunately these requirements are mutually exclusive, that is, the higher the number of correct predictions, the more complex

the prediction function is. As a consequence a tradeoff between number of correct predictions and hardware complexity must be found. In [8] it has been demonstrated that, by assuming a uniform distribution of the operands, the probability  $P(y_i = 0)$  for the counters of the first and second row of the  $8 \times 8$  multiplier of Figure 1 is the one reported in Table 1. According to these

row	$P(y_1 = 0)$	$P(y_0 = 0)$
1	0.75	0.75
2	0.75	0.984

Table 1. Probabilities of the Discarded Bits for the  $(5,3)$  Counters.

values, we found that  $P(y_1 + y_0 \leq 1) > 0.9$  for the counters of the first two rows [8]. Thus a very good choice could be  $\sigma_1 = 0$  and  $\sigma_2 = 1$ . This also means that  $\sum_{i=0}^4 y_i$  is more likely to be  $\sum_{i=2}^4 y_i$  or  $\sum_{i=2}^4 y_i + 1$ . As a consequence, to each combination of the  $(y_4, y_3, y_2)$  we may associate two possible values, namely  $\sum_{i=2}^4 y_i$  or  $\sum_{i=2}^4 y_i + 1$ , as we will see more in detail in section 2.4. This higher degree of freedom allows us to synthesize prediction functions with a very high number of correct predictions and a short execution time as it will be shown in the next section. The simulations performed have shown that  $P(y_1 + y_0 \leq 1) > 0.9$  also for  $(5,3)$  counters of larger multipliers.

## 2.4 Variable-Delay Counters

Figure 2 depicts a possible realization of a  $(5,3)$  counter. This realization is not the one used in this paper. It is explained to ease the description of the actual approach. In this scheme, that reminds a conditional sum adder [16], all the possible sum values are computed in parallel using bits  $y_4, y_3, y_2$ , whereas bits  $y_1$  and  $y_0$  are used to select the correct value. The proposed scheme is

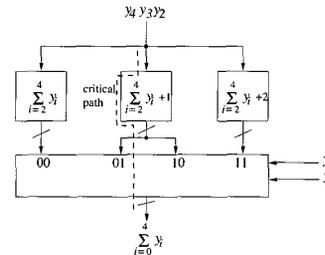


Figure 2. A Possible Implementation of a  $(5,3)$  Counter.

very similar to that of Figure 2, but where only the sum values more

likely to happen are computed. This choice is made according to statistical considerations. Since not all of the possible sums are computed, what is made in reality is a speculation of the result and hence a correction may be necessary if the prediction is wrong. Since this class of counter exhibits a variable-delay behaviour we call it *variable-delay counter* (VDC).

#### 2.4.1 Boolean Relations for Fast Prediction

Let us consider the case of a (5,3) VDC. Prediction is carried out by function  $F^P(y_4, y_3, y_2) = \sum_{i=2}^4 y_i + \sigma$ , where  $\sigma = \sigma(y_4, y_3, y_2)$  is a function that returns an integer value such that  $\sigma \in [0, 1]$ . This means that for every combination of  $(y_4, y_3, y_2)$  we may choose between two predictable values, namely, we may choose to associate to  $F^P(y_4, y_3, y_2)$  either  $\sum_{i=2}^4 y_i$  or  $\sum_{i=2}^4 y_i + 1$ , this may be specified by using *Boolean relations* [5]. Boolean relations allow a higher degree of freedom during the synthesis since they permit to associate to each combination of the input variables of a boolean function  $f$  several valid output values. Among all the possible output values the one that produces the  $f$  with the simplest implementation is chosen. However a "classical" approach to prediction and correction, like in [9] does not produce great improvements of the average execution time, since only a value at a time is predicted and the rate of correct predictions may be not sufficiently high. But if we anticipate the computation of the value not predicted by  $F^P$  by means of other functions that work in parallel, two predictable values at a time will be available. This will result in a higher rate of correct predictions and hence in a smaller average delay of the counter. This task is carried out by functions  $F_{-1}^P(y_4, y_3, y_2) = \sum_{i=2}^4 y_i + \sigma - 1$  and  $F_{+1}^P(y_4, y_3, y_2) = \sum_{i=2}^4 y_i + \sigma + 1$ . In addition function  $\sigma(y_4, y_3, y_2)$  is necessary to identify which value has been predicted by function  $F^P(y_4, y_3, y_2)$  in order to perform the selection among the predicted values and the correction, in case of wrong prediction. A correct choice of  $\sigma_1$  and  $\sigma_2$  is crucial since it determines:

1. the number  $(\sigma_2 - \sigma_1) + 1$  of possible output combinations of  $F^P$  that may be associated to each combination of the input variables by means of boolean relations;
2. the number  $\lceil \log_2(\sigma_2 - \sigma_1) \rceil + 1$  of bits necessary to encode  $\sigma$ ;
3. the number  $2(\sigma_2 - \sigma_1)$  of functions  $F_j^P$ 's.

Finally, Figure 2 shows the truth table of the prediction function. The central column reports the output specification using boolean relations, whereas the rightmost one shows the selection performed by the boolean relation solver [17].

$y_4 y_3 y_2$	$z_2 z_1 z_0 \sigma$	
	Possible Outputs	Selected
000	{0000, 0011}	0000
001	{0010, 0101}	0101
010	{0010, 0101}	0101
011	{0100, 0111}	0100
100	{0010, 0101}	0010
101	{0100, 0111}	0111
110	{0100, 0111}	0111
111	{0110, 1001}	0110

Table 2. (5,3) VDC Prediction Function Truth Table for the Boolean Relation.

#### 2.4.2 Architecture

Figure 3(a) depicts the basic scheme of (5,3) VDC. The prediction function  $F^P = \sum_{i=2}^4 y_i + \sigma$  operates in parallel with functions  $F_{-1}^P = \sum_{i=2}^4 y_i + \sigma - 1$ ,  $F_{+1}^P = \sum_{i=2}^4 y_i + \sigma + 1$  and  $\sigma$ . Thus

function  $F_{-1}^P$  handles the case in which  $F^P = \sum_{i=2}^4 y_i + 1$  and  $y_0 + y_1 = 0$  whereas function  $F_{+1}^P$  handles the one in which  $F^P = \sum_{i=2}^4 y_i$  and  $y_0 + y_1 = 1$ . Since  $\sigma \in [0, 1]$ , the only combination of  $y_1$  and  $y_0$  that produces a prediction error is  $(y_1 y_0) = (1, 1)$ , that is the one such that  $y_1 + y_0 = 2 \notin [\sigma_1, \sigma_2]$ , hence the error detection function  $e(y_1, y_0)$  is simply  $e = y_1 \wedge y_0$  and the correction function is  $F^c(F^P, \sigma) = F^P + (2 - \sigma)$ . Input combinations  $(y_1, y_0) \in \{(0, 0), (0, 1), (1, 0)\}$  identify a good prediction. The case in which  $y_1 + y_0 = 0$  is managed by function  $s_0(y_1, y_0) = (y_1 \vee y_0)'$ , whereas the case  $y_1 + y_0 = 1$  is managed by function  $s_1(y_1, y_0) = y_1 \oplus y_0$ . For example, considering the scheme of Figure 3(a) and the simplified truth table of Figure 3(b),  $F^P$  is selected if and only if  $s_0 = \sigma' = 1$  or  $s_1 = \sigma = 1$ . According to these considerations, we deduce that a (5,3) VDC performs the following selection:

$$\sum_{i=0}^4 y_i = \begin{cases} F^P & \text{if } (\sigma \wedge s_1) \vee (s_0 \wedge \sigma') \\ F_{+1}^P & \text{if } (s_1 \wedge \sigma') \\ F_{-1}^P & \text{if } (s_0 \wedge \sigma) \\ F^c & \text{if } e \end{cases} \quad (1)$$

From equation (1) and Figure 3 we deduce that bits  $y_1, y_0$  and  $\sigma$  permit to switch between two different delays: a short delay in case of correct prediction, and a long delay in case of wrong prediction. Table 3 shows the boolean equations of a (5,3) VDC. The equations of all the designed VDCs may be found in [8].

$\sigma$	$y_3 \oplus y_2$
$F^P$	$(0, y_3 \vee y_2, y_4)$
$F_{-1}^P$	$(0, y_4, y_4')$
$F_{+1}^P$	$(y_4 \wedge y_2, y_4 \oplus y_2, y_4')$
$F^c \dagger$	$(p_1 \wedge (\sigma' \vee p_0), (\sigma \wedge p_0') \vee p_1', \sigma \oplus p_0)$

$\dagger$  We define  $F^P$  as  $F^P = (p_2, p_1, p_0)$ .

Table 3. (5,3) VDC Prediction and Correction Functions.

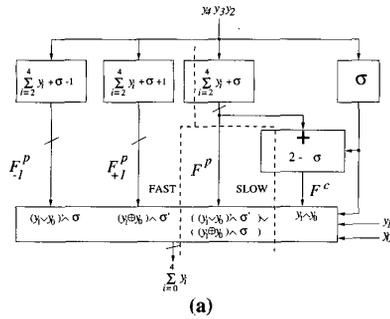
### 3 Implementation

We have designed a radix-4 array for multiplication. All the basic processing elements are implemented using elementary CPL gates [2] since they are faster and less power consuming than standard CMOS logic. The designed gates have been modified to permit using a  $0.8\mu\text{m}$  and modified to permit the precharge of the output nodes. The whole design has been simulated using HSPICE. Prediction functions have been synthesized using *Boolean relations* [17], in order to produce a minimum cost function. In [8] we also prove that the designed cells have a monotonic behaviour. This is crucial to assure the correctness of the dual-rail protocol [6].

To give technology-independent estimations, delays and areas are normalized with respect worst-case delay and area of an unloaded XOR-XNOR gate. Area estimations take into account the area due to interconnections among the cells. It must be also pointed out that the proposed scheme, as well as the one described in [7], produces the most significant half of the result in redundant carry-save form. Hence the result is converted on-the-fly into non-redundant form as described in [8].

#### 3.1 Synchronization and Timing

Figure 4 shows a sketch of implementation. In Figure 5 is reported the four-phase handshake protocol of the proposed multiplication scheme as well as its reset time ( $2t_{XOR}$ ). As long as the input operands are not valid, the circuit is held in the precharge state, forcing the outputs of all the VDCs to logical "0". When the operands are valid the  $P/E'$  signal goes down and the circuit enters the evaluation phase. The completion detection logic (whose operation is described in section 3.2) forces  $RDY$  high, as soon



$y_4 y_3 y_2$	$\sum_{i=2}^4 y_i + \sigma - 1$	$\sum_{i=2}^4 y_i + \sigma$	$\sum_{i=2}^4 y_i + \sigma + 1$	$\sigma$
000	---	000	001	0
001	001	010	---	1
010	001	010	---	1
011	---	010	011	0
100	---	001	010	0
101	010	011	---	1
110	010	011	---	1
111	---	011	100	0

Figure 3. A (5,3) VDC: (a) Basic Scheme, and (b) Prediction Function Simplified Truth Table.

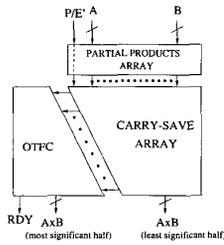


Figure 4. Sketch of Implementation.

as a valid result has been produced. *RDY* goes down when the input operands are no longer valid, that is when the circuit is forced in the precharge state by *P/E'* going high. To generate a correct *RDY* signal efficiently and using a reduced amount of hardware resources, the carry-save array and the OTFC must be synchronized. To achieve this we exploit the fact that the carry-save array generates the result digits sequentially from the most to the least significant one. As a consequence we may delay the last stage of the OTFC unit keeping it precharged, until the least significant digit is valid. All the implementation details may be found in [8].

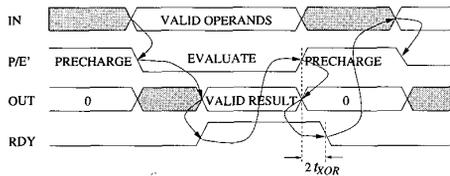


Figure 5. Timing.

### 3.2 Completion Detection

To detect the end of a multiplication we have to detect the end of the conversion into non-redundant form. This is done using a completion detection scheme based on the one proposed in [15] and depicted in Figure 6. In the precharge phase the *RDY* output is set to "0" and, since the differential outputs  $m_i$  and  $m'_i$  of the OTFC are NORed, all the n-mos pull-downs conduct, keeping the output low. In the evaluation phase, as soon as the last differential output is valid, all the pull-downs will be disabled and the p-mos pull-up will push the *RDY* output at logical "1". The extra latency introduced by the completion detection logic is only  $t_{XOR}$ ,  $t_{XOR}$  being the delay of a XOR gate.

### 3.3 Extension to Higher Radices

Along this paper we have dealt exclusively with (5,3)-VDCs as basic processing elements of multiplicative arrays of several sizes. However the proposed design approach may be extended to higher radices. [8] reports also the boolean equations for (6,3) and (7,3) VDCs used to implement radix-8 and radix-16 multipliers. We must remark that, in the proposed multiplication scheme, each row of the carry-save array that computes the least significant half of the result must generate  $\log_2 r$  bits in carry-assimilated form. This task is performed by a fast carry-propagation adder. For high values of the radix  $r$ , the latency introduced by the adder may be critical, restricting our design methodology to radices not bigger than 16.

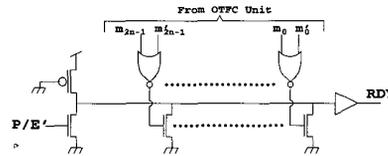


Figure 6. Completion Detection Logic.

## 4 Comparisons

We will compare the designs both in terms of delay and in terms of area, even if the implementation described in [13] does not require any supplementary hardware for the on-the-fly conversion as it is done in [7] and in the multiplication scheme presented in this paper. In fact, the use of an OTFC unit requires extra area respect to a design like [13] that uses an adder at the final stage of the array. In order to give coherent estimations, all the designs have been implemented using the same technology and CPL gates. For the same reason we will limit our analysis to the radix-4 case, that is the case of the multiplication scheme of [7]. On the other hand, although the implementation proposed in [13] is a radix-2 scheme we will compare our design all the same, since with our approach is impossible to synthesize data-dependent counters smaller than (4,3) whereas the approach described in [13] is only suitable for radix-2 arrays. Delays and areas will be expressed as multiples of the delay and area of a two-input XOR-XNOR gates. The average delays of the asynchronous implementations have been computed by simulating  $10^4$  multiplications with randomly-generated operands and assuming a uniform distribution.

### 4.1 Delay Estimation

The proposed multiplication scheme has a delay  $\Delta_{VDCM}$  (where *VDCM* stands for variable-delay counter multiplier) that is the sum of four contributions:  $\Delta_{VDCM} = t_{array} + t_{ofc} + n t_{buf} + t_{sync}$ , where  $t_{array}$  is the latency of the array,  $t_{ofc}$  the latency due to the conversion logic,  $n \in [0, 3]$  is the number of buffers (with delay  $t_{buf}$ ) necessary to drive the stage of the OTFC unit

situated in the critical path, and  $t_{sync}$  the time overhead introduced by synchronization logic. The delay  $\Delta_{CM}$  of the architecture described in [7] is  $\Delta_{CM} = t_{array, CM} + t_{otfc} + n t_{buf}$ , whereas the delay  $\Delta_{KB}$  of the scheme described in [13] is  $\Delta_{KB} = t_{array, KB} + t_{MADD}$ ,  $t_{MADD}$  being the delay of a  $b$ -bit Manchester carry-resolution adder. According to [8] we may estimate the delay of the manchester adder as  $t_{MADD} = 1.22 \frac{(7+b)}{2} t_{XOR}$ , 1.22 being a coefficient that takes into account the wiring delay and that has been determined by simulation assuming realistic gate loads. The carry-save part of the scheme described in [13] has a minimum delay of  $1.51 t_{XOR}$  and a maximum delay of  $4.13 t_{XOR}$ . The array depth of both our multiplication scheme and the one proposed in [7] may be computed as shown in [8], whereas for the combinational scheme described in [13] is equal to the operand size. The conversion time or, in case of the multiplication scheme proposed in [13] the delay of a Manchester adder, has to be added to the array delay. To our implementation and to the one described in [13] must also be added an additional delay of  $2 t_{XOR}$  due to completion detection logic. Table 4 re-

$b$	$n$	Delay			Speedup	
		VDCM	[CM96]	[KB97]	[CM96]	[KB97]
8	0	25.51	33.71	33.71	1.32	1.32
16	1	42.86	59.76	61.15	1.40	1.43
24	2	59.79	85.81	88.59	1.43	1.48
32	3	76.69	111.86	121.13	1.45	1.57

Table 4. Normalized Average Delays of the Multipliers.

ports the delays of all the implemented multipliers as well as the speedups measured. From Table 4 we deduce that the proposed implementation has a speedup  $S$  that goes from 1.32 to 1.45 with respect the synchronous realization described in [7] and a speedup ranging from 1.32 to 1.57 when compared with the asynchronous realization reported in [13]. The performance increase depends on the size  $b$  of the operands. The larger is  $b$  the larger is the measured speedup. Experimental results reported in Figure 4 are represented graphically in Figure 7.

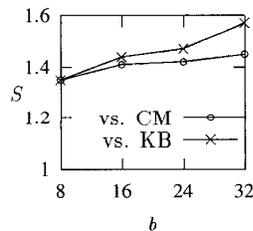


Figure 7. Speedup Versus Operands Size for the Designed Multipliers.

## 4.2 Area Estimation

In this section we evaluate the area occupation of both the OTFC unit and the multiplying array. Table 5 shows the overall areas of the implemented multiplication schemes obtained considering the areas of the array for partial products generation, the array of basic processing elements and the conversion unit or the final adder in case of [13]. Areas are expressed as multiples of the area of a XOR-XNOR gate.

## 5 Conclusions

Variable-delay counters (VDCs) are the basis for the design of the asynchronous multiplier presented in this paper. The trade-off between prediction accuracy and calculation speed has been explored by using *Boolean relations* for the specification and minimization of the behaviour of VDCs. This has resulted in a novel

$b$	Overall Area		
	VDCM	[CM96]	[KB97]
8	654.7	491.9	559.2
16	3109.6	2151	2116.8
24	7551.6	5144.1	4672.8
32	14068.4	9516.1	8227.2

Table 5. Normalized Overall Areas of the Implemented Multipliers.

design that for the first time, to the knowledge of the authors, outperforms the efficiency of synchronous multipliers.

Counters are highly used in arithmetic circuits. We think we have only presented one possible application and that the use of variable-delay building blocks and the exploration of implementations by means of Boolean relations can be combined in many other areas that may benefit from the average-case performance offered by asynchronous circuits.

## References

- [1] C. R. Baugh and R. A. Wooley. A Two's Complement Parallel Array Multiplication Algorithm. *IEEE Transaction on Computers*, C-22(12):1045–1047, December 1973.
- [2] A. Bellaouar and M. I. Elmasry. *Low-Power Digital VLSI Design*. Kluwer Academic Publisher, Norwell, MA, 1995.
- [3] L. Benini, G. De Micheli, A. Liyo, et al. Automatic Synthesis of Large Telescopic Units Based on Near-Minimum Timed Supersetting. *IEEE Transaction on Computers*, C-48(8):769–779, August 1999.
- [4] A. D. Booth. A Signed Binary Multiplication Technique. *Quarterly Journal Mech. Appl. Math.*, 4, Part2:236–240, 1951.
- [5] R. K. Brayton and F. Somenzi. Boolean Relations and the Incomplete Specification of Logic Networks. In *Int. Conf. Very Large Scale Integration*, 1989.
- [6] J. A. Brzozowski and C.-J. H. Seger. *Asynchronous Circuits*. Springer-Verlag, New York, 1994.
- [7] L. Ciminiera and P. Montuschi. Carry-Save Multiplication Schemes Without Final Addition. *IEEE Transaction on Computers*, C-45(9):1050–1055, September 1996.
- [8] G. Cornetta. *Design and Analysis of Variable-Delay Arithmetic Units*. PhD thesis, Polytechnic University of Catalonia-Dept. of Computer Architecture, Barcelona, September 2001.
- [9] J. Cortadella and T. Lang. High-Radix Division and Square Root with Speculation. *IEEE Transaction on Computers*, C-43(8):919–931, August 1994.
- [10] L. Dadda. Some Schemes for Parallel Multipliers. *Alta Frequenza*, 34:349–356, March 1965.
- [11] A. De Gloria and M. Olivieri. Statistical Carry Lookahead Adders. *IEEE Transaction on Computers*, C-45(3):340–347, March 1996.
- [12] M. D. Ercegovic and T. Lang. On the Fly Conversion of Redundant into Conventional Representations. *IEEE Transaction on Computers*, C-36(7):895–897, July 1987.
- [13] D. Kearney and N. W. Bergmann. Bundled Data Asynchronous Multipliers with Data Dependent Computation Times. In *3rd IEEE Symposium on Advanced Research in Asynchronous Circuit and Systems*, pages 186–197, 1997.
- [14] G. Matsubara and N. Ide. A Low Power Zero-Overhead Self-Timed Division and Square Root Unit Combining a Single-Rail Static Circuit with a Dual-Rail Dynamic Circuit. In *Symposium on Advanced Research in Asynchronous Circuit and Systems*, pages 198–209, 1997.
- [15] S. M. Nowick, K. Y. Yun, P. A. Beerel, and A. E. Dooply. Speculative Completion for the Design of High-Performance Asynchronous Dynamic Adders. In *Symposium on Advanced Research in Asynchronous Circuit and Systems*, pages 210–223, 1997.
- [16] J. Sklansky. Conditional Sum Addition Logic. *IRE Transaction on Electronics Computers*, EC-9:226–231, June 1960.
- [17] Y. Watanabe and R. E. Brayton. Heuristic Minimization of Multiple-Valued Relations. *IEEE Transaction on Computer-Aided Design of Integrated Circuits*, 12(10):1458–1472, October 1993.
- [18] T. E. Williams and M. Horowitz. A 160ns 54-bit CMOS Division Implementation Using Self-Timing and Symmetrically Overlapped SRT Stages. In *10th IEEE Symposium on Computer Arithmetic*, pages 210–217, 1991.