

We observe that the rotation cell-based MIMO LBR system drawn in Fig. 6 takes a regular structure. Due to this regularity, the MIMO LBR system can be conveniently implemented in VLSI circuits. Each rotation cell in this structure can be efficiently implemented by using CORDIC processors [4].

A further work is called for examining the relationship among various realization structures of MIMO LBR systems.

#### REFERENCES

- [1] A. Fettweis, "Wave digital lattice filters," *Int. J. Circuit Theory Applicat.*, vol. 2, pp. 203–211, June 1974.
- [2] E. Deprettere and P. Dewilde, "Orthogonal cascade realization of real multiport digital filters," *Int. J. Circuit Theory Applicat.*, vol. 8, pp. 245–277, July 1980.
- [3] D. Henrot and C. T. Mullis, "A modular and orthogonal digital filter structure for parallel processing," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Boston, MA, Apr. 1983, pp. 623–626.
- [4] S. K. Rao and T. Kailath, "Orthogonal digital filters for VLSI implementation," *IEEE Trans. Circuits Syst.*, vol. CAS-31, pp. 933–945, Nov. 1984.
- [5] P. P. Vaidyanathan and S. K. Mitra, "A general family of multivariable digital lattice filters," *IEEE Trans. Circuits Syst.*, vol. CAS-32, pp. 1234–1245, Dec. 1985.
- [6] U. B. Desai, "A state-space approach to orthogonal digital filters," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 160–169, Feb. 1991.
- [7] J. B. Lee and B. G. Lee, "Transform domain filtering based on pipelining structure," *IEEE Trans. Signal Processing.*, vol. 40, pp. 2061–2064, Aug. 1992.
- [8] D. Y. Kim and B. G. Lee, "Transform domain IIR filtering," *IEEE Trans. Signal Processing.*, vol. 43, pp. 2431–2434, Oct. 1995.
- [9] D. C. Youla, "On the factorization of rational matrices," *IRE Trans. Inform. Theory*, pp. 172–189, July 1961.
- [10] P. Dewilde, "Input–output description of roomy systems," *SIAM J. Contr. Optimizat.*, vol. 14, pp. 712–736, July 1976.

## A VLSI Design Methodology for RNS Full Adder-Based Inner Product Architectures

D. J. Soudris, V. Paliouras, T. Stouraitis, and C. E. Goutis

**Abstract**—In this paper, a systematic graph-based methodology for synthesizing VLSI RNS architectures using full adders as the basic building block is introduced. The design methodology derives array architectures starting from the algorithm level and ending up with the bit-level design. Using as target architectural style the regular array processor, the proposed procedure constructs the two-dimensional (2-D) dependence graph of the bit-level algorithm, which is formally described by sets of uniform recurrent equations. The main characteristic of the proposed architectures is that they can operate at very high-throughput rates. The proposed architectures exhibit significantly reduced complexity than ROM-based ones.

**Index Terms**—Bit-level design methodology, inner-product processor, residue number system.

### I. INTRODUCTION

Many systematic methodologies have been proposed for mapping digital signal processing (DSP) algorithms onto regular array processors and, particularly, on systolic arrays [1]–[3]. However, these methodologies do not exploit any inherent parallelism and pipelining of the algorithm at the bit level. Several research attempts have been reported for the design of bit-level arrays [4]–[6]. McCanny *et al.* [4] presented an approach for the systematic design of bit-level systolic arrays exploiting a methodology developed by Kung [1]. However, the arithmetic is not considered as an individual parameter. Shang and Fortes [5] presented a methodology for mapping multidimensional nested algorithms to processor arrays of lower dimension. However, they do not exploit the design alternatives that are available at the bit level. Burleson and Scharf presented a VLSI design methodology for multiply/add units operating in distributed arithmetic [6]. Their design approach is based on a parameterized dependence graph, which leads to tree or regular array architectures.

In this paper, a systematic methodology for designing inner product step processors (IPSP's) based on residue number system (RNS) is presented. Generally, an IPSP consists of an accumulator and a multiplier. When it is adopted for arithmetic in a finite integer ring  $\mathbb{R}(m)$  [7], this unit is called IPSP<sub>*m*</sub>. The performed operation is of great importance for DSP algorithms since they include "inner-product" type of operations [1]. Because of its efficiency in sum-of-products calculations, RNS can execute DSP algorithms at higher rates than other arithmetic systems [7]. Taking into account the target architectural style and the specific value of modulo *m*, the proposed methodology produces the two-dimensional (2-D) graph of the inner-product algorithm, which is specified through a series of lemmas and several sets of *uniform recurrent equations* (URE's) [1]. The dependence graph (DG) is then mapped onto the selected array processor style. The proposed architectures for the IPSP<sub>*m*</sub> are

Manuscript received January 13, 1995; revised July 19, 1996. This work was supported in part by the European Community under the ESPRIT Basic Research Action 3281 (ASCIS).

D. J. Soudris is with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi 67 100, Greece.

V. Paliouras, T. Stouraitis, and C. E. Goutis are with the Department of Electrical and Computer Engineering, University of Patras, Patras 26110, Greece.

Publisher Item Identifier S 1057-7130(97)02738-9.

based on full adder (FA) arrays, which outperform conventional look-up table based structures [8], when very high throughput rates are required. As it is proved, a 2-D FA-based array can produce an inner product in a time interval equal to one pipelined FA cell latency.

## II. THE PROPOSED ARCHITECTURE

In RNS arithmetic any integer  $X$  is represented by an  $N$ -tuple  $\{X_0, X_1, \dots, X_{N-1}\}$ , with  $X_i = \langle X \rangle_{m_i}$ , where  $m_i$  is an element of the base  $\{m_0, m_1, \dots, m_{N-1}\}$ , which contains relatively prime integers, with  $0 \leq X < \prod_i m_i$ . In general, the notation  $\langle F \rangle_m$  denotes the operation  $F$  modulo  $m$ . Any computation requires  $N$ -parallel channels, each one performing the inner product modulo  $m_i$  in a finite-integer ring  $\mathbb{R}(m_i)$ ,  $i = 0, 1, \dots, N-1$  [7]. The proposed methodology addresses the design of an IPSP for each channel, denoted  $\text{IPSP}_m$  for simplicity. Its function can be described by [7], [8]

$$Y_{out} = \langle Y + \langle A \times X \rangle_m \rangle_m \quad (1)$$

where  $Y_{out}$ ,  $Y$ ,  $A$ , and  $X \in \mathbb{R}(m) = \{0, 1, \dots, m-1\}$ . Let  $Y$ ,  $A$ , and  $X$  be defined as  $Y = \sum_{c=0}^{n-1} y_c 2^c$ ,  $A = \sum_{c=0}^{n-1} a_c 2^c$ ,  $X = \sum_{c=0}^{n-1} x_c 2^c$ , where  $n = \lfloor \log_2 m \rfloor + 1$ ,  $y_c$ ,  $a_c$ , and  $x_c \in \{0, 1\}$ . If we define  $Y_0$  to be

$$Y_0 = \sum_{c=0}^{n-1} y_c 2^c + \sum_{c=0}^{n-1} \sum_{d=0}^{n-1} a_c x_d \langle 2^{c+d} \rangle_m. \quad (2)$$

Equation (1) gives

$$Y_{out} = \left\langle \sum_{c=0}^{n-1} y_c 2^c + \sum_{c=0}^{n-1} \sum_{d=0}^{n-1} a_c x_d \langle 2^{c+d} \rangle_m \right\rangle_m = \langle Y_0 \rangle_m. \quad (3)$$

Three stages of computations are required for the calculation of  $Y_{out}$ . The first stage computes  $Y_0$ , which is defined by (2) and can be written in binary form as  $Y_0 = \sum_{i=0}^{\ell-1} y_{0,i} 2^i$  with  $\ell = \lfloor \log_2 (m-1 + \max\{\sum_{c=0}^{n-1} \sum_{d=0}^{n-1} a_c x_d \langle 2^{c+d} \rangle_m\}) \rfloor + 1$ . The second stage computes  $Y_r$ , which has the properties  $\langle Y_r \rangle_m = Y_{out}$  and  $n = \lfloor \log_2 Y_r \rfloor + 1$ , while the third stage maps  $Y_r$  to  $Y_{out}$ .

More specifically, the function of the second stage is based on the modulo arithmetic property

$$\left\langle \sum_i b_i 2^i \right\rangle_m = \left\langle \sum_i b_i \langle 2^i \rangle_m \right\rangle_m \quad (4)$$

where  $b_i \in \{0, 1\}$ . This stage recursively replaces any bit that corresponds to a power of two that is larger than  $2^n - 1$  to a set of bits that correspond to powers of two whose sum is less than or equal to  $2^n - 1$ . This operation may be repeated as needed to eventually reduce the number of bits of  $Y_0$  from  $\ell$  to  $n$  bits that are required for  $Y_{out}$ . Therefore, except from the case where  $Y_0 \leq 2^n - 1$ , at least one usage of (4), called hereafter a "recursion," is required.

Assuming that the  $(k-1)$ st,  $0 < k \leq r$ , recursion has as output  $Y_{k-1} = \sum_{i=0}^{n_{k-1}-1} y_{k-1,i} 2^i$ . Then, the output of the  $k$ th recursion is obtained recursively from  $Y_{k-1}$

$$Y_k = \sum_{i=0}^{n_k-1} y_{k,i} 2^i = \sum_{i=0}^{n_{k-1}-1} y_{k-1,i} \langle 2^i \rangle_m. \quad (5)$$

The maximum required word length of the output of each recursion  $n_k$  and the number of the required recursions  $r$  can be calculated

using the following algorithm:

```

k = 0
maxout0 = maxfirst; //maxfirst is the
//maximum output of the first stage
n0 = ℓ
while (nk > n) do {
  k = k + 1
  maxoutk = 0
  for x = 0 to maxoutk-1 do
    {   for i = 0 to nk-1 - 1 do
        {   yk-1,i = (Yk-1 AND 2i) DIV 2i
            //AND denotes the bitwise logic
            //operator that returns an integer
            //and DIV denotes the integer
            //division
            {Yk = ∑i=0nk-1-1 Yk-1,i ⟨2i⟩m
              if Yout > maxoutk then maxoutk = Yk}
            nk = ⌊log2 {maxoutk}⌋ + 1}
    }
  }
r = k

```

Finally, the output  $Y_r$  of the second stage is mapped during a third stage (or output stage) to its residue modulo  $m$  value. This mapping can be achieved by using one  $n$ -bit adder.

Since (3) requires a total of three stages, the proposed architecture of the  $\text{IPSP}_m$  consists of three blocks. Moreover, due to the fact that (3) falls into the class of algorithms called *weak single assignment codes* [3], each block can be implemented as an array processor.

## III. THE DESIGN OF THE FA-BASED ARRAYS

The detailed design procedure of each of the FA-based stages is obtained by three steps: 1) Calculation of the number of FA's; 2) Derivation of the DG; and 3) the design of the output stage.

### A. Calculation of the Number of FA's

To specify the number of FA's required for the computation of the  $i$ th output bit of the  $k$ th recursion,  $y_{k,i}$ , the number of the input bits of this computation should be calculated.

The input bits of the FA's of the first stage comprise the products  $a_c x_d$ , which can be computed by  $n^2$  AND gates, and the bits  $y_c$  of (2). Each term  $\langle 2^{c+d} \rangle_m$  of (2) can be expressed in a binary form as a sum of powers of two, where the bits  $q_{i,c,d}^{(0)}$  are defined through

$$\langle 2^{c+d} \rangle_m = \sum_{i=0}^{n-1} q_{i,c,d}^{(0)} 2^i \quad (6)$$

or  $q_{i,c,d}^{(0)} = (\langle 2^{c+d} \rangle_m \text{ AND } 2^i) \text{ DIV } 2^i$  and  $c, d = 0, 1, \dots, n-1$ . By substituting (6) into (2), we find

$$Y_0 = \sum_{i=0}^{n-1} \left\{ y_i + \sum_{c=0}^{n-1} \sum_{d=0}^{n-1} a_c x_d q_{i,c,d}^{(0)} \right\} 2^i. \quad (7)$$

Equation (7) implies that the  $i$ th output bit  $y_{0,i}$  is the sum of the bits  $y_i$  and  $a_c x_d q_{i,c,d}^{(0)}$  for  $c, d = 0, 1, \dots, n-1$ . The value of  $q_{i,c,d}^{(0)}$  being 1 or 0 specifies whether the bits  $a_c x_d$  contribute to  $y_{0,i}$  or not, respectively. Therefore, the set  $Z_{0,1} = \{a_c x_d | c, d = 0, 1, \dots, n-1 \wedge q_{i,c,d}^{(0)} = 1\}$ , with  $i = 0, 1, \dots, n-1$ , contains only the products  $a_c x_d$ , which contribute to the  $i$ th output bit.

Consequently, the number of input bits  $\alpha_{0,i}$  required for the computation of the  $i$ th output bit of the first stage can be specified by

$$\alpha_{0,1} = 1 + \sum_{c=0}^{n-1} \sum_{d=0}^{n-1} q_{i,c,d}^{(0)}, \quad i = 0, 1, \dots, n-1 \quad (8)$$

where the unity in (8) is due to the  $y_i$  of (7).

TABLE I  
COMPARISON OF THE FA-BASED IPSP<sub>m</sub> ARCHITECTURES WITH THE ROM-BASED ONES. (AU=ARITHMETIC UNIT, GM=GENERAL MULTIPLIER).

Modulus	FA Complexity Breakdown					Total FAs	AND gates	MUXes	Transistors	ROM AUm	ROM GM	Delay FA
	First stage	Second stage			Third stage							
		First recursion	Second recursion	Third recursion								
5	9	3	3	0	3	18	9	3	572	189	378	14
13	23	6	4	0	4	37	16	4	1150	640	1280	21
17	46	12	5	5	5	73	25	5	2216	1485	2970	37
29	34	8	5	0	5	52	25	5	1628	1485	2970	26
37	71	10	6	6	6	99	36	6	3014	4260	8520	43
41	61	10	6	0	6	83	36	6	2566	4260	8520	35
53	65	12	10	6	6	99	36	6	3014	4260	8520	41
61	52	9	6	0	6	73	36	6	2286	4260	8520	33
73	83	11	7	7	7	115	49	7	3544	9233	18466	49
89	99	18	10	7	7	141	49	7	4272	9233	18466	52
97	121	20	7	7	7	162	49	7	4860	9233	18466	57
101	102	16	7	0	7	132	49	7	4020	9233	18466	45
109	100	14	9	7	7	137	49	7	4160	9233	18466	50
113	106	15	7	0	7	135	49	7	4104	9233	18466	47

Similarly, the input bits for the  $k$ th recursion,  $k = 1, \dots, r$ , of the second stage are those coefficients of  $\langle 2^c \rangle_m$  that can be calculated by expressing each  $\langle 2^c \rangle_m$  in a binary form as in

$$\langle 2^c \rangle_m = \sum_{i=0}^{n-1} q_{i,c}^{(k)} 2^i \quad (9)$$

or  $q_{i,c}^{(k)} = (\langle 2^c \rangle_m \text{ AND } 2^i) \text{ DIV } 2^i$ , where  $c = 0, 1, \dots, n_{k-1} - 1$  and  $q_{i,c}^{(k)} \in \{0, 1\}$ . Using (9) into (5), we obtain

$$Y_k = \sum_{i=0}^{n-1} \left\{ \sum_{c=0}^{n_{k-1}-1} y_{k-1,c} q_{i,c}^{(k)} \right\} 2^i. \quad (10)$$

The set  $Z_{k,i}$  is defined as  $\{y_{k-1,c} | c = 0, 1, \dots, n_{k-1}-1 \wedge q_{i,c}^{(k)} = 1\}$ . Thus, the number of input bits,  $\alpha_{k,i}$ , which should be added to obtain the  $i$ th output bit of the  $k$ th recursion is determined by

$$\alpha_{k,i} = \sum_{c=0}^{n_{k-1}-1} q_{i,c}^{(k)}, \quad i = 0, 1, \dots, n-1. \quad (11)$$

Having calculated the exact number of the input bits of each recursion, the required number of FA's can be computed. Let  $A_{k,i}$  be the number of FA's required for computing the  $i$ th output bit,  $y_{k,i}$ , of the  $k$ th recursion,  $0 \leq k \leq r$ . Also, let  $\beta_{k,i}$  be the total number of bits that should be added by the  $A_{k,i}$  FA's. The  $\beta_{k,i}$  bits include: 1) the  $\alpha_{k,i}$  bits as they were specified above and 2) the carry bits  $c_{k,i}$ , which result from the computation of  $y_{k,i-1}$ . Since each FA has only one carry,  $c_{k,i}$  equals the number  $A_{k,i-1}$  of FA's that compute the  $(i-1)$ st bit, where  $i = 0, 1, \dots, n_k - 1$  and  $c_{k,0} = 0$ . Therefore,

$$\beta_{k,i} = \alpha_{k,i} + A_{k,i-1} \quad (12)$$

where  $i = 0, 1, \dots, n_k - 1$  and  $A_{k,-1} = 0$ . The minimum required number of FA's is

$$A_{k,i} = \left\lceil \frac{\beta_{k,i} - 1}{2} \right\rceil, \quad \beta_{k,i} \geq 1. \quad (13)$$

### B. Derivation of the Dependence Graph

In this subsection, the  $A_{k,i}$  parameters are employed to derive a graph-based description of the IPSP<sub>m</sub> architectures. In the proposed methodology, the DG of the  $k$ th recursion, DG<sub>k</sub>, is derived. The structure of the DG can be expressed formally using a system of URE's [1]. Among the existing *target architectural styles*, the *regular array processor* [1] is selected for the implementation of the proposed IPSP<sub>m</sub>, whose each processing element is a simple 1-b FA.

Let  $D = \{(i, j) | i, j \in Z\}$  be a 2-D index space and  $Z$  be the set of integers. The system of URE's over the domain  $D$ , which describes the DG of the target architecture, is

$$\begin{aligned} x(i, j) &= x(i, j+1) \oplus y(i, j) \oplus c(i-1, j+1) \\ c(i, j) &= \text{maj} \{x(i, j+1), y(i, j), c(i-1, j+1)\} \\ y(i, j) &= y(i-1, j) \end{aligned} \quad (14)$$

where the variables  $x(\cdot)$ ,  $y(\cdot)$ , and  $c(\cdot)$  correspond to the augment (i.e., sum), addend, and carry bit, respectively. Also,  $\oplus$  stands for the XOR operator and maj stands for the *majority* operator.

Equation (14) describes an infinite DG. However, an IPSP<sub>m</sub> does not accommodate an infinite number of FA's and, therefore, a finite space suffices to describe (3). Hence, (14) should be modified to describe the effects of the boundary conditions of the finite space. This modification is accomplished by the following set of lemmas.

$$\delta_k(i-1, j) = \begin{cases} \delta_k(i-1, j+1), & (p_{k,i-1} + A_{k,i-1} - j - 1, \text{ odd}) \wedge (p_{k,i-1} + A_{k,i-1} - j - 1 \geq 3) \\ \delta_k(i-1, j+1) - 1, & (p_{k,i-1} + A_{k,i-1} - j - 1, \text{ even}) \wedge (p_{k,i-1} + A_{k,i-1} - j - 1 < 3) \end{cases} \quad (15)$$

We can define as  $i$ th “column” of FA’s the finite subspace  $D_{k,i} = \{(i, j) | p_{k,i} \leq j \leq p_{k,i} + A_{k,i} - 1, p_{k,i} \in Z\}$ , where  $0 \leq i \leq n_k - 1$ . In a similar manner, we can define as  $j$ th “row” of FA’s the finite subspace  $F_{k,j} = \{(i, j) | (i, j) \in D_{k,i} \wedge 0 \leq i \leq n_k - 1\}$ . The number of rows of FA’s can be given by  $R(p_{k,i}) = \max_{0 \leq j \leq i} \{p_{k,j} + A_{k,j} - 1\} - \min_{0 \leq j \leq i} \{p_{k,j}\} + 1$ .

*Lemma 1:* The number of vertices (i.e., FA’s) in  $D_{k,i}$  that receive at least one nonlocal carry is

- 1) 0, if  $A_{k,i} \geq A_{k,i-1}$
- 2)  $A_{k,i-1} - A_{k,i}$ , if  $A_{k,i} < A_{k,i-1}$   
 $\wedge 0 \leq i \leq n - 1$
- 3)  $A_{k,i-1} - A_{k,i} - 1$ , if  $A_{k,i} < A_{k,i-1}$   
 $\wedge n - 1 < i \leq n_k - 1$ .

Lemma 1 computes the number of FA’s per column that transmit a local carry. To achieve maximum locality, the number of FA’s that transmit nonlocal carries should be minimized. The minimization is achieved by not allowing the first FA’s of any column to accept noncarry inputs unless all the nonlocal carries of the previous column have been exhausted.

*Lemma 2:* If  $A_{k,i}$  is the number of FA’s of  $i$ th column,  $i \geq 1$ , then

- a) If  $A_{k,i} \leq A_{k,i-1}$  then  $p_{k,i} = p_{k,i-1} - 1$
- b) If  $A_{k,i} > A_{k,i-1}$  then  $p_{k,i} \in \{p_{k,i} + A_{k,i-1} - A_{k,i} - 1, \dots, p_{k,i-1} - 1\}$   
 and  $R(p_{k,i})$  is minimum.

The initial value  $p_{k,0}$  of the recursion is an arbitrary integer.

Since the locality of any  $DG_k$  implies that the distance between successive FA’s that lie on the same column (have the same  $j$ ) equals one and since  $A_{k,i}$  have been computed, all that is necessary for complete construction of the  $DG_k$  is to determine for each  $i$  the smallest value of  $j$  (called here  $p_{k,i}$ ) at which an FA is placed. Recursive formulas for calculating the  $p_{k,i}$  are provided by Lemma 2.

*Lemma 3:* If  $A_{k,i} < A_{k,i-1}$  and  $p_{k,i-1} - A_{k,i-1} + 2A_{k,i} + 2 \leq j \leq p_{k,i-1} + A_{k,i-1} - 1$ , the carry link that originates from  $(i-1, j)$  ends to  $(i, j')$  where  $j' = j - \delta_k(i-1, j)$  and  $\delta_k(i-1, j)$  is as defined in (15), shown at the previous of the page. The value of  $\delta_k(i-1, p_{k,i-1} + A_{k,i-1} - 1)$  is  $p_{k,i-1} + A_{k,i-1} - p_{k,i} - A_{k,i}$ .

Lemma 3 specifies the nonlocal links required for carry propagation in the case  $A_{k,i} < A_{k,i-1}$ . In particular, the origin and the end of a nonlocal link are determined.

*Lemma 4:* The number of the different sets of URE’s, which describe  $DG_k$  equals seven.

Generally, the derived  $DG_k$  is not completely homogeneous. This implies that the graph can be “broken up” into more than one homogeneous subgraphs, each of which can be described by a set of URE’s. The seven possible sets of URE’s and the proper conditions are provided by Lemma 4.

The proofs of the above lemmas are given in [9].

### C. The Design of the Output Stage

The third stage of the IPSP $_m$  performs the mapping  $Y_{out} = \langle Y_r \rangle_m$ , which can be described as follows

$$Y_{out} = \begin{cases} Y_r, & Y_r < m \\ Y_r - m, & Y_r \geq m. \end{cases} \quad (16)$$

More specifically, in the case  $Y_r \geq m$ , (16) implies that an  $n$ -bit adder can compute  $Y_{out}$  by summing  $Y_r$  and the two’s complement of  $m$ . Whenever  $Y_r \geq m$ , it produces a carry bit which can be used to select either  $Y_r - m$  or  $Y_r$ . The implementation of the  $n$ -bit adder affects the pipelining of the whole IPSP $_m$ .

## IV. HARDWARE REALIZATION ISSUES

A comparison study of the FA-based IPSP $_m$  to a bit-sliced ROM-based architecture and a general RNS multiplier [10] in terms of the required number of transistors is given in Table I. The complexity of the ROM-based architectures has been computed in [8]. It has been assumed that a 1-b FA can be implemented in CMOS technology by 28 transistors, an AND gate with six transistors, and a 1-b multiplexer with four transistors [11]. It can be noted that the FA-based architectures require significantly reduced number of transistors than the ROM-based designs for large moduli.

A 2-D FA-based IPSP $_m$  with parallel loading of the bits  $a_c x_d$  and  $y_c$  can produce an  $n$ -b  $Y_{out}$  every  $2\Delta$  time units (the latency of one FA). Similarly, for serial loading of the  $a_c x_d$  and parallel loading of the  $y_c$ , a throughput rate of  $\ell \times 2\Delta$  can be achieved. In addition, since the throughput rate of the architecture is defined by the latency of a pipelined 1-b FA cell, it is anticipated that throughput rates up to 200 MHz can be achieved, depending on the VLSI process technology and the implementation style [12].

## V. CONCLUSIONS

The systematic derivation of several FA-based architectures for inner product step processors that operate in finite rings was presented. The usage of URE’s has been extended to bit-level algorithms. The proposed IPSP $_m$  FA-based architectures are more advantageous than the corresponding ROM-based ones in terms of the area, the latency and the throughput. Since the proposed methodology involves graph theory, recurrent equations, and mapping techniques, it forms a design environment, which can be included in a computer-aided design tool.

## REFERENCES

- [1] S. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
- [2] S. Rao and T. Kailath, “Regular iterative algorithms and their implementation on processor arrays,” *Proc. IEEE*, vol. 76, no. 3, pp. 259–269, Mar. 1988.
- [3] M. K. Birbas, D. J. Soudris, and C. E. Goutis, “Design methodology for mapping iterative algorithms on array architectures,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Singapore, June 1991, pp. 3058–3061.
- [4] J. V. McCanny, J. G. McWhirter, and S. Y. Kung, “The use of data dependence graphs in the design of bit-level systolic arrays,” *IEEE Trans. Acoust. Speech Signal Processing*, vol. 38, pp. 787–793, May 1990.
- [5] W. Shang and J. A. B. Fortes, “On time mapping of uniform dependence algorithms into lower dimensional processor arrays,” *IEEE Trans. Parallel Distributed Syst.*, vol. 3, pp. 350–363, May 1992.
- [6] W. P. Bursleson and L. L. Scharf, “A VLSI design methodology for distributed arithmetic,” *J. VLSI Signal Processing*, vol. 2, no. 4, pp. 235–252, 1991.
- [7] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, Eds., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York: IEEE Press, 1986.
- [8] T. Stouraitis, S. W. Kim, and A. Skavantzou, “Full adder-based arithmetic units for finite integer rings,” *IEEE Trans. Circuits Syst. II*, vol. 40, pp. 740–745, Nov. 1993.
- [9] D. Soudris and C. Goutis, “Mapping nested loops with if statements,” L. Svensson, Ed., BRA 3281, ESPRIT II, deliverable rep. PU/m33/C2/4, Feb. 1992.
- [10] M. Taheri, G. Jullien, and W. Miller, “High-speed signal processing using systolic arrays over finite rings,” *IEEE J. Selected Areas Commun.*, pp. 504–512, Apr. 1988.
- [11] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Reading, MA: Addison-Wesley, 1993.
- [12] F. Lu and H. Samuelli, “A 200-MHz CMOS pipelined multiplier-accumulator using a quasidomino dynamic full-adder cell design,” *IEEE J. Solid-State Circuits*, vol. 28, pp. 123–132, Feb. 1993.