# Design of a High-Speed Matrix Multiplier Based on Balanced Word-Width Decomposition and Karatsuba Multiplication

**R. L. Bhargavi, M.Merlin Moses, V.Karthikeyan and C.Karthikeyan**

***Abstract*— This paper presents a flexible 2x2 matrix multiplier architecture. The architecture is based on word-width decomposition for flexible but high-speed operation. The elements in the matrices are successively decomposed so that a set of small multipliers and simple adders are used to generate partial results, which are combined to generate the final results. Balanced word-width decomposition scheme is discussed, which support 2's complement inputs, and its overall functionality is verified and designed with a field-programmable gate array (FPGA). The architecture can be easily extended to a reconfigurable matrix multiplier. The objective is to propose a flexible and energy efficient matrix multiplier, which can be extended to reconfigurable high speed processing implementation, using word width decomposition technique. This technique is based on divide and conquers approach. The Karatsuba multiplication is proposed in this basic approach. This Karatsuba multiplication is an efficient procedure for multiplying large numbers, which gives high speed performance than the booth multiplier.***

***Keywords*— Balanced word-width decomposition. Field-programmable gate array (FPGA) implementation, matrix multiplier, Reconfigurable architecture.***

## I. INTRODUCTION

In many multimedia applications, fast and efficient matrix multipliers are very critical in overall operations. Matrix multiplication is an often used core operation in a wide variety of graphic, image, robotics, and signal processing applications. Traditional approaches for matrix multiplications are carried out either by software on fast processors or by hardware with multiple scalar multipliers. Software operations of the matrix multiplications can be very slow and become bottlenecks in overall system operations. On the other hand, hardware multipliers incorporate high-speed logic, but may be very costly in terms of complexity and power consumption. Moreover, these multipliers are usually designed for a specific word length and often lack flexibility where the size of scalar multipliers must correspond to the input of the matrix multiplier. The objective of this paper is to propose a flexible and energy efficient matrix multiplier, which can be extended to reconfigurable high speed processing implementation. A key concept is to apply the divide-and-conquer approach by decomposition, both at matrix level and at word level.

**Manuscript Received on December 2012.**

**R.L.Bhargavi,** Lecturer's in Einstein College of Engineering, Tirunelveli (Tamil Nadu), India.

**M.Merlin Moses,** Lecturer's in Einstein College of Engineering, Tirunelveli (Tamil Nadu), India.

**V.Karthikeyan,** Lecturer's in Einstein College of Engineering, Tirunelveli (Tamil Nadu), India.

**C.Karthikeyan**, Assistant Professor in Einstein College of Engineering, (Tamil Nadu), India.

Many studies have been done on matrix multiplications. Most notably, [1] introduced an efficient matrix multiplication algorithm for linear arrays with reconfigurable pipelined bus systems. This architecture also follows the divide-and-conquer approach for high-speed parallel processing. However, their decomposition is limited to the matrix level. A matrix multiplier for sparse matrices that reduces the input/output (I/O) and the number of trivial multiplications is introduced in [2]. This architecture is focused on the data elements of the matrices. However, the architecture is highly beneficial only when the matrices are sparse. In this paper, the proposed architecture also reduces computation energy by exploiting zeros during actual computations. Most of the previous implementations of matrix multiplication on field-programmable gate arrays (FPGAs) are focused mainly on tradeoff between area and maximum running frequency [3]–[7]. A bit-serial matrix multiplier using Booth encoding was implemented on the FPGA [3]. The multiplier discussed in [4] improved the design in [3] using modified Booth-encoder multiplication along with Wallace tree addition, where the running frequency has been doubled from the design described in [3]. These designs are further improved in terms of area and speed in [7]. All of their designs consider the input word-width directly and the size of scalar multipliers can be significantly large for practical VLSI implementation when the input word-width increases. Moreover, their designs are mainly focused on reducing FPGA resources by incorporating static memory in the implementation. This design is focused on both computational and energy efficiency, as well as structural flexibility.

In this paper, high-speed matrix multiplications based on two-level matrix decomposition is considered. The idea of decomposition has been applied in many arithmetic computations to reduce the amount of computation and latency [8] [10]. The idea of decomposition is further extended into the word level. Initially, a matrix multiplication with higher dimension, is decomposed into several 2x2 matrix multiplications. Then, these decomposed 2x2 matrix multiplications are further decomposed into several matrix multiplications where each element in these matrix multiplications is represented with smaller word-width. Then a set of simple, but fast operators is used concurrently to generate partial results. The structure and size of the simple operators are fixed irrespective of the input element word-width. The final outputs are generated by combining (composing) the partial results through accumulation with adder trees. The proposed matrix multiplications support 2's complement input with energy efficient computation. The design methodology of the proposed architecture can utilize varying numbers of simple operators to support various

degrees of time multiplexing to reduce complexity with a low-latency penalty. This presentation is focused on an architecture for a 2x2 matrix multiplier based on word-width decomposition, which can be a building block for computing larger dimension matrix multiplications.

In this paper, its performance and complexity with design on commercially available FPGA is demonstrated. The remainder of this paper has four sections. In Section II, it describes the concept of matrix multiplications via two-level decomposition. Balanced word-width decomposition scheme is discussed, as well as an issue on 2's complement data representation in the proposed architecture. In Section III, a set of key units comprising a flexible matrix multiplier architecture is described in detail. A key set of design tradeoff parameters is also discussed. In Section IV, Proposed multiplication is presented. Then Simulation results are summarized in Section V.

## II. MATRIX MULTIPLICATION DESIGN OVERVIEW

### A. *Theory of Operation*

The overall operation is based on two basic assumptions. 1) Matrix multiplication is performed on NxN matrices where N is a power of two. 2) Each element in the matrices is a fixed point integer with word-width of W. In the design, it adopts two-level decomposition. Initially, NxN matrix multiplication is decomposed into several 2x2 matrix multiplications. This process is illustrated with N=4 as

$$AB = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} \quad (1)$$

$$= \begin{bmatrix} A_1B_1 + A_2B_3 & A_1B_2 + A_2B_4 \\ A_3B_1 + A_4B_3 & A_3B_2 + A_4B_4 \end{bmatrix} \quad (2)$$

where $A_1, \ldots, A_4, B_1, \ldots, B_4$ are all 2x2 matrices. After the initial decomposition, all matrix multiplications are 2x2 matrix multiplications where word-width of each element is W. Then, each matrix multiplication (i.e., eight such matrix multiplications for the example above) is decomposed in terms of word-width. For example, for the value of p equal to4

$$A_1B_1 = \begin{bmatrix} 327 & 20 \\ 71 & 236 \end{bmatrix} \begin{bmatrix} 127 & 430 \\ 86 & 11 \end{bmatrix} \quad (3)$$

$$A_1 = Q_{A_1}2^p + R_{A_1}$$

$$= \begin{bmatrix} 20 & 1 \\ 4 & 14 \end{bmatrix} 2^p + \begin{bmatrix} 7 & 4 \\ 7 & 12 \end{bmatrix} \quad (4)$$

$$B_1 = Q_{B_1}2^p + R_{B_1}$$

$$= \begin{bmatrix} 7 & 26 \\ 5 & 0 \end{bmatrix} 2^p + \begin{bmatrix} 15 & 14 \\ 6 & 11 \end{bmatrix}. \quad (5)$$

Thus, 2x2 matrix multiplication is represented as

$$A_1B_1 = (Q_{A_1}Q_{B_1}2^p + Q_{A_1}R_{B_1} + R_{A_1}Q_{B_1})2^p + R_{A_1}R_{B_1} \quad (6)$$

where

$Q_{A_1}Q_{B_1}, Q_{A_1}R_{B_1}, R_{A_1}Q_{B_1}$, and $R_{A_1}R_{B_1}$

represent decomposed 2x2 matrix multiplications. Two matrices, QA1 and QB1, are decomposed further until each element in all the decomposed matrices is less than 2^p where all elements are represented with p-bit precision. Upon completion of the decomposition process, all matrix multiplications with p-bit precision are computed in parallel with much smaller booth multipliers than W. The outputs from this computation are accumulated to generate the final outputs for one 2x2 W-bit matrix multiplication. After repeating the same process, the outputs for NxN matrix multiplications are constructed. Throughout the paper, the presentation is focused on word-width decomposition.

### B. *Word Width Decomposition*

There are two ways to decompose the matrices. The first approach is to divide the width of the original elements successively in half.

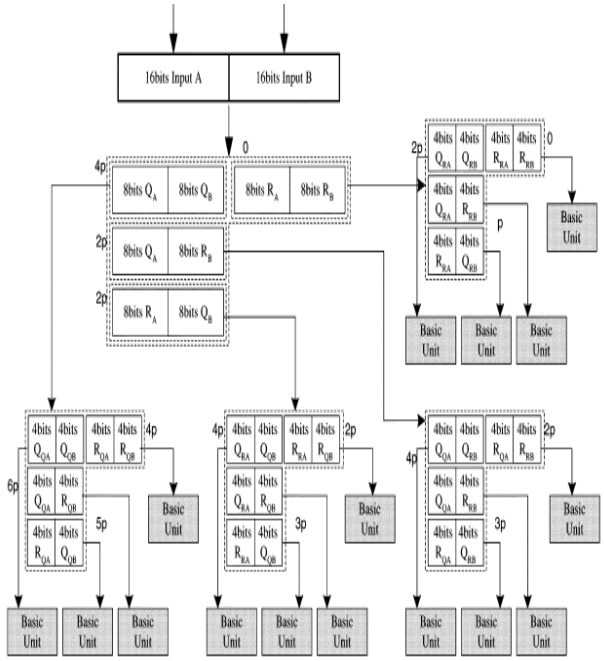


Fig. 1. Illustration of the balanced word-width decomposition with W = 16 and p = 4.

This approach, called balanced word-width decomposition, is illustrated in Fig. 1. Let a finite number represent the decomposed data width. Then, a multiplication by 2^p becomes a simple p-bit shift. Initially, the matrix multiplication with W–bit input elements is decomposed into two sub matrix multiplications with W/2 bit elements. Then, the decomposed matrix multiplication is further decomposed until the element word length is equal to p. After the decomposition, there will be many but smaller sub matrix multiplications which can be performed with simple arithmetic units. The depth of the decomposition tree depends on the word length of the original data element. The restriction with this approach is that the size of W must be p*2^i , where i is an integer. For example, for p=4, the only possible word lengths are W=4,8,16,32…….

The illustrations shown in Figs.1 may seem to suggest that the decomposition process takes multiple stages of operations. Actually, the decomposition of the original matrix multiplication results in 16 sub matrix multiplications (i.e., for W=16). It will show that the actual decomposition can be done directly from the input elements and the decomposition processes illustrated above, are handled during the composition where shiftings and summations are performed. Basically, the decomposition process is merely dividing the original values through interconnection distribution.

### *C. Composition*

The composition is the reverse operation of the decomposition. Consider two matrices A and B W=16 with input word-width which are multiplied as

$$AB = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \quad (7)$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix} \quad (8)$$

$$= \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = C. \quad (9)$$

Consider the balanced decomposition with W=16 and p=4. Then after the decomposition, the matrix multiplication AB is represented as

AB=
(QQAQQB)2^4p+(QQARQB)2^3p+(RQAQQB)2^3p+(RQARQB)2^2p+(QRAQQB)2^3p+(QRARQB)2^2p)+(RRAQQB)2^2p+(RRARQB)2^p+(QQAQRB)2^3p)+
(QQARRB)2^2p)+(RQAQQB)2^2p)+(RQARRB)2^P+(QRAQRB)2^2p+(QRARRB)2^p)+(RRAQRB)2^p+
(RRARRB)

Note that with the exception of multiplication factors, the original matrix multiplication AB consists of many smaller booth matrix multiplications, which can be computed with the same hardware. The results from these smaller matrix multiplications are the partial results for cij where they are accumulated by an adder tree to generate the outputs of the matrix multiplication. Hence, the adder tree is executed four times to generate a complete 2x2 output matrix C.

Figs.2 illustrates composition schemes for the balanced decomposition schemes, respectively. Note that each adder in the adder tree consists of two inputs where one of the inputs is scaled by 2^p, which is equivalent to a p-bit shift operation. Thus, in the adder tree, the output of each adder is appropriately shifted to reverse the decomposition operations.

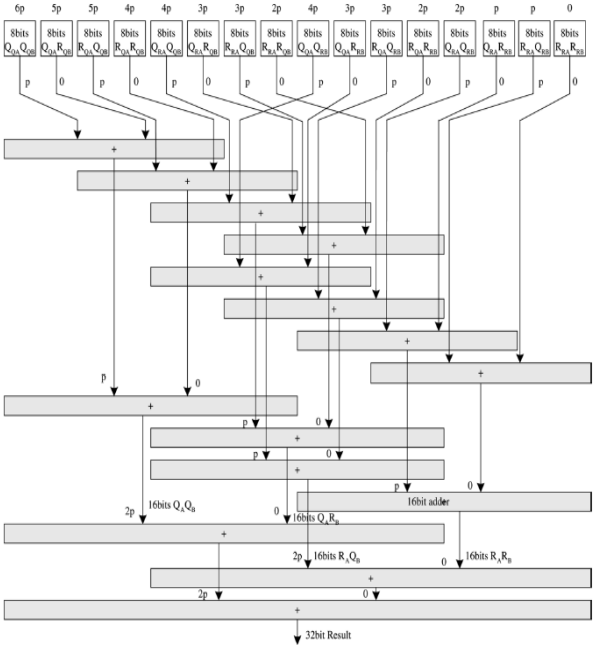


Fig. 2. Illustration of 2-input adder tree structure in the balanced composition process with W = 16 and p = 4.

## III. DESIGN APPROACH

### *A. Overall Architecture*

The proposed overall architecture is shown in Fig. 3. The architecture consists of decomposition, basic operation, and composition unit. In this architecture, a small number of p-bit multipliers is used several times to generate W-bit results. Both decomposition and composition units are operated at the same clock frequency, which is equal to the I/O rate.

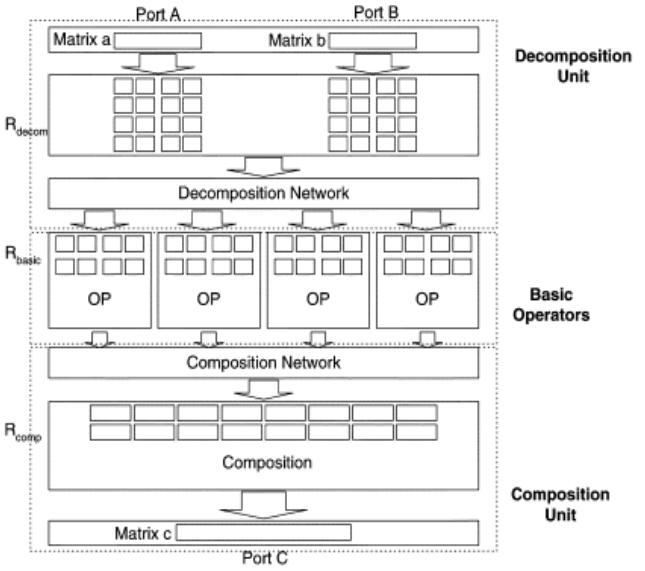


Fig. 3. Overall architecture with three main units. Figure is illustrated with the number of basic operators, N , which is equal to 4.

### *B. Decomposition Unit*

In the decomposition unit, a direct decomposition of W-bit elements is accomplished. The inputs enter the decomposition unit in a pair of elements of matrix A and B as (a11,b11),(a12,b12),(a21,b21), and (a22,b22). The decomposition unit is responsible for two main tasks. The first task is to segment W-bit elements into several p-bit elements and make conversion to support 2's complement number representation.

The second task is basically ordering the data so that output elements are generated at the same rate with the input rate with low overall latency. Once the data are properly ordered by the decomposition unit, all basic operators can start the processing as early as possible without any stoppage. The decomposition unit is designed to take a continuous stream of 2x2 matrices.

### *C.Radix- Basic Operator*

In a basic operator, each operation takes four inputs. The outputs from the booth multipliers are added for the final output. (p+2)-bit elements (one bit for sign extension and the other bit for zero flag) of the working submatrix are loaded in order into registers. Once ordered, the data will propagate through a chain of registers. This ordering is established to produce four elements corresponding to c11,c12,c21, and c22. For example, for word-width W of 16 and p of 4, there will be total of 16 concurrent submatrix multiplications. With four basic operators executing in parallel, each output is generated in 4 iterations.

### *D. Composition Unit: 0-mp Adder Tree*

Figs.4 illustrates the composition adder tree. The composition unit consists of registers and an adder tree with 0-mp adders. (i.e.,zero bits at the low significant bits of one input and extended sign bits at the high significant bits of the other).

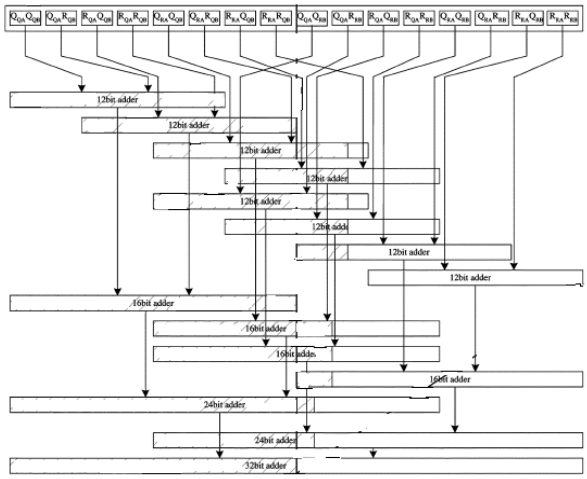


Fig. 4. Illustration of the ripple carry adder tree by merging 0-mp adders for the balanced composition. W = 16 and p = 4 for the illustration.

Hence, the worst case delay is less than a normal adder with the same width. In the implementation, the adders are constructed with carry-select adders where each segment of adder is designed with a p–bit ripple carry adder [12].

For 4-bit ripple carry adder we use 12-bit adder,16-bit adder,24-bit adder and 32-bit adder in the composition unit. At the end of adder tree, the output word-width is 2W-bits.

## IV. KARATSUBA MULTIPLICATION

The Karatsuba multiplication is an efficient procedure for multiplying large numbers which gives high speed performance than the booth multiplier. The Karatsuba algorithm is a notable example of divide and conquers method, specifically for binary splitting.

The basic step of Karatsuba algorithm is a formula that allows us to compute the product of two large numbers $x$ and $y$ using three multiplications of smaller numbers, each with about half as many digits as $x$ or $y$, plus some additions and digit shifts. Karatsuba observed that we can compute $xy$ in only three multiplications, at the cost of a few extra additions:

Let $z_2 = x_1y_1$

Let $z_0 = x_0y_0$

Let $z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0$

since

$$z_1 = (x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0) - x_1y_1 - x_0y_0 = x_1y_0 + x_0y_1$$

## V. SIMULATION RESULTS

Simulation is performed by using Xilinx ISE 9.1 development tool for the design. The speed of the designs is verified through synthesis-Report from post-synthesis simulation and all the results depend on given input. The simulation results are shown for booth multiplier and karatsuba multiplier. The simulation results are shown in figures 5(a),5(b),6(a).

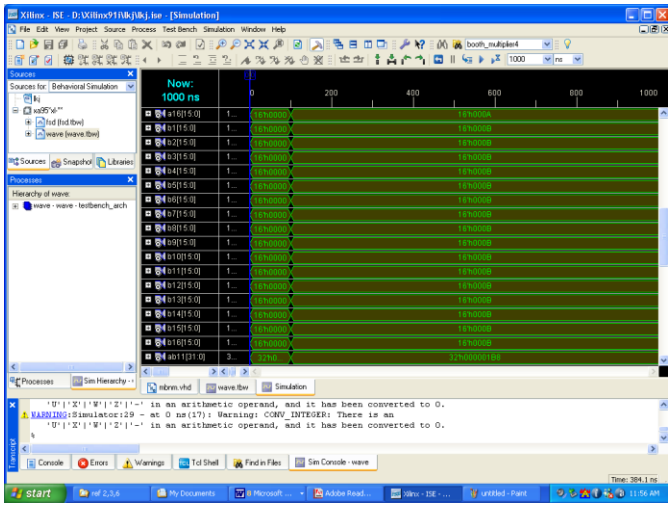

Fig.5(a).simulation output for matrix multiplications

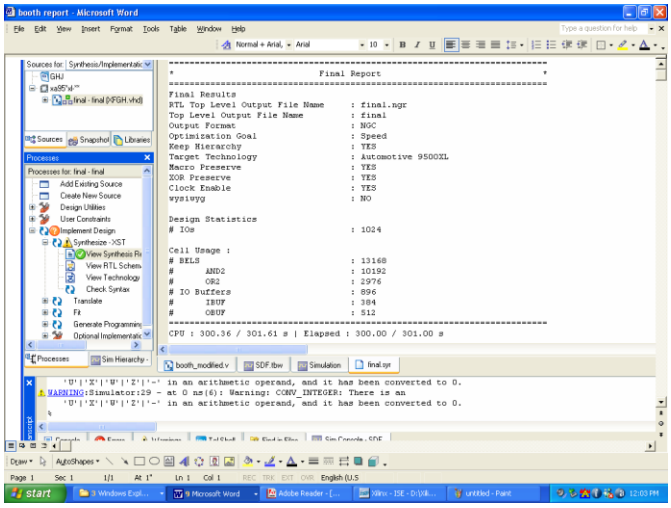

Fig.5(b).synthesis Report for booth multiplier

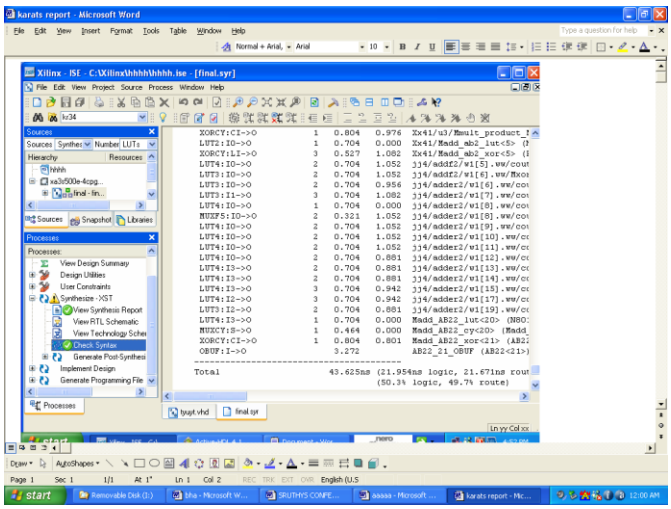

Fig.6(a).synthesis Report for karatsuba multiplier

## V. CONCLUSION

In this paper, a design methodology is proposed for a flexible 2x2 matrix multiplier architecture. In this particular architecture, a set of small but fast multipliers are used. The balanced word width decomposition is used for flexibility with high-speed operation. In this decomposition technique, the booth multiplier is used for multiplication unit. But this

booth multiplier results high latency. This can be overcome by the Karatsuba multiplication, which is an efficient procedure for multiplying large numbers, for giving high speed performance. Also by this methodology larger matrix multiplications can be computed.

## REFERENCES

[1] K.Li,Y.Pan,and S.Q.Zheng,"Fast and processor efficient parallel matrix multiplication algorithms on a linear array with a reconfigurable pipelined bus system,"IEEE Trans.Parallel Distrib.Syst.,vol.9,no. 8,pp.705–720,Aug.1998.

[2] C.I.Brown and R.B.Yates,"VLSI architecture for sparse matrix mul-tiplication," Electron.Lett.,vol.32,no.10,pp.891–893,May 1996.

[3] O.Mencer,M.Morf,and M.Flynn,"PAM-Blox: High performance FPGA design for adaptive computing,"in Proc.IEEE Symp.FPGAs Custom Computing Machines,1998,pp.167–174.

[4] A.Amira,A.Bouridane,and P.Milligan,"Accelerating matrix product on reconfigurable hardware for signal processing,"in Proc.11th Int. Conf.Field-Programmable Logic Appl.(FPL),2001,pp.101–111.

[5] J.Jang,S.Choi, and V.K.Prasanna, "Energy-efficient matrixmulti-plication on FPGAs,"in Proc.Int.Conf.Field Programmable Logic Appl.,2002,pp.534–544.

[6] V.K.Prasanna and Y.Tsai,"On synthesizing optimal family of linear systolic arrays for matrix multiplication,"IEEE Trans.Comput.,vol. 40,no.6,pp.770–774,Jun.1991.

[7] J.-W.Jang,S.Choi,and V.K.Prasanna,"Area and time efficient im-plementations of matrix multiplication on FPGAs,"in Proc.IEEE Int. Conf.Field Programmable Technol.,2002,pp.93–100.

[8] R.Lin,"Bit-matrix decomposition and dynamic reconfiguration:Uni-fied arithmetic processor architecture,design,and test,"in Proc.Re-configurable Arch.Workshop (RAW),2002,p.83.

[9] R.Lin,"Bit-matrix decomposition and dynamic reconfiguration: Uni-fied arithmetic processor architecture,design,and test,"in Proc.Re-configurableArch.Workshop (RAW),2002,p.83.

[10] R.Lin,"Reconfigurable parallel inner product processor architectures," IEEE Trans. Very Large Scale Integr.(VLSI)Syst.,vol.9,no.2,pp. 261–272,Apr.2001.

[11] S.Choi,R.Scrofano, V.K.Prasanna, and J.-W.Jang,"Energy-effi-cient signal processing using FPGAs,"in Proc.ACM/SIGDA Int.Symp. Field-Programmable Gate Arrays, 2003, pp.225–234.

[12] J.M.Rabaey, A.Chandrakasan and B.Nikolic ´,Digital Integrated Cir-cuits: A Design Persepective, 2nd ed.Englewood Cliffs,NJ: Pren-tice- Hall,2003.

[13] C.R.Baugh and B.A.Wooley,"A t wo's complement parallel array multiplication algorithm, "IEEE Trans.Comput., vol.C-22,no. 1–2, pp.1045–1047,Jan.193.