Design of Linear Systolic Arrays for Matrix Multiplication

Emina I. MILOVANOVIĆ, Mile K. STOJČEV, Igor Ž. MILOVANOVIĆ, Tatjana R. NIKOLIĆ
Faculty of Electronic Engineering, A. Medvedeva 14, 18000 Nis, Serbia
ema@elfak.ni.ac.rs

***Abstract*—This paper presents architecture for matrix multiplication optimized to be integrated as an accelerator unit to a host computer. Two linear systolic arrays with unidirectional data flow (ULSA), used as hardware accelerators, where synthesized in this paper. The solution proposed here is designed to accelerate both the computation and communication by employing hardware address generator units (AGUs). The proposed design has been implemented on Xilinx Spartan-2E and Virtex4 FPGAs. In order to evaluate performance of the proposed solution, we have introduced quantitative and qualitative performance criteria. For the ULSA with n processing elements (PEs), the speed-up is O(n/2). Average gain factor of hardware AGUs is about 2.7, with hardware overhead of 0.6% for 32-bit PEs.**

***Index Terms*—address generator units, linear systolic arrays, matrix multiplication.**

I. INTRODUCTION

Matrix multiplication is often used as a kernel operation in digital signal processing applications. It requires $O(n^3)$ multiply with accumulation (MAC) operations on a sequential computer. Significant speed-up in computation time can be achieved by assigning complex computation intensive tasks to hardware and by exploiting available parallelism in algorithms. This paper presents an efficient architecture for matrix multiplication optimized to be integrated as an accelerator unit to a host [1-7].

A number of dedicated circuits for matrix operations have been presented in [1-5]. However, the I/O bandwidth required by those architectures, with aim to achieve high processing rate, is not taken into account. In opposite with solutions presented in [1-5], that accelerate just matrix multiplication computation, the proposal presented in this paper has been primarily designed to speed up both computation and communication time. In order to achieve high level of independency of accelerator operation, we decided to implement accelerator as a stand alone processing unit.

This paper carries readers through the design and implementation of accelerator unit intended for matrix multiplication using FPGA platform. We next continue this paper with mathematical background for designing systolic algorithms for unidirectional systolic arrays (ULSA). Then we concentrate on the synthesis of two systolic arrays of ULSA type. We continue with global system structure. After that, we dive into a detailed system structure which includes the role of hardware accelerators in address calculations and accelerator operation. Speedup, efficiency, hardware overhead and gain factor, as quantitative and qualitative performance metrics of the proposed design are considered. FPGA computer boards based on Xilinx Spartan2E and Virtex4 series were used to verify both ULSA operation and address generation. Concluding remarks are given in Section VII.

II. MATHEMATICAL BACKGROUND AND SYSTOLIC ALGORITHMS

Let $A=(a_{ik})$ and $B=(b_{ki})$ be rectangular matrices of order $N_1 \times N_3$ and $N_3 \times N_2$, respectively. Denote with $\vec{A}_k$ and $\vec{B}_{k\cdot}$ column- and row-vectors, $k=1,2,\ldots,N_3$, of matrices A and B. Their product, $C = A \cdot B$ can be calculated using outer vector product method, defined as

$$\begin{aligned} & C^{(0)} := 0 \\ & C^{(k)} := C^{(k-1)} + \vec{A}_k \vec{B}_{k\cdot}, \quad k = 1,2,...,N_3 \\ & C := C^{(N_3)} \end{aligned} \tag{1}$$

In order to synthesize 1D systolic array (SA) that implements the computation defined by (1), it is enough to consider one iteration step only. Without affecting the generality, we will consider the case $k=1$, i.e. the computation of $C^{(1)} := \vec{A}_1 \cdot \vec{B}_{1\cdot}$. The final result is obtained after N_3 iteration steps.

The basic systolic algorithm for computing $C^{(1)} = (c_{ij}^{(1)})$ has the following form (see [6-7])

Algorithm_1

```
for $j := 1$ to $N_2$ do
for $i := 1$ to $N_1$ do
{
    $b(i,j,1) := b(i-1,j,1)$;
    $a(i,j,1) := a(i,j-1,1)$;
    $c(i,j,1) := c(i,j,0) + a(i,j,1) * b(i,j,1)$;
}
```

where $b(0,j,1) \equiv b_{1j}$, $a(i,0,1) \equiv a_{i1}$, $c(i,j,0) \equiv c_{ij}^{(0)} = 0$, $c(i,j,1) \equiv c_{ij}^{(1)}$, for $i = 1,2,\ldots,N_1$, and $j = 1,2,\ldots,N_2$.

Algorithm_1 can be accompanied with directed acyclic graph located in 3D Euclidian space. By projecting this graph along direction $\vec{\mu} = [1 \ \ -1 \ \ 0]^T$ on the plane orthogonal to this direction, a directed graph in 2D space is obtained. This graph directly corresponds to the ULSA convenient for implementation of Algorithm_1 (see for example [8-10]). However, the number of processing elements (PEs) in the obtained array is not optimal. Our goal

---

This work was supported in part by the by the Serbian Ministry of education and Science, project No TR 32009 and TR 32012.

Digital Object Identifier 10.4316/AECE.2014.01006

is to design an array which consists of optimal number of PEs, i.e. $\Omega = \min\{N_1, N_2\}$, and minimal possible execution time for that number of PEs. Therefore, instead of Algorithm_1 we consider the following two systolic algorithms

Algorithm_2

```
  for $j := 1$ to $N_2$ do
 for $i := 1$ to $N_1$ do
{
$b(i-\frac{1}{2}, 1-i+j+N_1, 1) := b(i-1, 1-i+j+N_1, 1)$
$b(i, 1-i+j+N_1, 1) := b(i-\frac{1}{2}, 1-i+j+N_1, 1)$
$a(i, 1-i+j+N_1, 1) := a(i, -i+j+N_1, 1)$ ;
$c(i, 1-i+j+N_1, 1) := c(i, 1-i+j+N_1, 0) +$
$+ a(i, 1-i+j+N_1, 1) * b(i, 1-i+j+N_1, 1)$
}
```

where $b(i, t+N_2, 1) \equiv b(i, t, 1)$, $c(i, t+N_2, 0) \equiv c(i, t, 0)$, for $i = 1, 2, \ldots, N_1$ and $t = 1, 2, \ldots, N_2$

Algorithm_3

```
for $j := 1$ to $N_2$ do
for $i := 1$ to $N_1$ do
{
$a(1+i-j+N_2, j-1/2, 1) := a(1+i-j+N_2, j-1, 1)$
$a(1+i-j+N_2, j, 1) := a(1+i-j+N_2, j-1/2, 1)$
$b(1+i-j+N_2, j, 1) := b(i-j+N_2, j, 1)$ ;
$c(1+i-j+N_2, j, 1) := c(1+i-j+N_2, j, 0) +$
$+ a(1+i-j+N_2, j, 1) * b(1+i-j+N_2, j, 1)$;
}
```

where $a(t+N_1, 0, 1) \equiv a(t, 0, 1)$, $c(t+N_1, j, 0) \equiv c(t, j, 0)$, for each $j = 1, 2, \ldots, N_2$ and $t = 1, 2, \ldots, N_1$. Let us note that Algorithms_2 and _3 are input/output equivalent to Algorithm_1. They are tailored to the directions $\vec{\mu} = [1 \ \ -1 \ \ 0]^T$ and $\vec{\mu} = [-1 \ \ 1 \ \ 0]^T$, on index variables *i* and *j*, respectively (see [11-13]).

III. ULSA SYNTHESIS

Denote with SA1 and SA2 the ULSA arrays synthesized according to Algorithms_2 and _3 and directions $\vec{\mu} = [1 \ -1 \ 0]^T$ and $\vec{\mu} = [-1 \ 1 \ 0]^T$, respectively.

Directed graphs that correspond to Algorithms_2 and _3 are $G_1 = (P_{\text{int}}^{(1)} \cup P_d^{(1)}, E_1)$ and $G_2 = (P_{\text{int}}^{(2)} \cup P_d^{(2)}, E_2)$. Vertices of these graphs are, respectively, defined by index sets

$$P_{\text{int}}^{(1)} = \{\vec{p} = [i \ \ 1-i+j+N_1 \ \ 1]^T\},$$
$$P_d^{(1)} = \{\vec{p}_d = \begin{bmatrix} i-\frac{1}{2} & 1-i+j+N_1 & 1 \end{bmatrix}^T\} \tag{2}$$

and

$$P_{\text{int}}^{(2)} = \{\vec{p} = [1+i-j+N_2 \ \ j \ 1]^T\},$$
$$P_d^{(2)} = \{\vec{p}_d = \begin{bmatrix} i+i-j+N_2 & j-\frac{1}{2} & 1 \end{bmatrix}^T\} \tag{3}$$

Edges in graphs G_1 and G_2 are determined by the column vectors of matrices E_1, i.e. E_2

$$E_1 = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, E_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

At each vertex defined by $P_{\text{int}}^{(1)}$ (i.e. $P_{\text{int}}^{(2)}$), a MAC operation is performed. In vertices defined by $P_d^{(1)}$ (i.e. $P_d^{(2)}$) data are delayed for one clock cycle, i.e. no computation is performed.

One of possible transformation matrices, S, that can be joined to the direction $\vec{\mu} = [1 \ -1 \ 0]^T$ (i.e. $\vec{\mu} = [-1 \ 1 \ 0]^T$) has the following form

$$S = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

This matrix maps 3D graph G_1 (i.e. G_2) into 2D graph which corresponds to the array SA1 (i.e. SA2).

Locations of PEs and delay elements in the x-y plane in the array SA1 are obtained from the following equations

$$PE \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = S \cdot \vec{p} == \begin{bmatrix} 1+j+N_1 \\ 1 \end{bmatrix}, \quad \vec{p} \in P_{\text{int}}^{(1)}, 1 \le j \le N_2 \tag{6}$$

and

$$P_d \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = S \cdot \vec{P}_d = \begin{bmatrix} j+N_1+\frac{1}{2} \\ 1 \end{bmatrix}, \vec{p}_d \in P_d^{(1)}, 1 \le j \le N_2. \tag{7}$$

Communication links in the SA1 are defined by the column vectors of matrix Δ

$$\Delta = \begin{bmatrix} \vec{e}_b^{\,2} & \vec{e}_a^{\,2} & \vec{e}_c^{\,2} \end{bmatrix} = S \cdot E_1 = \begin{bmatrix} \frac{1}{2} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{8}$$

Vectors $\vec{e}_b^{\,2}, \vec{e}_a^{\,2}$ and $\vec{e}_c^{\,2}$ define propagation directions for elements of matrices B, A and C, as well.

Similarly, the array SA2 which is obtained according to Algorithm_3, can be synthesized from the following equations

$$\left.\begin{aligned} PE &\rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} i+N_2+1 \\ 1 \end{bmatrix}, \quad 1 \le i \le N_1, \\ P_d &\rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} i+N_2+\frac{1}{2} \\ 1 \end{bmatrix}, \quad 1 \le i \le N_1, \\ \Delta &= \begin{bmatrix} \vec{e}_b^{\,2} & \vec{e}_a^{\,2} & \vec{e}_c^{\,2} \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}\right\} \tag{9}$$

According to (6) and (9) it can be concluded that the arrays SA1 and SA2 have $\Omega = N_2$ and $\Omega = N_1$ PEs, respectively. This implies that when $N_1 > N_2$ the array SA1 is optimal design choice. Otherwise, the array SA2 is optimal.

Retiming the initial data schedule

Initial schedule of elements of column vector $\vec{A}_1$, row-vector $\vec{B}_{1\cdot}$, and matrix $C^{(0)}$ for Algorithm_2 is determined by the set

$$P_{in} = P_{in}(a) \cup P_{in}(b) \cup P_{in}(c) \tag{10}$$

where

$$\left.\begin{aligned} P_{in}(a) &= \{\vec{p}_a = [i\,0\,1]^T\}, \\ P_{in}(b) &= \{\vec{p}_b = [0\,1-i+j+N_1\;1]^T\}, \\ P_{in}(c) &= \{\vec{p}_c = [i\,1-i+j+N_1\;0]^T\} \end{aligned}\right\} \tag{11}$$

for $i = 1,2,\ldots,N_1$ and $j = 1,2,\ldots,N_2$. This schedule does not provide correct execution of Algorithm_2 on SA1. Namely, in order to provide correct timing, it is necessary to involve skewing of input data in time domain (see for example [14]). This process can be described by the timing function

$$t(\vec{p}) = \vec{\Pi}\cdot\vec{p} + \alpha,\; \vec{\Pi} = [2\;\;1\;\;1] \tag{12}$$

where $\vec{p} \in P_{int}^{(1)}$, and α is a constant determined so that $t(\vec{p}) \geq 0$ for all $\vec{p} \in P_{int}^{(1)}$. This function defines time instances of activities in each node of graph G_1. In our case it is

$$\begin{aligned} &t(\vec{p}) = 2u + v + w - N_1 - 4, \\ &u = i,\; v = 1-i+j+N_1,\; w = 1 \end{aligned} \tag{13}$$

Reordering of space P_{in} into P_{in}^*, which provides correct execution of Algorithm_4, can be described by the following equation

$$\vec{p}_\gamma^{\,*} = \vec{p}_\gamma - (t(p_\gamma)+1)\vec{e}_\gamma^{\,3}, \qquad \gamma \in \{a,b,c\} \tag{14}$$

Reordered space $P_{in}^* = P_{in}^*(a) \cup P_{in}^*(b) \cup P_{in}^*(c)$ is defined by

$$\left.\begin{aligned} P_{in}^*(a) &= \{\vec{p}_a = \begin{bmatrix} i & -2i+N_1+2 & 1\end{bmatrix}^T\} \\ P_{in}^*(b) &= \{\vec{p}_b = \begin{bmatrix} \dfrac{i-j+1}{2} & 1-i+j+N_1 & 1\end{bmatrix}^T\} \\ P_{in}^*(c) &= \{\vec{p}_c = [i\,1-i+j+N_1\;2-i-j]^T\}. \end{aligned}\right\}$$

Finally, by mapping P_{in}^* using transformation S, we obtain data schedule for elements of vectors $\vec{A}_1$, $\vec{B}_{1.}$ and matrix $C^{(0)} = (c_{ij}^{(0)})$ at the beginning of the computation of Algorithm_2 on SA1. It is defined as

$$\left.\begin{aligned} a(i,0,1) &\rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -i+N_1+2 \\ 1 \end{bmatrix} \\ b(0,1-i+j+N_1,1) &\rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \dfrac{-i+j+2N_1+3}{2} \\ 1 \end{bmatrix} \\ c(i,1-i+j+N_1,0) &\rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} j+N_1+1 \\ 2-i-j \end{bmatrix}, \end{aligned}\right\} \tag{16}$$

for $i = 1,2,\ldots,N_1$ and $j = 1,2,\ldots,N_2$

Figures 1 and 2 illustrate the arrays SA1 and SA2 for the case $N_1 = N_2 = 3$.

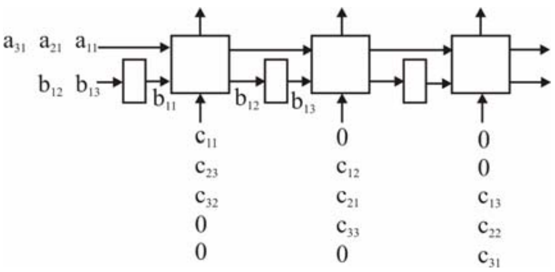

a_{31} a_{21} a_{11}
b_{12} b_{13}
b_{11}
b_{12}
b_{13}
c_{11} c_{23} c_{32} 0 0
0 c_{12} c_{21} c_{33} 0
0 0 c_{13} c_{22} c_{31}

Figure 1. Data schedule in the array SA1 at the beginning of the computation, for $N_1 = N_2 = 3$.

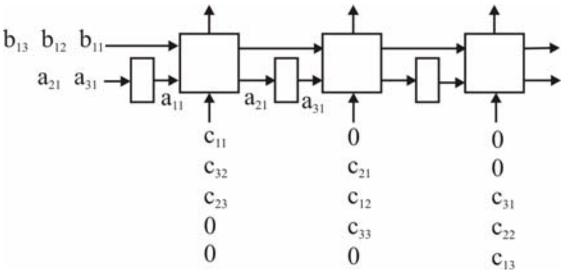

b_{13} b_{12} b_{11}
a_{21} a_{31}
a_{11}
a_{21}
a_{31}
c_{11} c_{32} c_{23} 0 0
0 c_{21} c_{12} c_{33} 0
0 0 c_{31} c_{22} c_{13}

Figure 2. Data schedule in the array SA2 at the beginning of the computation, for $N_1 = N_2 = 3$.

IV. STRUCTURE OF THE SYSTEM AND ROLE OF HARDWARE ACCELERATORS

Numerous scientific and embedded applications, such as multimedia, image and speech processing, are characterized by complex array index manipulation and great number of data accesses. Those applications require specific address calculations that general purpose processors can not fulfill at a reasonable time [8], [15-16]. To speedup address generation, special hardware building blocks, called address generation units, are designed [19].

Address sequences, used to access data in memory, are generated according to the address equations (AEs). AE is a function obtained from the software description (15) of the algorithm [17]. It is defined as:

$$AE = f(I_1, I_2, \ldots, I_n, r_1, r_2, \ldots, r_m), \tag{17}$$

where parameters are nested loop indices (I_i), $i = 1,2,\ldots,n$, or range addresses (r_j), $j = 1,2,\ldots,r_m$.

Our design solution is presented in Fig. 3. It is based on a shared system bus as an interconnection network. We have implemented accelerator as a stand alone processing unit in order to achieve high level of independency in accelerator operation. As shown in Fig. 3, the accelerator consists of three functional units: linear systolic array (ULSA), distributed memory (DM), and address generator unit (AGU). The AGU performs three functions. First, during the initialization, it transforms host address space into accelerator address space. Second, provides fast memory data access throughout ULSA operation. Third, performs efficient data transfer between accelerator and host at the end of the calculation. Each AGU consists of two building blocks: address transformer (AT) and address generator (AG). The AT performs mapping of host addresses into accelerator addresses. During the execution phase, AGs generate sequential addresses for accessing data.

A detailed structure of the system based on the array SA1, is sketched in Fig.4. The main constituents of the accelerator are: ULSA composed of n PEs, address generator units (AGU_A, AGU_B and AGU_C) and distributed memory composed of memory modules MA, MB and MC. AGU_A and AGU_B generate the addresses for accessing MA and MB, respectively. MC is distributed and consists of *n* dual-port RAMs (DPR). Write operation into DPR_i occurs during initialization and execution phase. Read operation occurs during execution and result transfer phase.

During address manipulation, we assume that memory address is composed of three parts

base address	*i*	*j*

where the part "base address" corresponds to a starting memory location of an array. Parts i and j define the offset of an element $x_{ij}\,(x \in \{a,b,c\})$ in regard to the base address. Address transformers modify only offset part of the address by converting i, j into i', j' [19,21]. Accelerator memory base address is fixed and determined by a design. For the sake of simplicity we take $n = 2^k$. Consequently, the parts i and j are k bits wide.

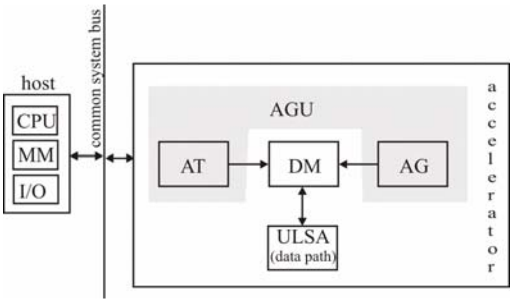


Figure 3. Global structure of the system.

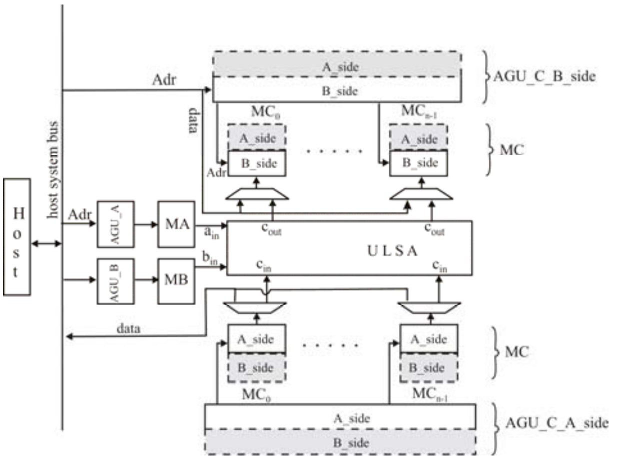


Figure 4. Structure of the system based on the array SA1.

Details which relate to the internal structure of the PE and interconnect between neighboring PEs were thoroughly described in [18-19].

V. ACCELERATOR OPERATION

Three phases characterize accelerator operation: a) Initial loading of input data A, B and C, into MA, MB and MC; b) Execution phase, and c) Result transfer.

A. Initialization phase

Suppose that matrices *A*, *B* and *C* are stored in a row-major order in the host memory. Address transformations performed by the corresponding AGUs during initial loading of MA, MB and MC can be described by the following address equations

$$f_A(i, j) = i + j * n, \tag{18}$$

for $i = 0,1,\ldots,n-1$ and $j = 0,1,\ldots,n-1$.

$$f_B(i, j) = (i * n + (n - j - 1) \bmod n), \tag{19}$$

for $i = 0,1,\ldots,n-1$ and $j = 0,1,\ldots,n-1$.

$$f_C(i, j) = ((i + j) \bmod n, i) = (MS, LS), \tag{20}$$

where *MS* defines memory module MC_i, *i*=0,1,…,*n*-1, and *LS* address location within it. According to (18), (19) and (20) direct hardware synthesis of AGUs can be performed. For more details see, for example [19].

B. Execution phase

During this phase, each PE executes identical program sequence. Functional property of PE is shown in Fig. 5. Program sequence performed by the processing element PE_i, $i = 0,1,\ldots p-1$ is given in Fig. 6.

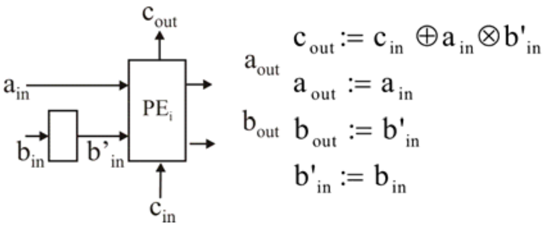
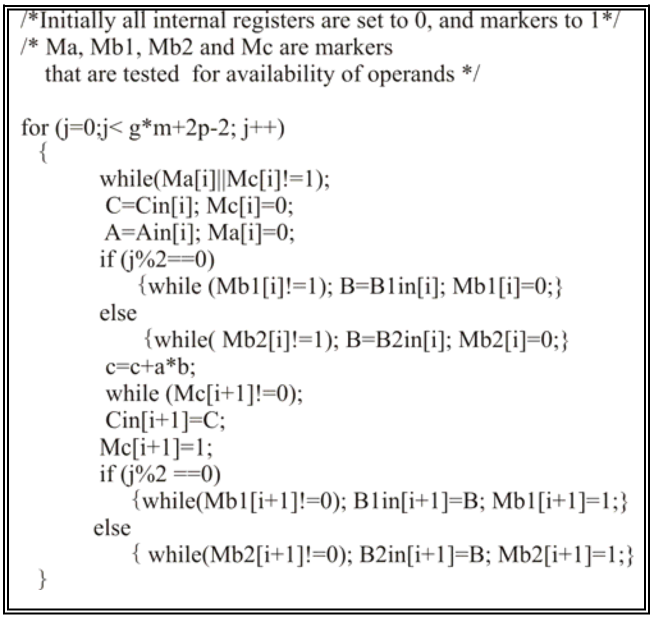


Figure 5. Functional property of PE.



```
/*Initially all internal registers are set to 0, and markers to 1*/
/* Ma, Mb1, Mb2 and Mc are markers
   that are tested  for availability of operands */

for (j=0;j< g*m+2p-2; j++)
  {
        while(Ma[i]||Mc[i]!=1);
         C=Cin[i]; Mc[i]=0;
         A=Ain[i]; Ma[i]=0;
        if (j%2==0)
             {while (Mb1[i]!=1); B=B1in[i]; Mb1[i]=0;}
        else
              {while( Mb2[i]!=1); B=B2in[i]; Mb2[i]=0;}
         c=c+a*b;
         while (Mc[i+1]!=0);
         Cin[i+1]=C;
        Mc[i+1]=1;
        if (j%2 ==0)
            {while(Mb1[i+1]!=0); B1in[i+1]=B; Mb1[i+1]=1;}
       else
            { while(Mb2[i+1]!=0); B2in[i+1]=B; Mb2[i+1]=1;}
  }
```

Figure 6. Program sequence performed by the PE_i during execution phase.

The algorithm is executed in *n* iterations. Each iteration requires 2*n*-1 instruction cycles, and, from data-flow point of view, consists of two sub-phases: passive, P_comp, and active computation, A_comp (see Fig. 7).

VI. PERFORMANCE EVALUATION

For evaluation of our design we will use quantitative and qualitative criteria. The former includes number of PEs, execution time, and PE's and AGUs' hardware complexity in terms of equivalent gate count (two input NAND gate). The latter includes speedup, efficiency, gain factor and overhead coming from AGUs hardware implementation. Together, these metrics provide a good characterization of the proposed design.

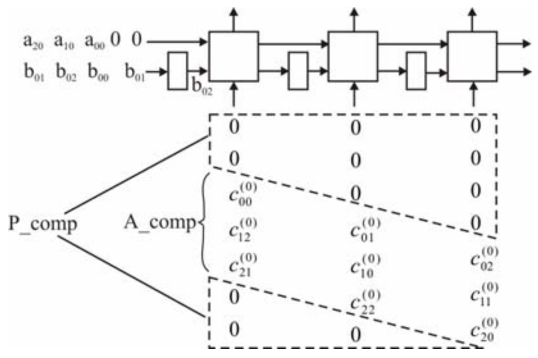


Figure 7. Data flow in the ULSA during the first iteration.

A. Quantitative measures

We assume that execution time of matrix multiplication algorithm on a uniprocessor system is equal to

$$T1 = N_1 \times N_2 \times N_3, \tag{21}$$

i.e. for the square case $T1 = n^3$, time units. Here the duration of MAC operation is taken as a time unit.

The total execution time of the algorithm on the ULSA is equal to

$$T_{tot} = T_{in} + T_{exe} + T_{out} \tag{22}$$

where T_{in} is a start-up time at the beginning of the computation, T_{exe} active execution time, and T_{out} flushing time at the end of processing.

The results that relate to T_{in}, T_{exe}, T_{out}, T_{tot} and number of PEs, Ω, are presented in Table I. Table I contains results for the arrays SA1 and SA2 of ULSA type obtained by the procedure described in Section III for the case of rectangular and square matrices. As can be seen, there is no difference between SA1 and SA2 when square matrices are considered. However, in the case of rectangular matrices, performance are different. Namely, when $N_1 < N_2$, the array denoted with SA2 is superior. In opposite, the array SA1 is better design choice.

AGUs and systolic array, composed of multi-functional PEs (see [18]), were implemented on Xilinx FPGA technology. For hardware complexity estimation ISE 9.01 software tool was used. Similarly as in [21], we have analyzed hardware complexity in terms of equivalent gate count for 32-bit PEs, for various matrix dimensions, both for Spartan 2E xc2s100e-7ft256 and Virtex 4 xc4v1x15-12sf363. The obtained results are presented in Table II.

B. Qualitative criteria

Here, we will analyze speed-up (S_p), efficiency (E), gain factor (G), and hardware overhead (O_{HW}).

Speed-up is defined as

$$S_p = \frac{T1}{T_{tot}}, \tag{23}$$

and efficiency as

$$E = \frac{S_p}{p}. \tag{24}$$

TABLE I. EXECUTION TIME AND NUMBER OF PEs OF THE ARRAYS SA1 AND SA2

ARRAY	T_{in}	T_{exe}	T_{out}	T_{tot}	Ω
SA1	N_2	$N_3(N_1+N_2-1)$	0	$N_2+N_3(N_1+N_2-1)$	N_2
	n^2	$2n^2-n$	0	$2n^2$	n
SA2	N_1	$N_3(N_1+N_2-1)$	0	$N_1+N_3(N_1+N_2-1)$	N_1
	n^2	$2n^2-n$	0	$2n^2$	n

TABLE II. IMPLEMENTATION RESULTS FOR 32-BIT PE

	32-BIT PE						
	n	AGU_A	AGU_B	AGU_C	PE	SA1	Overhead [%]
SPARTAN 2E	32	469	430	2956	21057	673824	0.574
	64	514	490	5876	21057	1347648	0.9
	128	568	508	12142	21057	2695296	0.560
	256	616	556	25636	21057	5390592	0.572
	512	664	604	54598	21057	10781184	0.599
	1024	718	652	116586	21057	21562368	0.634
	2048	783	700	248728	21057	43124736	0.673
Virtex4	32	410	334	2848	18026	576832	0.623
	64	467	400	5756	18026	1153664	0.574
	128	512	412	11992	18026	2307328	0.560
	256	524	460	25402	18026	4614656	0.572
	512	566	508	54178	18026	9229312	0.599
	1024	691	556	115791	18026	18458624	0.634
	2048	739	604	247039	18026	36917248	0.673

In the case of square matrices we have

$$S_p = \frac{T_1}{T_{tot}} = O\left(\frac{n^3}{2n^2}\right) = O\left(\frac{n}{2}\right), \quad \text{and} \quad E = \frac{S_p}{n} = \frac{1}{2} \qquad (25)$$

We define gain factor, G, as

$$G = \frac{T_{sw}}{T_{hw}}$$

where T_{sw} (T_{hw}) corresponds to time needed to transfer data from host to accelerator memory when address calculations are performed by software (by hardware AGUs). Transfer time was estimated by a profiler in *ms* as time units. Number of PEs was taken as a parameter. Reports obtained from the profiler are summarized in Table III. Clock frequency of the host was 3.3 GHz. From the obtained results an average gain factor G_{av} can be determined. For our design we have $G_{av} = 2.7$. This means that the speed up achieved by implementing address transformations in hardware is more than two and a half compared to the software implementation. However, this is achieved at the cost of increased hardware overhead.

TABLE III. REPORTS OBTAINED FROM THE PROFILER

N	AGU_A	AGU_B	AGU_C	HW	G
32	0.041	0.042	0.052	0.039	1.15384
64	0.051	0.06	0.07	0.05	1.20666
128	0.128	0.145	0.155	0.095	1.50175
256	0.562	0.463	0.59	0.259	2.07850
512	2.2846	1.769	3.05	0.915	2.58783
1024	11.828	6.722	14.461	3.186	3.45375
2048	67	25.665	68.83	11.575	4.65068
4096	284	102.733	287.462	45	4.99403
Average gain factor $G_{av} = 2.70338$					

Hardware overhead, O_{HW}, corresponds to a ratio of total number of equivalent gates in the ULSA with and without hardware AGUs. The results are presented in the last column of Table II. As can be seen, the overhead is about 0.6% both for Spartan2E and Virtex4. The obtained results clearly justify the usage of hardware AGUs.

VII. CONCLUSION

We have considered the problem of matrix multiplication on a linear systolic arrays with unidirectional data flow (ULSA). First, we have described mathematical procedure for systolic array synthesis. In order to speed-up data transfer between the host and ULSA, used as an accelerator, we have designed hardware address generator units (AGUs). We have implemented ULSA and AGUs on FPGA technology. Performance of the ULSA and hardware AGUs from aspect of speed-up, efficiency, gain factor and hardware overhead, were estimated. The obtained results show that the speed-up of the ULSA composed of n PEs is $O(n/2)$ and efficiency is 0.5. Gain factor and hardware overhead were used to estimate performance of hardware AGUs. The obtained results show that average gain factor is 2.7, with hardware overhead of at most 0.623%.

REFERENCES

[1] J. Jang, S. Choi, V.K. Prasanna, "Area and time efficient implementations of matrix multiplication on FPGAs", In Proc. 1st IEEE International Conf. Field-Programmable Technology (ETP'02), Hong Kong, 2002, pp. 93-100. [Online]. Available: http://dx.doi.org/10.1109/FPT.2002.1188669

[2] A. Amira, P. Bouridane, A. Milligan, "Accelerating Matrix Product on Reconfigurable Hardware for Signal Processing", In Proc. 11th International Conf. Field- Programmable Logic Appl. (FPL'01), Sidney, Australia, 2001, pp. 101-111.

[3] F. Bensaali, A. Amira, A. Bouridane, " Accelerating matrix product on reconfigurable hardware for image processing applications", IEE Proceedings -- Circuits, Devices and Systems, Vol. 152, No 3, pp. 236-246, june 2005.

[4] L. Jianwen, J.C. Chuen, "Partially reconfigurable matrix multiplication for area and time efficiency on FPGAs", In Proc. 8th Euromicro Symp. Digital System Design (DSD2004), Renes, France, 2004, pp. 244- 248.

[5] P. Zicari, P. Corsonello, S. Perri, G. Cocorullo, "A matrix product accelerator for field programmable systems on chip", Microprocessors and Microsystems, Vol. 32, No 2, pp. 53-67, March 2008.

[6] I. Ž. Milovanović, E. I. Milovanović, B. M. Randjelović, I. Č. Jovanović, "Matrix mutiplication on a bidirectional systolic arrays", FILOMAT, Vol. 17, No 1, pp. 135-141, 2003.

[7] D. I. Moldovan, "On the design of algorithms to VLSI systolic arrays", Proceedings of the IEEE, Vol. 71, No 1, pp. 113-120, 1983.

[8] D. Grant, P. Denyer, I. Finlay, "Synthesis of address generators", In Proc IEEE Int. Conf. Computer Aided Design (ICCAD-89), Santa Clara, CA, 1989, pp. 116-119.

[9] H. T. Kung, "Why systolic architectures?", Computer, Vol. 15, No 1, pp. 37-46, January 1982.

[10] V. V. Voevodin, Mathematical models and methods in parallel processing, Nauka, Moscow, 1986 (In Russian).

[11] I. Ž. Milovanović, E. I. Milovanović, M. P. Bekakos, "Synthesis of unidirectional systolic array for matrix-vector multiplication", Math. Comput. Modelling, Vol. 43, No 1-2, pp. 612-619, March 2006.

[12] E. I. Milovanović, M. P. Bekakos, I. Ž. Milovanović, "Synthesis of space optimal systolic arrays for band matrix-vector multiplication", J. Supercomputing, Vol. 49, No 3, pp. 269-290, September 2009.

[13] M. P. Bekakos, I. Ž. Milovanović, E. I. Milovanović, T. I. Tokić, M. K. Stojčev, Hexagonal systolic arrays for matrix multiplication, In: Highly Parallel Computations" Algorithms and Applications (M.P. Bekakos, ed.), Chapter 6, WIT Press, Southampton-Boston, 2001, 139-177.

[14] S. G. Sedukhin, "The Designing and Analysis of Systolic Algorithms and Structures", Programming, 2, pp. 20-40, 1991, (In Russian.)

[15] K. Compton, S. Hauck, "Reconfigurable computing: A syrvey of systems and software", ACM Comput. Surveys, Vol. 34, No 2, pp. 171-210, June 2002.

[16] M. Hertz, R. Hartenstein, M. Miranda, E. Brockmeyer, F. Catthoor, "Memory Addressing Organization for Stream-based Reconfigurable Computing", Proc. IEEE ICECS 2002, Dubrovnik, Croatia, 2002, pp. 813-817. [Online]. Available: http://dx.doi.org/10.1109/ICECS.2002.1046298

[17] G. Talavera, M. Jayapala, J. Carrabina, F. Catthoor, "Address generation optimization for embedded high-performance Processors: A Survey", J. Sign. Process. Syst., Vol. 53, No 3, pp. 271-284, December 2008.

[18] E. I. Milovanović, T. R. Nikolić, M. K. Stojčev, I. Ž. Milovanović, "Multi-functional systolic array with reconfgurable micro-power processing elements", Microelectron. Reliab., Vol. 49, No 7, pp. 813-820, July 2009.

[19] M. K. Stojčev, I. Ž. Milovanović, E. I. Milovanović, T. R. Nikolić, "Address generators for linear systolic array", Microelectron. Reliab., Vol. 50, No 2, pp. 292-303, February 2010.

[20] Embeded Development HW/SW kit, Spartan 3A DSP S3D1800A, MicroBlaze Processor Edition.

[21] I. Ž. Milovanovic, M. K. Stojcev, E. I. Milovanovic, T. R. Nikolic, "Linear Processor array in DSP", Proc. 28th International Conference on Microelectronics (MIEL 2012), Niš, Serbia, 13-16 May, 2012, pp. 387-392, ISBN 978-1-4673-0236-4