Mapping full-systolic arrays for matrix product on XILINX's XC4000(E,EX) FPGAs

Article *in* COMPEL The International Journal for Computation and Mathematics in Electrical and Electronic Engineering · March 2002

DOI: 10.1108/03321640210410751

3 authors, including:

Abdelkrim Kamel Oudjida
Centre de Développement des Technologies Avancées
68 PUBLICATIONS **489** CITATIONS
SEE PROFILE

Mustapha Hamerlain
Centre de Développement des Technologies Avancées, EMP
128 PUBLICATIONS **1,148** CITATIONS
SEE PROFILE

Mapping full-systolic arrays for matrix product on XILINX's XC4000(E,EX) FPGAs

Received August 2000
Accepted February 2001

A.K. Oudjida, S. Titri and M. Hamerlain
CDTA/Microelectronics and Robotics Laboratories, Algiers, Algeria

Keywords *Robotics, Algorithms, Control systems*

Abstract *Matrix product is a compute bound problem that can be efficiently handled by elementary systolic algorithms. From a theoretical point of view, most of the algorithms are very simple and sometimes even trivial. However, the task of designing efficient implementation on a fixed-connection network, such as on FPGA where resources are very limited, has been more demanding, and sometimes quite tedious. The objective of this paper is twofold: we first describe a full-systolic algorithm for matrix product that has the merit over its existing counterparts, to require no preloading of input data into elementary processors (EPs) and generates output data only from boundary EPs. The resulting architecture can accept an uninterrupted stream of input data and produces an uninterrupted one with a latency of 2N-1 for N × N matrix product. This architecture is also scalable and complies with the constraint of problem-size independence (ψ). Secondly, we present a methodology for generating a family of very compact MP arrays on FPGA based essentially upon manual mapping at CLB level coupled with VHDL structural level.*

1. Introduction

This work is a part of a challenging project aimed at exploring the possibility of a VLSI implementation of the explicit dynamic model of a robotic arm (Armstrong *et al.*, 1986). This issue did not raise much interest in the past because the computation requirements were prohibitive with respect to the limited VLSI computational power available. However, as VLSI technology has dramatically evolved in the last decade, many interesting solutions that have so far remained unimplemented due to their excessive speed demands have been reconsidered. Consequently, this contribution is a first step of a long and complex process, that of designing a VLSI architecture that provides sufficient computing power to realize advanced robotic control system in real time.

The dynamic model used for this purpose follows from Liègois *et al.* (1977). It is:

$$A(q)q + B(q)[qq] + C(q)\left[q^2\right] + g(q) = \Gamma,$$

where: $A(q)$ is the $N \times N$ kinetic energy matrix, $B(q)$ is the $N \times N(N\text{-}1)/2$ matrix of Coriolis torques, $C(q)$ is the $N \times N$ matrix of centrifugal torques, $g(q)$ is the N vector of accelerations, q is the N vector of accelerations, Γ is the generalized joint force vector, $[qq]$ is the $N(N\text{-}1)/2$ vector of velocity products and $[q]$ is the N vector of square velocity.

The evaluation of this expression can be logically decomposed into two subsequent sub-operations:

(1) generation of the different matrices and vectors (precalculation of the elements of each matrix and vector); and
(2) calculation of the matrix-products.

This paper deals exclusively with the second sub-operation and is mainly focused on the implementation issues that arises form matrix-product problems using FPGA.

The paper is organized as follows. In this section, the motivating idea behind designing fast matrix-product architecture is outlined. Section 2 sets some specifications and constraints according to the intended objectives. Section 3 presents a comparative study of different MP architectures showing their pros and cons. Section 4 gives a detailed description of our full-systolic architecture and discusses both the scalability and the ψ problems. Section 5 introduces a methodology for mapping generic VHDL-described MP arrays on Xilinx's FPGAs and, finally, some concluding remarks are in Section 6.

2. Design considerations

As our ultimate objective is to design a MP architecture that will be used as a macro-function within a larger system, special care must be devoted to the computation speed since it constitutes a critical factor limiting the overall system performance. With FPGA, the size also becomes a major hurdle, since the area in term of CLBs is very limited.

In order to reconcile these two antagonistic constraints into a good compromise, a number of decisions were made.

- *Parallelism*: to multiply $N \times N$ matrix, it takes $O(N^3)$ steps when using the sequential algorithm. To reduce the complexity to $O(N)$, the architecture must be two-dimensional.
- *Pipelinability*: to cope with an uninterrupted stream of input data, the architecture must be pipelined.
- *Latency*: to maximize the throughput rate while minimizing the latency, the architecture must require no preloading of input data and must generate output data only from the boundary.
- *Regularity*: to easily synthesize MP architecture of any size, the architecture must exhibits a high degree of regularity.
- *Systolicity*: to respect the above-mentioned constraints, the full-systolic principle must be applied.

- *Scalability*: to maintain latency linearly proportional to the number of EPs whatever the size of the problem (N), the architecture must be scalable.
- *Problem-size independence* (ψ): to cope with any size of problem (N) when just a smaller number P of EPs ($P \ll N^2$) can be implemented, the architecture must comply with the ψ constraint.
- *Fast prototyping*:: to reduce the design cycle, the final architecture is to be implemented on Xilinx's FPGAs.
- *I/O bandwidth*: as the number of IOBs of even the largest Xilinx's XC4000(E,EX) FPGA circuit (XC4044EX) is very limited (320) (Xilinx, 1996), the I/O bandwidth is fixed to 2N, which is a well balanced compromise between speed and available IOBs.
- *Partitioning and Mapping*: to minimize as much as possible the number of CLBs, the architecture must be manually partitioned and mapped at CLB level.
- *Generation*: to guarantee 100% conformity between synthesis results and design objectives, the generator must be written in VHDL structural level with explicit gate binding.

3. Existing matrix-product algorithms

Although the literature on MP is extensive (International Conference on Systolic Arrays, 1988; Leighton, 1992; Mead and Conway, 1983; Quinton and Robert, 1989; Swartzlander, 1996) and the topic is old, to the best of the authors' knowledge there is no comprehensive analysis of the MP problem using Xilinx's FPGA. By comprehensive analysis, we apply a treatment that starts from an algorithm and goes down to the actual FPGA implementation.

In our attempt to solve this problem according to the above-established constraints, we first made a synthesis of most of the existing algorithms (Benaini and Robert, 1989; Jagadish and Kailath, 1989; Kak, 1988; Kung, 1980; Kung and Leiserson, 1978; Li and Wah, 1985; Porter and Aravena, 1987, 1988; Tsay and Chang, 1995a, b). For the most part, the algorithms are efficient in that they achieve near optimal speedups $O(N)$ over the corresponding sequential algorithms $O(N^3)$. Unfortunately, none of them complies with the *full-systolicity* constraint. This is well illustrated by designs (Porter and Aravena, 1987; Tsay and Chang, 1995a,b) described by Figures 1-3 and considered in the literature as the three *fastest designs* that have been developed so far. Because these designs are not full-systolic (semi-systolic), an EP draining is required at the end of the multiplication process. Worse, in the orbital array (Figure 3) an EP preloading is also required. The pre- or post-operations performed before or after the actual treatment induce a severe limitation in that they stop the continuous flow of input data and, therefore, make the cascading of MP operations impossible. These extra operations yield a real performance degradation since an *extra time* is required that may be as

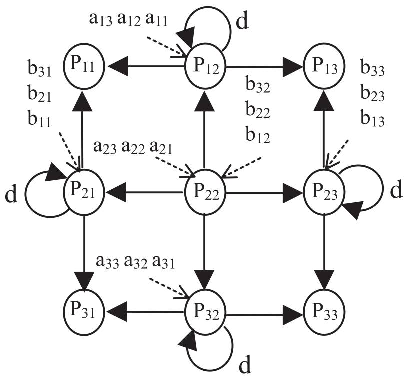

Figure 1.
Mesh array

important as the latency itself. They also require extra control circuitry, inevitably disturbing the high regularity of the array.

The designs of Figures 1 and 2 (Tsay and Chang, 1995a,b) each present one major drawback. In Figure 1, the number of delay elements added in each EP according to the expression $d = \|i - \lceil N/2 \rceil| - |j - \lceil N/2 \rceil\|$ varies from 1 to $N/2 - 1$. Therefore, it becomes prohibitive to use it for typical values of N and in Figure 2, a result reordering is needed at the end of the multiplication process, requiring once more an extra time.

Our algorithm, as will be illustrated in the next section, circumvents all these severe shortcomings while remaining faithful to the initial specifications, and

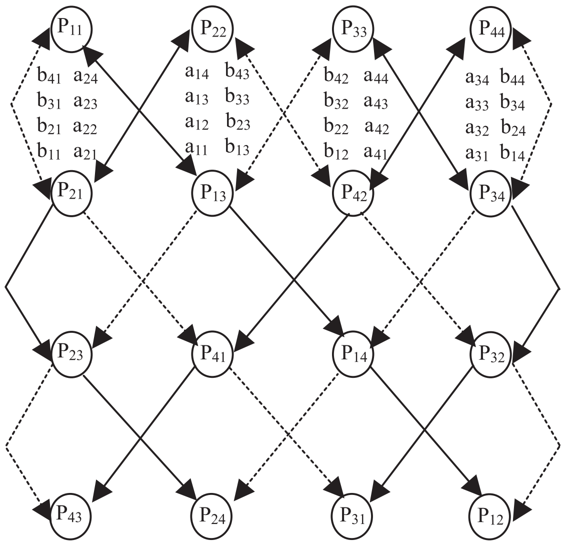

Figure 2.
Two-layered mesh array

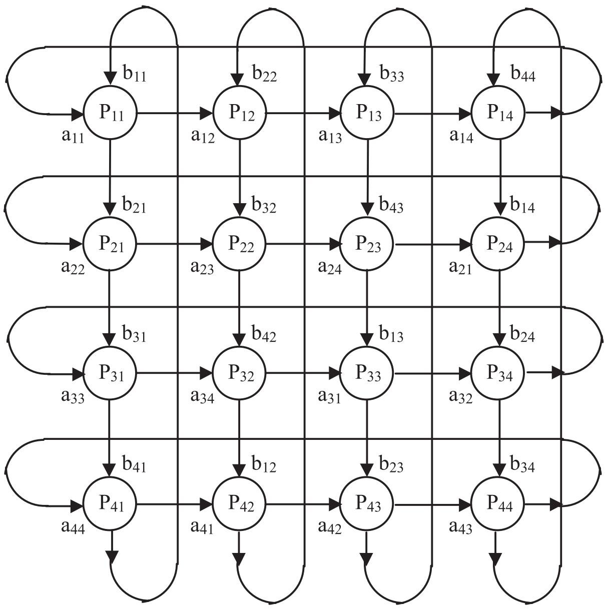

Figure 3.
Orbital array

MP algorithms	Full systolicity	Latency (without extra time)	I/O bandwidth
Figure 1	No	$3N/2$	$2N$
Figure 2	No	$3N/2$	$2N$
Figure 3	No	N	$2N^2$
Figure 4	Yes	$2N-1$	$2N$

Table I.
Comparison of different MP algorithms

also offers better performance over its traditional counterparts when considering the extra time needed (Table I).

4. Our full-systolic architecture

As full-systolicity compliance was our primary concern, we tried throughout this project to adopt, wherever possible, the best working possibilities that fit this requirement. As a first choice, the top/down design methodology was employed because it offers the possibility to make fast architecture explorations and to come to rapid decisions.

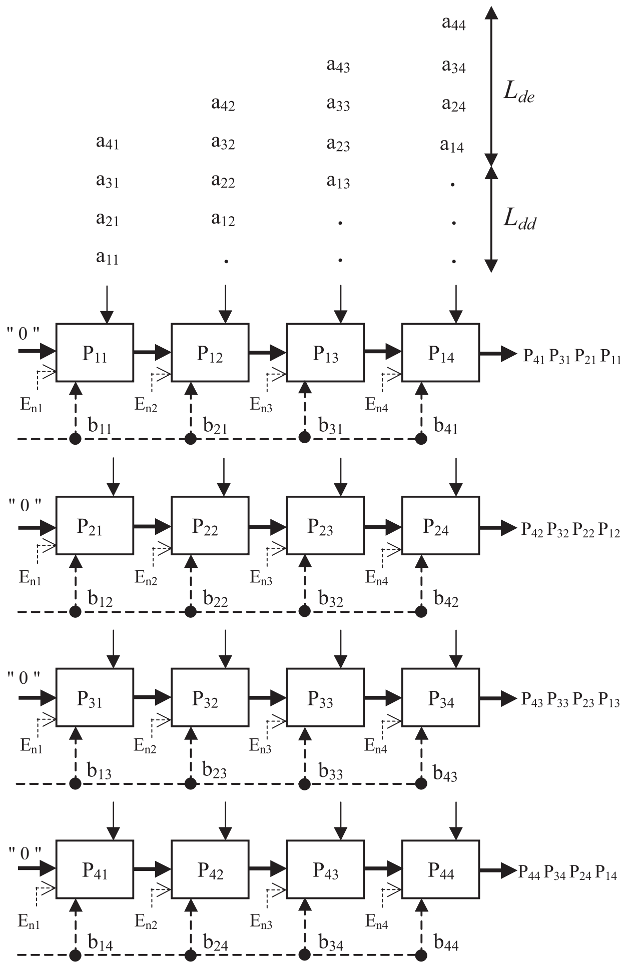

Figure 4.
Our full-systolic array

4.1 Description of the architecture
The resulting architecture is described in Figure 4. It is built up exclusively of two blocks: the EP (Figure 6) and the controller (Figure 5) Each EP performs a multiply-and-accumulate (MAC) operation where the vertical (a_{ij}) and horizontal (b_{ij}) inputs are multiplied and added to the preceding PPin value and the result is propagated to the next EP via PPout. The controller (En_j) serves to successively and selectively validate each clock cycle the registers of the appropriate j-th EP column. It is noteworthy that the data inputs are injected line-by-line and the results are alsoproduced line-by-line. Each line i of

the EPs independently performs a matrix–vector product ($A \times B_j$). This feature serves as the basis for the resolution of the ψ problem as will be explained below. Just one pulse exerted by the host machine on the START command at the beginning of the treatment is sufficient to perform an uninterrupted stream of matrix products (Figure 7).

4.2 Latency

Conceptually, latency (L) is defined as the time between the first entry of elements of matrices A and B, until the last element of the matrix P is calculated. Graphically, L corresponds to the time needed for data to move along the critical path of the design. Note that in Figure 4, L can be decomposed into two independent parts: L_{de}; and L_{dd}. So, the total latency can be written as:

$$L = L_{de} + L_{dd}$$

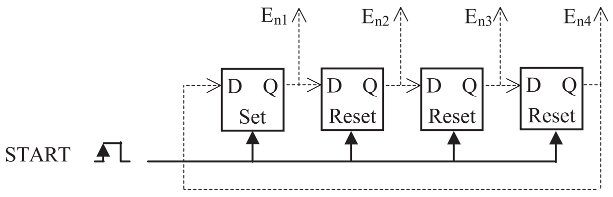

Figure 5.
The controller

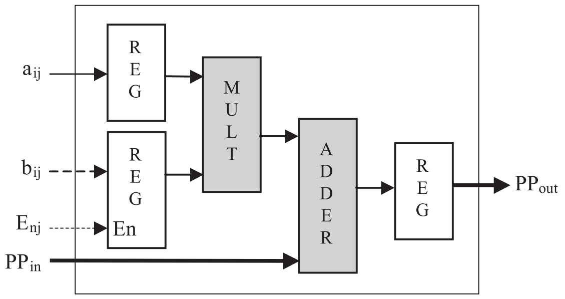

Figure 6.
EP architecture

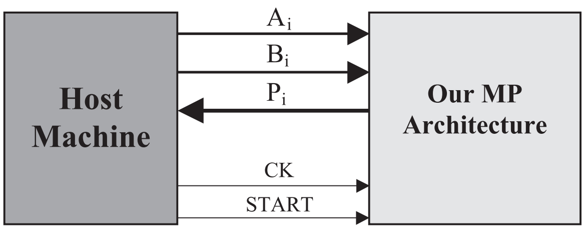

Figure 7.
Interface host/MP

where L_{de} (data entry time): is the time between the entry of the first element and the entry of the last element of the critical path, L_{dd} (data delay time): is the number of steps that data are delayed before entering the first EP, $L_{de} = N$ and $L_{dd} = N - 1$. So, $L = 2N - 1$.

4.3 Implementation on Xilinx's FPGA

As our algorithm is two dimensional, the area required in terms of CLB is very excessive. To minimize it as much as possible, a special effort was focused on the design of the EP cell since it constitutes the main building block of the architecture. Apart from the I/O registers of the EP, the MAC function is very space consuming (CLBs) since the multiply algorithm must also be a two-dimensional one, as stated by the first specification.

We developed for that purpose our own multiplier based on the Modified Booth Algorithm (MBA) (Oudjida, 1997). Our choice went to MBA because it leads to 25 per cent speedup over its conventional counterparts. The whole architecture was manually partitioned and mapped at CLB level. As a result, an average of 35 per cent savings were achieved over existing Xilinx's multiplier cores (Table II).

As an application, a 6×6 by 6×1 MP architecture (corresponding to the PUMA 560 arm; Armstrong *et al.*, 1986) was mapped on Xilinx's XC4020EHQ208-3 package consuming 385 CLBs and exhibiting a delay (clock-to-setup) time of 65 ns.

4.4 Scalability

In pure systolic arrays, only short interconnects are used, in that each EP interacts only with its nearest surrounding neighbours. In this case, the ability to maintain latency linearly proportional to the number of EPs is guaranteed. However, in our case, where the architecture is not pure systolic, but rather a mixture of parallel/systolic (but full-systolic), the problem of scalability may occur. This is because a_{ij}, b_{ij} as well as En_j lines are broadcasted simultaneously to N EPs. The fanin becomes very important and, hence, the delay to charge these high capacitances may surpass that of the EP to perform the MAC function.

In such a case, we cannot say any more that L is still equal to $2N - 1$ because we have either to enlarge the clock cycle as much as it is needed to charge these high capacitances, which leads to a rapid performance

Table II.
Comparison in term of CLBs

Operand size	Xilinx's multipliers	Our multipliers	Saving (%)
8	57–73	43	24–40
16	230–270	151	34–44

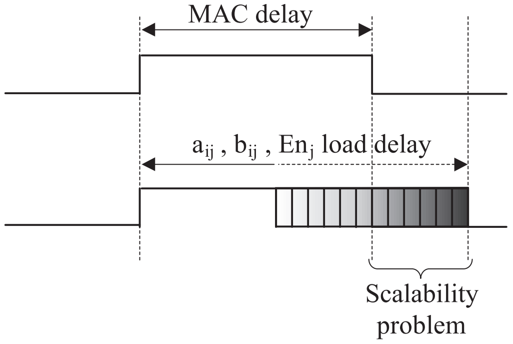

Figure 8.
Scalability problem

degradation, or to split the array into as many sub-arrays as necessary and insert registers between them (Figure 8).

In Figure 9, we split the 4×4 array into $(4/2)^2 = 4$ 2×2 sub-arrays. Latency in this special case is increased by 2 clock cycles since $4/2 = 2$ registers are inserted in the critical path. Likewise, when an $N \times N$ array is split into $(N/k)^2$ $k \times k$sub-arrays, N/k registers are added and, hence, $L = 2N - 1 + (N/k)$.

4.5 Problem-size independence (ψ)

A parallel algorithm or architecture is ψ if, with the given finite size hardware, any instance of the problem can be solved. This implies that the problem can be decomposed, the components solved using the available hardware, and the results put back together to give the complete solution. The definition simply reflects the real life solution where more often the available hardware is much smaller than the input size of the problem that is required to be solved.

In MP, we consider that the size N^2 of the matrix is much larger than the available number P of EPs of the array. As previously stated, in Subsection 4.1, this architecture presents the specificity that each line of EPs performs independently a matrix–vector ($A \times B_j$) product. With $P = kN$ EPs available, only k vectors can be calculated. To calculate the whole set of vectors, we have to reiterate this process $N/k = N^2/P$ times and since each process requires $2N - 1$ cycles, the total latency then becomes:

$$L = (2N - 1) \times \left(N^2/P\right) = \left(2N^3 - N^2\right)/P.$$

5. Generation

The key to successful application of VHDL synthesis is the use of a total design methodology that will yield predictable, accurate and high quality results (Oudjida, 1997). To be successful a design needs to meet the functional, area and timing goals. The style used for design description is a powerful way one

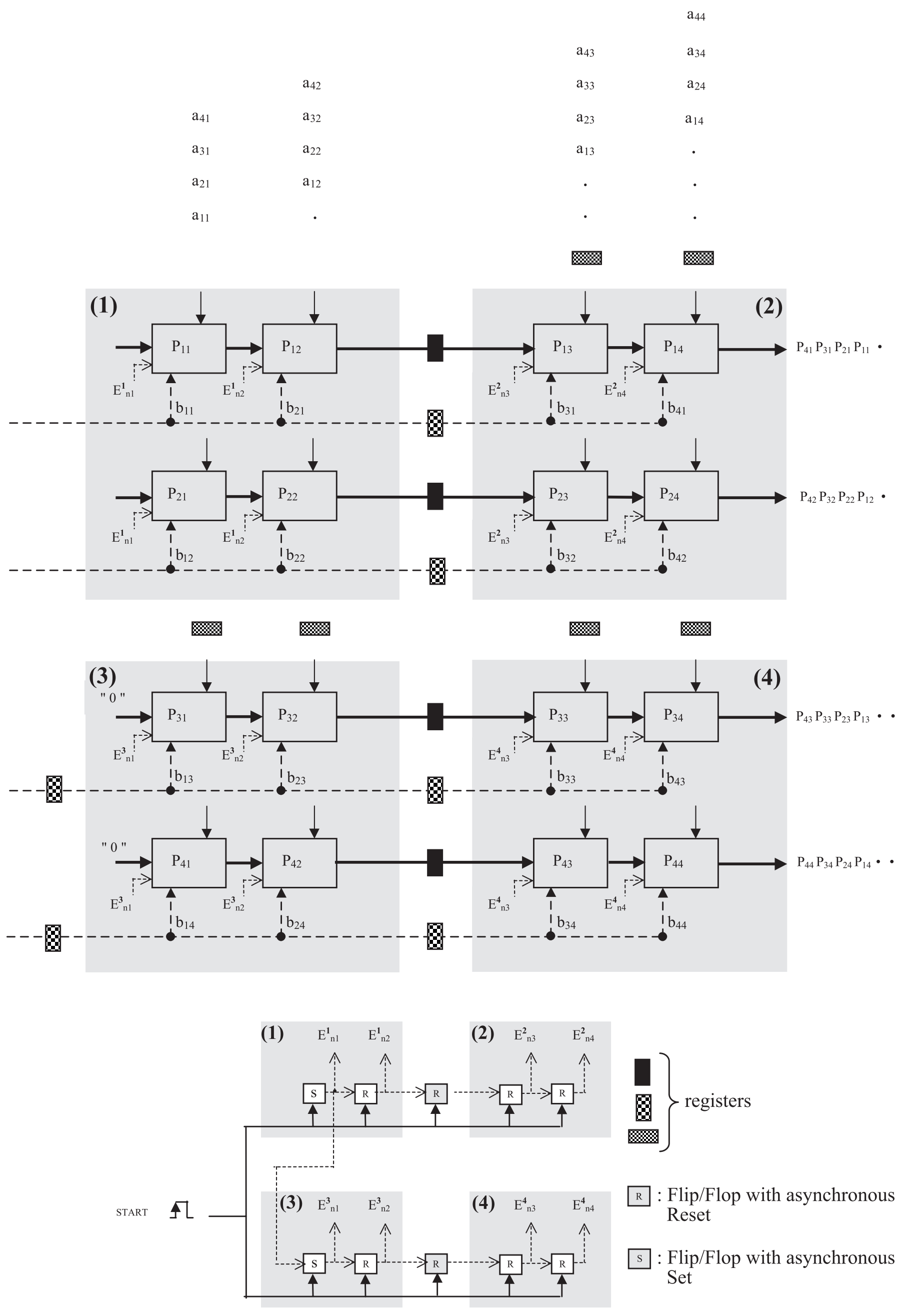

Figure 9.
Scalable array

can control the synthesis process because different VHDL descriptions of the same functionality can yield radically different networks. The power to control the synthesis process is the power to express the implementation intent. The resultant implementation quality comes from both the description style used by the designer and the power of the synthesis tool.

In our case, the synthesis process has been drastically simplified since the EP cell which is the main building block of the array was already manually designed at CLB level. So, apart the controller that requires a translation from RTL to structural level, the main task of the synthesizer is to exclusively handle problems of repetitiveness and interconnections. Because a purely structural-level description with explicit binding is used, 100 per cent conformity is guaranteed in terms of area between the intended implementation and the synthesis result. Only a slight timing overhead occurs due to interconnection routing.

As the architecture is very regular, the generator program may extend by a step of one on both A and B matrices. The generator program is written in VHDL structural level (see the Appendix).

6. Concluding remarks

This paper provides an overview of work ongoing at the Robotics laboratory with collaboration of Microelectronics laboratory, aiming at exploring the possibility of VLSI implementation of the explicit dynamic model of a robotic arm for real-time control applications. This work, which is a small part of a larger project, addressed the problem of full systolicity compliance of MP architectures.

Early in the design process we outlined main directions and established some decisions for a best trade-off between speed and area. The results were as follows.

- Full systolic architecture capable of handling an uninterrupted steam of data.
- Scalable and problem-size independent architecture.
- Latency of $2N - 1$.
- Most compact architecture since it is built up of the most compact multiplier core.
- Very regular architecture that may extend to any size of the operands A and B.
- 65 ns clock rate on Xilinx's XC4020EHQ208-3 package.

Further improvements of the throughput rate may be achieved by using the micro-pipelining technique inside the EP cell; however, the major drawback is that more area is required.

References

Armstrong, B., Khatib, O. and Burdick J. (1986), "The explicit dynamic model and inertial parameters of the Puma 560 arm", *The International Conference on Robotics and Automation*, CA.

Benaini, A. and Robert, Y. (1989) "An even faster array for matrix multiplication", *Parallel Computing*, Vol. 12, pp. 249-54.

International Conference on Systolic Arrays, May 25–27 1988, San Diego, CA.

Jagadish, H.V. and Kailath (1989) "A family of a new efficient arrays for matrix multiplication", *IEEE Trans. on Computers*, Vol. 38, pp. 149-55.

Kak, S.C. (1988) "A two-layered mesh array for matrix multiplication", *Parallel Computing*, Vol. 6, pp. 383-85.

Kung S.Y. (1980), "VLSI array processor for signal processing", *Proc. Conf. Advanced Res. Integrated Circuits*.

Kung, H.T. and Leiserson, C.E. (1978) "Systolic Arrays for VLSI", *Sparse Matrix Symposium, SIAM*, pp. 256-82.

Leighton, F.T. (1992), *Introduction to Parallel Algorithms and Architecture*, Morgan Kaufmann.

Li, G.J. and Wah, B.W. (1985) "The design of optimal systolic arrays", *IEEE Trans. on Computers*, Vol. C-34, pp. 66–77.

Liègois, *et al.* (1977), *Mathematical and Computer Models of Interconnected Mechanical Systems*, Elsevier Science, Amsterdam.

Mead and Conway, L. (1983), "Introduction aux system VLSI", *Interedition*, Paris.

Oudjida, A.K. (1990), "Design and analysis of a high performance multiplier and its generator", *The International Conference on Microelectronics ICM'90*, October, Damascus.

Oudjida, A.K. (1994), "VLSI implementation of highly regular structures from VHDL structural-level", *The Sixth International Conference on Microelectronics ICM'94*, September 5-7, Istanbul.

Oudjida, A.K. (1996), "A high speed and very compact two's complement serial/parallel multipliers using Xilinx's FPGAs", *International Conference on Signal Processing Applications & Technology ICSPAT'96*, Boston, MA.

Oudjida, A.K. (1997), "A 43 CLBs 8 × 8 bit two's complement parallel multiplier using Xilinx's XC4000(E) FPGAs", *International Conference on Signal Processing Applications & Technology ICSPAT'97*, Boston, MA.

Porter, W.A. and Aravena, J.L. (1987) "Orbital architecture with dynamic reconfiguration", *IEEE Proc.*, Vol. 134, pp. 281-87.

Porter, W.A. and Aravena, J.L. (1988) "Cylindrical arrays for matrix multiplication", in Proc. 24th Annual Alleroton Conf. Commun., Control, Computing, vol. 6 pp. 595-602.

Quinton, P. and Robert, Y. (1989), *Algorithmes et Architectures Systolique*, Edition Masson.

Swartzlander, E.E. (1996), *Application Specific Processors*, Kluwer Academic, Dordrecht.

Tsay, J.C. and Chang, P.Y. (1995a) "Design of efficient regular arrays for matrix multiplication by two-step regularization", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 6, pp. 2.

Tsay, J.C. and Chang, P.Y. (1995b) "Some new designs of 2D-array for matrix multiplication and transitive closure", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 6, pp. 4.

Xilinx, (1996), *The Programmable Logic Data Book*, Xilinx.

Appendix: VHDL Structural-level Generator of 6 × 6 by 6 × 1 Matrix Product

```
library lib,ieee ;
use ieee.numeric_bit.all;
use lib.my_type.all;

entity matrix is
port (Aj : in bit8_vector(1 to AN_C);
      Bj : in bit8_vector(1 to BN_C);
      Pj : out bit20_vector(1 to BN_C);
      CK,RST : in bit);
end matrix;
architecture structural of matrix is
 component PEIN
 port (Ain,Bin : in bit8;
       Pout : out bit20;
       CK,En : in bit);
 end component;
 component PE
 port (Ain,Bin : in bit8;
       PPin : in  bit20;
       Pout : out bit20;
       CK,EN : in bit);
 end component;
 component FD8CE
 port (D : in bit8;
       CE : in bit;
       CLR : in bit;
       Q : out bit8;
       C : in bit);
 end component;
 component CONTROL
 port (CK,RST : in bit;
       ENj : buffer unsigned(AN_C downto 1));
 end component;
 component BUFG
 port (I : in bit;
       O : out bit);
 end component;
 attribute noopt: boolean;
 attribute noopt of PE,PEIN,FD8CE,BUFG:
                     component is true;
 signal ENj : unsigned(AN_C downto 1);
 signal CKBUF : bit ;
 signal un : bit := '1';
 signal zero : bit := '0';
 constant N1 : integer := BN_C*(AN_C-1);
 constant N2 : integer := (AN_C*(AN_C-1))/2;
 signal acc : bit20_vector(1 to N1);
 signal lat : bit8_vector(1 to N2);
begin
 BUFG_Inst: BUFG port map(CK,CKBUF);
 CONT_Inst: CONTROL port map (CKBUF,RST,ENj);
 LAT_array: for i in 1 to AN_C-1 generate
           LAT_col: for j in 1 to i generate
                    LATIN: if (j=1) generate
                           LATIN_inst: FD8CE port map (Aj(i+1),un,zero,lat(i*(i-1)/2+1),CKBUF);
                    end generate LATIN;
                    LATOTHERS: if (j/=1) generate
                               LATOTH_inst: FD8CE port map (lat(i*(i-1)/2+j-1),un,zero,lat(i*(i-
1)/2+j),CKBUF);
                    end generate LATOTHERS;
           end generate LAT_col;
 end generate LAT_array;
 PEIN_array: for i in 1 to BN_C generate
            PEIN_inst: PEIN port map(Aj(1),Bj(i),acc(i),CKBUF,ENj(1));
 end generate PEIN_array;
 Test: if (AN_C>2) generate
       PE_array: for i in 2 to AN_C-1 generate
                PP: for j in 1 to BN_C generate
                    PE_inst: PE port map(lat(i*(i-1)/2),Bj(j),acc(j+BN_C*(i-2)),acc(j+BN_C*(i-
1)),CKBUF,ENj(i));
                end generate PP;
       end generate PE_array;
 end generate Test;
 PEOUT_array: for i in 1 to BN_C generate
           [illegible]
```

```
package MY_type is

constant AN_L : integer :=6;
constant AN_C : integer :=6;
constant BN_L : integer :=6;
constant BN_C : integer :=1;
subtype bit8 is bit_vector(7 downto 0);
subtype bit16 is bit_vector(15 downto 0);
subtype bit20 is bit_vector(19 downto 0);
type bit8_vector is array (natural range <>) of bit8;
type bit16_vector is array (natural range<>) of
bit16;
type bit20_vector is array (natural range<>) of
bit20;
```

```
entity controller is
port(CK,RST:in bit;
     ENj : buffer unsigned(AN_C downto 1):=
to_unsigned(1,AN_C));
end control;

architecture behavioral of controller is
begin
 process(CK,RST)
 variable D : integer range 0 to
to_integer(abs(to_signed(AN_L-BN_L,8)));
 begin
  if (RST='1') then ENj<=to_unsigned(1,AN_C); D:=0;
  elsif (CK='1' and CK'event) then
       if ((AN_L > BN_L) and (ENj(AN_C)='1')) then
         ENj <= to_unsigned(0,AN_C);
       elsif (ENj/=to_unsigned(0,AN_C)) then
            ENj <= ROTATE_LEFT(ENj,1);
       else D:=D+1;
            if D=AN_L-BN_L then
              ENj<=to_unsigned(1,AN_C);
              D:=0;
            end if;
       end if;
  end if;
 end process;
end;
```