# FPGA Design and Implementation of Matrix Multiplier Architectures for Image and Signal Processing Applications

**Syed M. Qasim, Ahmed A. Telba and Abdulhameed Y. AlMazroo**

King Saud University, College of Engineering, Department of Electrical Engineering, Riyadh 11421, Saudi Arabia

**Summary**
Matrix multiplication is the kernel operation used in many image and signal processing applications. In this paper, we present the design and Field Programmable Gate Array (FPGA) implementation of matrix multiplier architectures for use in image and signal processing applications. The designs are optimized for speed which is the main requirement in these applications. First design involves computation of dense matrix-vector multiplication which is used in image processing application. The design has been implemented on Virtex-4 FPGA and the performance is evaluated by computing the execution time on FPGA. Implementation results demonstrate that it can provide a throughput of 16970 frames per second which is quite adequate for most image processing applications. The second design involves multiplication of tri-matrix (three matrices) which is used in signal processing application. The proposed design for the multiplication of three matrices has been implemented on Spartan-3 and Virtex-II Pro platform FPGAs respectively. Implementation results are presented which demonstrate the suitability of FPGAs for such applications.

***Key words:***
*FPGA, Matrix Multiplier, Systolic Array, VLSI.*

## 1. Introduction

Computation intensive algorithms used in image and signal processing, multimedia, telecommunications, cryptography, networking and computation domains in general were first realized using software running on Digital Signal Processors (DSPs) or General Purpose Processors (GPPs). Significant speed-up in computation time can be achieved by assigning complex computation intensive tasks to hardware and by exploiting the parallelism in algorithms [1].
Recently, Field Programmable Gate Arrays (FPGAs) have become a platform of choice for hardware realization of computation-intensive applications [1-13]. Especially, when the design at hand requires very high performance, designers can benefit from high density and high performance FPGAs instead of costly multicore Digital Signal Processing (DSP) systems [1]. FPGAs enable a high degree of parallelism and can achieve orders of magnitude speedup over GPPs [7]. This is as a result of the increasing embedded resources on FPGA. FPGA have the benefits of the hardware speed and the software flexibility; also they have a price/performance ratio much more favorable than Application Specific Integrated Circuits (ASICs). Since the major resources for implementing computation-intensive algorithms are embedded on FPGA, latency associated with device communication has been eliminated. However, these embedded resources are limited hence it is important to use these resources optimally.
The last decade has seen ever increasing application areas for FPGAs. Modern FPGAs currently accommodate more than ten million gates with clock rates approaching 550 MHz [13]. Example application areas include single chip replacements for old multichip technology designs, DSP, image processing, multimedia applications, high-speed communications and networking equipment such as routers and switches, the implementation of bus protocols such as Peripheral Component Interconnect (PCI), microprocessor glue logic, coprocessors and controllers.
Most of the computation intensive algorithms such as those used in signal, image and video processing, numerical analysis, computer graphics and vision involve matrix operation as the kernel operation. In this paper, different architectures of matrix multiplication for use in image and signal processing applications are considered for hardware realization using FPGA.
The paper is organized as follows. Section 2 presents a brief overview of FPGA technology and its comparison with other technologies is presented in section 3. The design methodology adopted in this work is presented in section 4. A brief literature review on the use of FPGAs for hardware implementation of matrix multiplication is presented in section 5. Section 6 and 7 presents FPGA design and implementation of different matrix multiplier architectures. Finally, concluding remarks are discussed in section 8.

## 2. FPGA Overview

Programmable devices, such as programmable logic arrays (PLAs), have been available since 1970s. However, for a number of years, their use was quite limited, mainly due to

Manuscript received February 5, 2010
Manuscript revised February 20, 2010

technological reasons. In the early 1980s, programmable array logic (PALs) devices started to be used as glue-logic parts but suffered from power consumption problems. The extension of the gate array technique to post manufacturing customization, based on the idea of using arrays of custom logic blocks (LBs) that are surrounded by a perimeter of input/output (I/O) blocks, all of which could be assembled arbitrarily [14-15], gave rise to the FPGA concept, which was introduced by Xilinx' cofounder Ross Freeman in 1985.

FPGAs are digital integrated circuits (ICs) that belong to a family of programmable logic devices (PLDs). An FPGA chip includes I/O blocks and the core programmable fabric. The I/O blocks are located around the periphery of the chip, providing programmable I/O connections and support for various I/O standards. The core programmable fabric consists of programmable logic blocks also called configurable logic blocks (CLBs) and programmable routing architectures. By using the appropriate configuration, FPGAs can, in principle, implement any digital circuit as long as their available resources are adequate. Fig. 1 illustrates a general FPGA fabric [16], which represents a popular architecture that many commercial FPGAs are based on, and is also a widely accepted architecture model used by FPGA researchers.

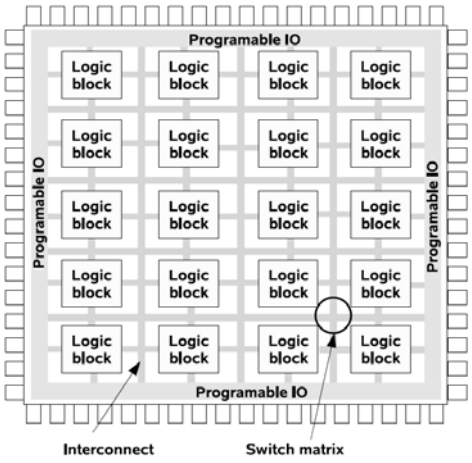


Fig. 1 General FPGA fabric.

FPGAs can be programmed after it is manufactured rather being limited to a predetermined, unchangeable hardware function. The term "field programmable" refers to the fact that its programming takes place "in the field" as opposed to devices whose internal functionality is hardwired by the manufacturer [17-18]. Many different architecture and programming technologies have evolved to provide better designs that make FPGAs economically viable and an attractive alternative to ASICs. Modern FPGAs have superior logic density, low chip cost and performance specifications comparable to low end microprocessor. With multimillion programmable gates per chip, current FPGAs can be used to implement digital systems capable of operating at frequencies up to 550 MHz. In many cases, it is possible to implement an entire system using a single FPGA. This is very economical for specialized applications that do not require the performance of custom hardware.

Significant technological advancements have led to architectures that combine FPGA's logic blocks and interconnect matrices, with one or more microprocessors, embedded Intellectual Property (IP) cores, memory blocks, DSP blocks integrated on a single chip to facilitate the implementation of Programmable System-on-a-Chip (PSoC) designs [19-20].

Examples of PSoC are the Xilinx Virtex-II Pro, Virtex-4 and Virtex-5 FPGA families, which include one or more hard-core PowerPC processors embedded along with the FPGA's logic fabric [21-23]. Alternatively, soft processor cores that are implemented using part of the FPGA logic fabric are also available. Many soft processor cores are now available such as: Xilinx 32-bit MicroBlaze [24] and PicoBlaze, and the Altera Nios and the 32-bit Nios II processor [14].

## 3. Comparison of FPGAs with ASICs, GPPs and DSPs

An ASIC is highly optimized for one specific application or product. ASICs can provide the best performance and lowest power consumption. For large volume applications, ASICs can also provide the lowest chip cost and system cost. Despite the advantages of ASICs, they are often infeasible or uneconomical for many embedded systems because of high nonrecurring engineering (NRE) cost and longer design time [13-14, 25]. As compared to ASICs, FPGAs offer many advantages such as reduced NRE cost and shorter time to market. However, relatively high size and power consumption shown by FPGA devices has been the most important drawback of that technology.

GPPs on the other hand are microprocessors that are designed to perform a wide range of computing tasks. As mentioned earlier, FPGAs are most often contrasted with ASICs. However, before deciding on the implementation technology, it is very important to study the application carefully and then determine if it is possible to meet performance requirements with existing programmable processors-GPPs or DSPs. Development of code for such processors require much less effort as compared to that required for FPGAs or ASICs, because developing software with sequential languages such as C or C++ is much less challenging than writing parallel code with Hardware Description Languages (HDLs).

GPPs are also generally cheaper than FPGAs. Hence, if a GPP can meet application requirements (performance, power, etc.), it is almost always the best choice. In general, FPGAs are well suited to applications that demand extremely high performance and reprogrammability.

DSPs are also microprocessors that are specifically optimized for the efficient execution of common signal processing tasks. DSPs are not as specialized as ASICs, so they are usually not as efficient in terms of speed, power consumption and price. DSPs are characterized by their flexibility and ease of programming relative to the FPGA. In a DSP system, the programmer does not need to understand the hardware architecture [26]; the hardware implementation is hidden from the user. The DSP programmer uses either C or assembly language.

With respect to the performance criterion, the speed is limited by the clock speed of the DSPs, given that the DSPs operate in a sequential manner and accordingly cannot be fully parallelized. FPGAs, on the other hand, can work very fast if an appropriate parallelized architecture is designed. Reconfigurability in DSPs can be achieved by changing the memory content of its program. This is in contrast to FPGAs where reconfigurability can be performed by downloading reconfiguration data to the RAM.

Power consumption in a DSP depends on the number of memory elements used regardless of the size of the executable program. For FPGA, the power consumption depends on the circuit design. FPGAs are important when there is a need to implement a parallel algorithm, that is, when different components operate in parallel to implement the system functionality. Thus the speed of execution is independent of the number of modules. This is in contrast to DSP systems where the execution speed is inversely proportional to the number of functionalities. FPGAs deliver an order of magnitude higher performance than DSPs [27].

## 4. Design Methodology

Design methodology for the hardware realization of computation intensive algorithm is a combined effort of Electronic Design Automation (EDA) tools, methods and FPGA technology that enables to produce the optimized circuit for the end applications. A right combination of FPGA hardware, designed IP core and EDA tools will definitely enhance the efficiency of the design methodology.

By design methodology, we imply the step-by-step process of FPGA design. The FPGA design methodology is used as a guideline for the hardware realization of algorithms. A number of design flows are used by different FPGA vendors but all are basically similar in sequence of tasks performed. These steps are common in all FPGA EDA tools and are essential in today's FPGA design process. The EDA tools like Xilinx Integrated Software Environment (ISE), Altera's Quartus II and Mentor Graphics' FPGA Advantage plays a very important role in obtaining an optimized digital circuit using FPGA [13-14]. A typical FPGA design flow followed in this work is shown in fig. 2.

In this flow, design Entry is used to describe the algorithm/circuit that has to be implemented onto the FPGA device. There are two standard approaches to specify the FPGA designs: HDL-based and Schematic-based depending upon the complexity of FPGA design. However, for complex and computationally intensive algorithms HDL-based (VHDL or Verilog) design entry is the dominant method used by FPGA designers. After specifying the design using HDLs or Schematic, the designer needs to validate the logical correctness of the design. This is performed using functional or behavioral simulation.

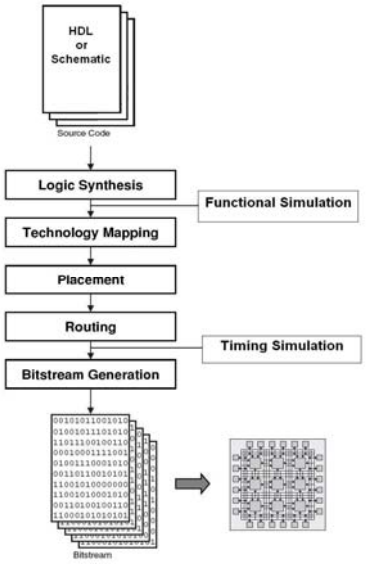


Fig. 2 FPGA design flow.

Designers usually go through this step right after they finish the design entry and logic synthesis. Logic synthesis converts HDL or schematic-based design entry into a netlist of actual gates/blocks specified in FPGA devices. This is the most important step of the whole design process. Technology mapping is a step in the middle of typical FPGA design flow. In this step, the EDA tool transforms a netlist of technology independent logic gates into one comprised of logic cells and IOBs in the target FPGA architectures [28]. Mapping plays a significant role on the quality of the implemented circuits [29].

Placement follows technology mapping in an FPGA design flow and selects the optimal position for each block

in a circuit. A good placement is extremely important for FPGA designs. It directly affects the routability and the performance of a design on FPGA [30]. FPGA placement algorithms can be broadly classified as routability-driven and timing-driven [31]. The next step in the FPGA design flow is routing [32-34]. It is the last step in the design flow prior to generating the bit-stream to program the FPGA. FPGA routing is a tedious process because it has to use only the prefabricated routing resources such as wire segments, programmable switches and multiplexers [33].

Analysis is essential for today's designs that have complex algorithms and huge amount of gates. The analysis tools (ModelSim, ISE simulator, Quartus II) are linked with the initial step and when an error occurs, the whole design has to go back to previous steps or in certain situations to the very beginning depending on the severity of the problem. Timing simulation validates the logical correctness of the design taking into account the delays of the FPGA device. Bit stream generation and downloading the generated bit file in the FPGA is the final step of the FPGA design flow.

## 5. Literature Review

Matrix multiplication is a computationally intensive problem, especially the design and efficient implementation on an FPGA where resources are very limited, has been more demanding. FPGA based designs are usually evaluated using three performance metrics: speed (latency), area, and power (energy). Fixed point implementations in FPGA are fast and have minimal power consumption. Additionally, a fixed point matrix multiplier unit often requires less silicon real estate in an FPGA or ASIC than its floating-point counterpart. The limitation of fixed point number is that very large and very small numbers cannot be represented and the range is limited to bit-width of the number. There has been extensive previous work in the area of designing an FPGA based system for the computation of fixed point matrix multiplier.

In [35], a design methodology for synthesizing a family of very compact systolic arrays on FPGA based essentially upon manual mapping at CLB level coupled with VHDL structural-level is discussed. The authors of [36] used matrix multiplication as the benchmark to compare the performance of FPGAs, DSPs and embedded processors. The results show that the FPGAs can multiply two matrices with both lower latency and lower energy consumption than the other two types of devices. This makes FPGA ideal choice for matrix multiplication in signal processing applications.

Amira *et al*. presented a novel architecture based on systolic architecture for a matrix multiplication [37]. A serial-parallel matrix multiplier based on the Baugh-Wooley algorithm has been used. The design based on the systolic architecture has been implemented using a Xilinx XCV1000E of Virtex-E FPGA family.

Amira *et al*. designed a parameterizable system for 8-bit fixed point matrix multiplication using FPGA [38]. Their design used both systolic architecture and distributed arithmetic design methodology for the implementation of matrix multiplication. The architecture proposed in this paper was targeted to Xilinx XCV2000E of Virtex-E FPGA family. The results presented in this paper showed better performance than the architecture presented in [37] in terms of area and speed. For n=4, distributed arithmetic based design used 57 Slices as compared to 72 slices used in [37] and operated at a maximum frequency of 166.47 MHz as compared to 58.302 MHz used in [37].

Distributed Arithmetic based design provides better performance in terms of speed and area as compared to systolic array based design. The I/O bandwidth required by the design is directly proportional to the problem size. The designs presented in [37-38] were restricted to small matrix size. For multiplying large matrices (n=128, 256 and 512), Bensaali *et al*. designed an FPGA based coprocessor [39]. The designed coprocessor first partitions the input matrices into smaller sub-matrices and then calculates the product.

In [40], Mencer *et al*. implemented the matrix multiplication on Xilinx XC4000E FPGA device. Their design employs bit serial multipliers using Booth encoding. They focused on tradeoffs between area and maximum running frequency with parameterized circuit generators. Their design was improved by Amira *et al*. in [41-42] using modified booth encoder multiplication along with Wallace tree addition. For n=4, 296 CLBs were used to achieve a maximum operating frequency of 60 MHz using Xilinx XCV1000E FPGA.

Jang *et al*. improved the design in [40-42] in terms of area, speed [43] and energy [44] by taking advantage of data reuse. They reduced the latency for computing matrix product by employing internal storage registers in the processing element (PE). The algorithms need n multipliers, n adders, and total storage of size $n^2$ words. For $4 \times 4$ matrix multiplication, the latency of the design in [40] is 0.57μs, while the design in [43-44] uses 0.15μs utilizing 18 % less area as compared to [40].

Belkacemi *et al*. [45] presented the design and implementation of a high performance, fully parallel matrix multiplication core. The core was parameterized and scalable in terms of the matrix dimensions (i.e., number of rows and columns) and the input data word length. Fully floor planned FPGA configurations were generated automatically, from high-level descriptions of the matrix multiplication operation, in the form of Electronic Design Interchange Format (EDIF) netlist in less than one second. These are specifically optimized for Xilinx Virtex FPGA chips. By exploiting the abundance of

logic resources in Xilinx Virtex FPGAs (LUTs, fast carry logic, shift registers, flip flops etc.), a fully parallel implementation of the matrix multiplier core is achieved; with a full matrix result being generated every clock cycle. A 3 × 3 matrix multiplier instance consumes 2,448 Virtex slices and can run at 175 MHz on an XCV1000E-6 Virtex-E chip.

Traditionally, the performance metrics for FPGA based designs have been latency and area. However, with the proliferation of portable, mobile devices, it has become increasingly important that the systems are also energy efficient and consume low power. In FPGA devices, major chunk of power is consumed by the programmable interconnects, while the remaining power is consumed by the clocking, logic and I/O blocks. Another source of power dissipation in FPGAs is resource utilization and switching activity [46]. Research efforts towards the design of energy efficient matrix multiplier have been reported in [44], [47-49].

Most of the previous work in fixed point matrix multiplication focused only on reducing the latency and the area. Choi *et al*. developed new designs and architectures for FPGAs which minimize the power consumption along with latency and area [47-48]. They used linear systolic architecture to develop energy efficient designs. For linear array architecture, the amount of storage per processing element affects the system wide energy. Thus, they used maximum amount of storage per processing element and minimum number of multipliers to obtain energy-efficient matrix multiplier.

Partially reconfigurability feature was exploited for the first time for the computation of matrix multiplication by Jianwen *et al*. in [50]. Partially reconfigurable devices offer the possibility of changing the design implementation without stopping the whole execution process. The matrix multiplier was implemented in Xilinx Virtex-II device, which supports partial reconfiguration. The design was evaluated in terms of latency and area and it was found that area is reduced by 72%-81% for matrix sizes between 3 × 3 and 48 × 48 as compared to [43] and the performance further improves for larger matrices.

## 6. Matrix-Vector Multiplication: Design and Implementation

In this section, we present the design and discuss the results of implementing matrix-vector multiplication which is computationally very intensive. It requires several multiply and add units. In DSPs, the overall performance is limited by the number of multiplications and additions that could be done in parallel. DSPs take several clock cycles to perform all the necessary multiply-add operations. However, modern FPGAs on the other hand has large number of hardware resources embedded in the FPGA fabric itself such as DSP48 blocks, multipliers, Block RAMs, etc. It can provide higher and more efficient processing rates required by such applications if the algorithm is coded in a way to utilize these embedded resources efficiently. The objective of this paper is to realize a large matrix-vector multiplier for image processing applications [51]. To achieve this, FPGA is used for faster and efficient realization.

### 6.1 Mathematical Formulation

We represent the vector C as $(C_1, C_2...C_m)^T$ and vector G which represents the image data. According to the application, we want to multiply matrix S with vector C represented by the following equation

$$C=SG \tag{1}$$

where, S is a Jacobian matrix. In the discrete form, it is required to find the unknown vector G from the known vector C, while S is treated as a constant matrix for simplicity. We can represent G by the following relationship

$$G=S^TC \tag{2}$$

where, $S^T$ is the transpose of S. Replacing $S^T$ by A, mathematically; the above equation is approximated by the following relationship

$$G=AC \tag{3}$$

The key idea here is to calculate G using (3). The dimension of the given matrices depends on the application, which, in this case is summarized in table 1.

Table 1: Matrix Dimensions

| ***Matrix Symbol*** | ***Matrix Dimension*** |
|---|---|
| A | 1024×28 |
| C | 28×1 |
| G | 1024×1 |

### 6.2 Hardware Architecture

This section presents the technique to design hardware architecture for implementing matrix-vector multiplication algorithm on FPGA. As can be seen from (3), matrix-vector multiplication is the kernel operation. For efficient implementation and maximum speed-up, integer arithmetic is used. Since the floating-point arithmetic unit consumes more silicon real estate of FPGA and are slower as compared to integer arithmetic, we used integer arithmetic for our designs.

The design involves the computation of G = AC, where A is a matrix, C and G are vectors as summarized in table 1. We need to calculate vector G. Broadcast algorithm is adopted for the matrix-vector multiplication. The matrix–vector multiplication is performed by broadcasting rows of matrix A and multiplying the corresponding column elements of vector C. Following operations are involved:

- Reading individual row elements of matrix A and individual column elements of vector C
- Storing them in internal buffers row and column wise respectively
- Multiplying row and column elements
- Accumulating the multiplier output and writing back the results to the output buffers.

The input and output buffers are implemented on the FPGA. The matrix-vector multiplications involve multiply and accumulate operations. The multiply-accumulate unit consists of a multiplier and adder. The row and the column elements are supplied as the two inputs to the multiplier. The output of the multiplier is directly given to the adder as one of the inputs. The previous output of the adder is fed back as the second input to the adder.

The multiply-accumulate unit takes each element of the matrix A in row major format and each element of vector C, multiplies them and adds the result to the running total. This process is repeated till the last element of row A and column C. The values are fed in a sequential manner. If the reset signal is asserted high, the contents of registers A and C are cleared. After a delay, as determined by the implementation results, the first element of vector G is available at the serial output and this output is stored in on-chip memory. This operation is repeated and the process continues until all the rows of matrix A are processed. Finally, the output vector G is available with all the elements stored in the memory locations. A simplified block diagram of matrix-vector multiplication is shown in fig. 3.

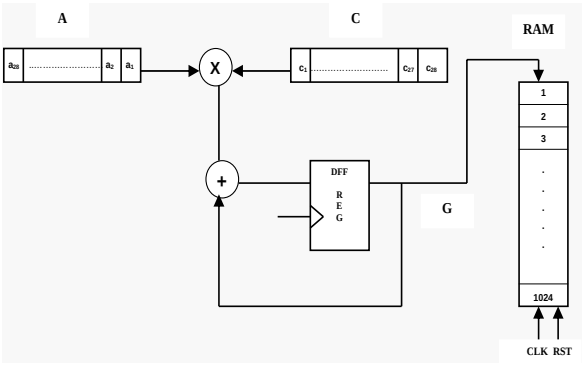


Fig. 3 Block diagram of matrix-vector multiplication

### 6.3 Implementation

In order to evaluate the performance of our FPGA-based implementation, the algorithm was coded in VHDL and implemented on Xilinx Virtex-4 (XC4VLX200FF1513, speed grade: -11) family using Xilinx ISE 9.2i tool. The design was synthesized into Virtex-4 FPGA optimized for speed. The hardware resource utilization is summarized in table 2.

Table 2: FPGA Resource Utilization

| *Resource* | *Used/Available* | *Utilization* |
|---|---|---|
| Slices | 1,3010 out of 89,088 | 14% |
| 4-input LUTs | 9,612 out of 178,176 | 5% |
| DSP48s | 55 out of 96 | 57% |
| Max. Frequency | 17.376 (MHz) | - |

As shown in table 2, roughly 14% of the slices and 57% DSP48s are utilized leaving a plenty of room to implement more parallel processors on the same FPGA chip. The results listed in table 2 were obtained using Xilinx ISE 9.2i tool configured to optimize for speed. The total processing time using Virtex-4 FPGA is found to be 58.93 μs; this is equivalent to a throughput of 16970 frames per second. The results indicate the feasibility of using FPGA for real time high speed image processing applications using this matrix-vector multiplication.

## 7. Tri-Matrix Multiplication: Design and Implementation

In this section we will present the design of tri-matrix multiplier which is commonly used in DSP applications [52]. Matrix T can be written as

$$T = XYZ \tag{4}$$

where, X and Z are rectangular matrices given by (5) and (6) respectively. Y is a diagonal square matrix, where $n$ = 0, 1, …, N-1.

$$X = \begin{bmatrix} 0 & 0 & \cdots & 0 & x_1(0) & \cdots & x_1(N-k-1) \\ 0 & \cdots & \cdots & x_1(0) & x_1(1) & \cdots & x_1(N-k) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & x_1(0) & \cdots & \cdots & \cdots & \cdots & x_1(N-2) \\ x_1(0) & x_1(1) & \cdots & \cdots & \cdots & \cdots & x_1(N-1) \\ x_1(1) & x_1(2) & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1(k) & x_1(k+1) & \cdots & x_1(2k-1) & x_1(2k) & \cdots & 0 \end{bmatrix} \tag{5}$$

$$Z = \begin{bmatrix} 0 & 0 & \cdots & x_2(0) & x_2(1) & \cdots & x_2(k) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & x_2(0) & \cdots & \cdots & \cdots & \cdots & x_2(2k-1) \\ x_2(0) & x_2(1) & \cdots & \cdots & \cdots & \cdots & x_2(2k) \\ x_2(1) & x_2(2) & \cdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_2(N-k-1) & x_2(N-k) & \cdots & x_2(N-1) & 0 & \cdots & 0 \end{bmatrix} \quad (6)$$

## 7.1 Hardware Architecture

The system for the above given mathematical formulation translates into two blocks, in which the first block multiplies matrix X by diagonal matrix Y and then serves the output from this block to another block, which multiplies the product XY by Z. We used the two-dimensional systolic array based architecture as shown in fig. 4 and fig. 5 for the matrix multiplication.

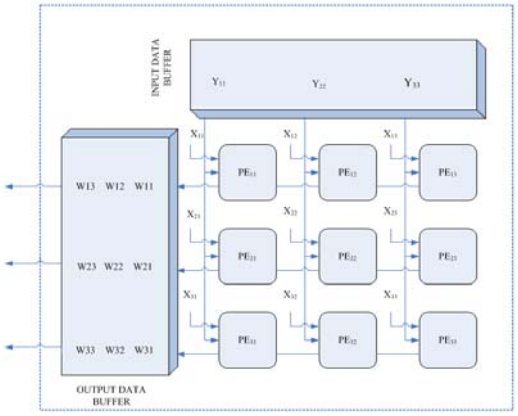


Fig. 4 Architecture of first block

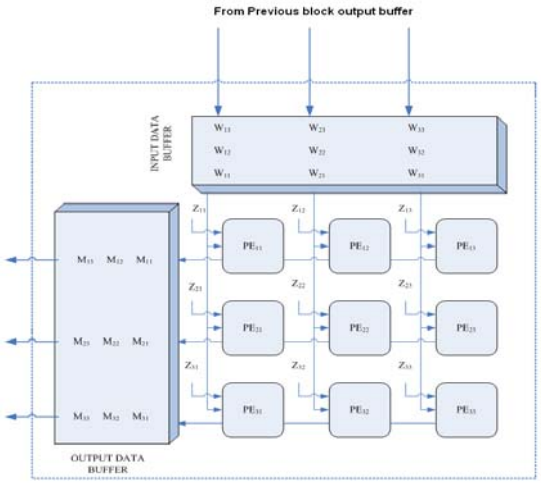


Fig. 5 Architecture of second block

Systolic arrays accelerate medium sized matrix multiplication by exploiting the inherent data parallelism in matrix multiplication. Multiplying the matrix X by the diagonal square matrix Y is equivalent to multiplying the first diagonal element by the entries of first row of X, the second diagonal element by the entries of the second row of X and so on.

Fig. 4 and fig. 5 shows the systolic architecture for both the modules for $N_1$=3 and $N_2$=3 respectively. Both the matrix multiplier blocks consist of nine identical processing elements, $PE_1$ and $PE_2$, respectively. $PE_1$ consists of multiplier whereas $PE_2$ consists of MAC unit where each MAC unit consists of a multiplier and adder.

The function of each $PE_1$ in the first array is to multiply the diagonal elements of Y by one element of matrix X during each clock period. First column $PE_1$ are responsible for producing first column of the product XY referred to as W in the fig. 4, second column generates the second column and so on. The entries are stored in an internal buffer to be used later by the next array. Similarly, the second array as shown in fig. 5 performs the multiplication of (XY) with Z.

## 7.2 Implementation

FPGA-based systolic array parallel architecture for the tri-matrix multiplication was evaluated for different matrix sizes ranging from 3×3 to 7×7 tri-matrix multiplications. The same architecture is extended for 7×7 tri-matrix multiplier. For 7×7 tri-matrix multiplier, the FPGA utilizes more resources as compared to 3×3 tri-matrix multiplier. It requires more hardware resources which is obvious from the computational complexity of 7×7 multiplier. The implementation results are summarized and compared in table 3. Since the 7×7 design could not be fit into Spartan-3 (XC3S2000FG900-4) device, we used a more advanced Virtex-II Pro (XC2VP100FF1704-6) platform FPGA for implementation.

Table 3: FPGA Resource Utilization Comparison

| ***FPGA Resources*** | ***T=[ ]7×7 (Virtex-II Pro)*** | ***T=[ ]3×3 (Spartan-3)*** |
|---|---|---|
| LUTs | 2,353 | 270 |
| CLB Slices | 1,177 | 144 |
| Eq. Gate Count | 1,151,761 | 148,215 |
| Max. Frequency | 102 MHz | 87 MHz |
| Embedded Multipliers | 280 | 36 |

## 8. Conclusions

Most of the algorithms which are used in DSP, image and video processing, computer graphics and vision and high

performance supercomputing applications have matrix multiplication as the kernel operation. In this paper, we considered two different examples of matrix multiplier architecture where speed is the main constraint. The first design involving computation of dense matrix-vector multiplication is implemented on Xilinx Virtex-4 FPGA and the performance is evaluated by computing its execution time on FPGA. Hardware implementation results demonstrate that it can provide a throughput of 16970 frames per second which is sufficient for many image and video processing applications. The second design for the multiplication of three matrices is based on systolic array and implemented on Spartan-3 and Virtex-II Pro platform FPGAs respectively. Implementation results demonstrate the suitability of FPGAs in such applications. Finally, we conclude that for multiplication of large matrices, memory based architecture is quite efficient whereas, for small and medium sized matrix multiplication, systolic array techniques prove to be quite efficient as demonstrated by the implementation results.

**Acknowledgment**

The authors gratefully acknowledge the financial support provided by the Research Center in the College of Engineering, King Saud University under research grant no. 11/430.

## References

[1] S. Ogrenci, A. K. Katsaggelos, and M. Sarrafzadeh, "Analysis and FPGA Implementation of Image restoration under resource constraint," IEEE Trans. on Computers, Vol. 52, No. 3, pp. 390-399, 2003.

[2] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, "Implementing an OFDM Receiver on the RaPiD Reconfigurable Architecture," IEEE Trans. on Computers, Vol. 53, No. 11, pp. 1436-1448, 2004.

[3] G. R. Goslin, "A Guide to Using Field Programmable Gate Arrays for Application-Specific Digital Signal Processing Performance," Microelectronics Journal, Vol. 28, Issue 4, pp. 24-35, 1997.

[4] J. Isoaho, J. Pasanen, O. Vainio, and H. Tenhunen, "DSP System Integration and Prototyping with FPGAs," Journal of VLSI Signal Processing, Vol. 6, pp. 155-172, 1993.

[5] A. G. Ye and D. M. Lewis, "Procedural Texture Mapping on FPGAs," in Proc. of ACM/SIGDA 7th Intl. Symp. on Field Programmable Gate Arrays, pp. 112-120, 1999.

[6] S. Knapp, "Using Programmable Logic to Accelerate DSP Functions," http://www.xilinx.com/appnotes/dspintro.pdf.

[7] J. Ma, "Signal and Image processing via Reconfigurable Computing," in Proc. of the First Workshop on Information and Systems Technology, 2003.

[8] F. Otto and Z. Pavel, "Hardware Accelerated Imaging Algorithms," in Proc. of AUTOS'2002 Automatizace systému, pp. 165-171, 2002.

[9] L. Batina, S. B. Ors, B. Preneel, and J. Vandewalle, "Hardware architectures for public key cryptography," Integration, the VLSI Journal, Vol. 34, pp. 1-64, 2003.

[10] D. Johnson, K. Gribbon, D. Bailey, and S. Demidenko, "Implementing Digital Signal Processing Algorithm's in FPGA's: Digital Spectral Warping," in Proc. of 9th Electronics New Zealand Conf., pp. 72-77, 2002.

[11] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," ACM Computing Surveys, Vol. 34, No. 2, pp.171-210, 2002.

[12] R. Tessier and W. Burleson, "Reconfigurable Computing for Digital Signal Processing: A survey," Journal of VLSI Signal Processing, Vol. 28, No. 3, pp.7-27, 2001.

[13] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable Computing: architectures and design methods," IEE Proc. of Computer Digital Techniques, Vol. 152, No. 2, pp. 193-207, 2005.

[14] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, Design Tools and Application Domains of FPGAs," IEEE Trans. on Industrial Electronics, Vol. 54, No. 4, pp. 1810-1823, 2007.

[15] Xilinx Staff, "Celebrating 20 years of innovation," Xcell Journal, No. 48, Spring 2004.

[16] V. Betz, J. Rose, and A. Marquardt, Architecture and CAD for Deep-Submicron FPGAs, Kluwer Academic Publishers, USA, 1999.

[17] D. Pellerin and S. Thibault, Practical FPGA programming in C, Prentice Hall, New York, USA, First Edition, 2005.

[18] C. M. Maxfield, The Design Warrior's Guide to FPGAs, Elsevier Publishers, New York, USA, First Edition, 2004.

[19] G. Stitt and F. Vahid, "Energy advantages of microprocessor platforms with on-chip configurable logic," IEEE Design and Test of Computers, Vol. 19, No. 6, pp. 36-43, 2002.

[20] A. Ansari, P. Ryser, and D. Isaacs, "Accelerated System Performance with APU-enhanced processing," Xcell Journal, First quarter 2005.

[21] Xilinx Inc, Virtex-II platform FPGA Data Sheet, 2005.

[22] Xilinx Inc, Virtex-4 multiplatform FPGA, 2005.

[23] Xilinx Inc, Virtex-5 multiplatform FPGA, May 2006.

[24] Xilinx Inc, MicroBlaze Soft Processor Core, 2005.

[25] I. Kuon and J. Rose, "Measuring the Gap between FPGAs and ASICs," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 26, No. 2, pp. 203-215, 2007.

[26] M. Cummings and S. Haruyama, "FPGA in the Software Radio," IEEE Communication Magazine, Vol. 37, pp. 108-112, 1999.

[27] B. Tithecott, "Why FPGAs are quickly moving into embedded signal processing systems," 2004. www.dsp-fpga.com/pdfs/SBS.Sum04.pdf

[28] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, Vol. 25, No. 11, pp. 2331-2340, 2006.

[29] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "Improvements to Technology Mapping for LUT-Based FPGAs," IEEE Trans. on Computer Aided Design of

Integrated Circuits and Systems , Vol. 26, No. 2, pp. 240-253, 2007.
[30] W. K. Mak and L. Hao, "Placement for modern FPGAs," in Proc. of Emerging Information Technology Conference, pp. 1-4, 2005.
[31] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in Proc. of the ACM/SIGDA Intl. Symp. on Field Programmable Gate Arrays, pp. 203-213, 2000.
[32] M. J. Alexander and G. Robins, "New-performance driven FPGA routing algorithms," IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, Vol. 15, No. 12, pp. 1505-1517, 1996.
[33] G. J. Nam, F. Aloul, K. A. Sakallah, and R. A. Rutenbar, "A Comparative study of two boolean formulations of FPGA detailed routing constraints," IEEE Trans. on Computers, Vol. 53, No. 6, pp. 688-696, 2004.
[34] http://www.tutorial-reports.com/computer-science/fpga/routing.php
[35] A. Oudjida, S. Titri, and M. Hamerlain, "Synthesizing Full-Systolic Arrays for Matrix Product on Xilinx's XC4000 (E, EX) FPGAs," in Proc. of the ACM/SIGDA Intl. Symp. on Field Programmable Gate Arrays, pp.222-222, 2000.
[36] R. Scrofano, S. Choi, and V. K. Prasanna, "Energy Efficiency of FPGAs and Programmable Processors for Matrix Multiplication," in Proc. of IEEE Intl. Conf. on Field Programmable Technology, pp. 422-425, 2002.
[37] A. Amira, A. Bouridane, P. Milligan, and P. Sage, "A High Throughput FPGA Implementation of a Bit-Level Matrix Product," in Proc. of IEEE Workshop on Signal Processing Systems, pp. 356-364, 2000.
[38] A. Amira and F. Bensaali, "An FPGA based parameterizable system for matrix product implementation," in Proc. of IEEE Workshop on Signal Processing Systems, pp. 75-79, 2002.
[39] F. Bensaali, A. Amira, and A. Bouridane, "An FPGA based coprocessor for large matrix product implementation," in Proc. of IEEE Intl. Conf. on Field Programmable Technology, pp. 292-295, 2003.
[40] O. Mencer, M. Morf, and M. J. Flynn, "PAM-Blox: High performance FPGA design for adaptive computing," in Proc. of IEEE Symp. on FPGAs for Custom Computing Machines, pp. 167-174, 1998.
[41] A. Amira, A. Bouridane, and P. Milligan, "Accelerating Matrix Product on Reconfigurable Hardware for Signal Processing," in Proc. of 11th Intl. Conf. on Field Programmable Logic and Applications, pp. 101-111, 2001.
[42] F. Bensaali, A. Amira, and A. Bouridane, "Accelerating matrix product on reconfigurable hardware for image processing applications," IEE Proc. of Circuits, Devices and Systems, Vol. 152, No. 3, pp. 236-246, 2005.
[43] J. Jang, S. Choi, and V. K. Prasanna, "Area and Time Efficient Implementations of Matrix Multiplication on FPGAs," in Proc. of IEEE Intl. Conf. on Field Programmable Technology, pp. 93-100, 2002.
[44] J. Jang, S. Choi, and V. K. Prasanna, "Energy and Time Efficient Matrix Multiplication on FPGAs," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 11, pp. 1305-1319, 2005.
[45] S. Belkacemi, K. Benkrid, D. Crookes, and A. Benkrid, "Design and implementation of a high performance matrix multiplier core for Xilinx Virtex FPGA," in Proc. of IEEE Intl. Workshop on Computer Architectures for Machine Perception, pp. 156-159, 2003.
[46] L. Shang, A. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex-II FPGA family," in Proc. of ACM/SIGDA 10th Intl. Symp. on Field Programmable Gate Arrays, pp. 157-164, 2002.
[47] J. Jang, S. Choi, and V. K. Prasanna, "Energy efficient matrix multiplication on FPGAs," in Proc. of 12th Intl. Conf. on Field Programmable Logic and Applications, pp. 534-544, 2002.
[48] S. Choi, V. K. Prasanna, and J. Jang, "Minimizing energy dissipation of matrix multiplication kernel on Virtex-II," in Proc. of SPIE, Vol. 4867, pp. 98-106, 2002.
[49] S. Choi, R. Scrofano, V. K. Prasanna, and J. Jang, "Energy efficient signal processing using FPGAs," in Proc. of ACM/SIGDA 11th Intl. Symp. on Field Programmable Gate Arrays, pp. 225-234, 2003.
[50] L. Jianwen and J. C. Chuen, "Partially Reconfigurable Matrix Multiplication for Area and Time Efficiency on FPGAs," in Proc. of Euromicro Symp. on Digital System Design , pp. 244-248, 2004.
[51] B. Almashary, S. M. Qasim, S. A. Alshebeili, and W. Almasry, "Realization of Linear Back-Projection Algorithm for Capacitance Tomography Using FPGA", in Proc. of 4th World Congress on Industrial Process Tomography, pp. 87-93, 2005.
[52] S. A. Alshebeili, "Computation of higher-order cross moments based on matrix multiplication", Journal of the Franklin Institute, 338, pp. 811-816, 2001.

**Syed M. Qasim** received the B.Tech and M.Tech Degrees in Electronics Engineering from Z. H. College of Engineering and Technology, Aligarh Muslim University, India in 2000 and 2002 respectively. He is now working as a researcher in the Electronics Group, Department of Electrical Engineering, King Saud University, Saudi Arabia. He is the author or coauthor of more than 30 papers in international journals and refereed conferences. He is a member of the Institution of Electronics and Telecommunication Engineers (IETE), India and International Association of Engineers (IAENG), Hong Kong.

**Ahmed A. Telba** received the Ph.D degree in Electronics and Communication Engineering from University of Bradford, UK in 2007. He is now working as a Post Doctoral Researcher in the Electronics Group, Department of Electrical Engineering, King Saud University, Saudi Arabia. He is a member of IEEE, USA.

**Abdulhameed Y. AlMazroo** received the B.S. and M.S. degrees in Electrical Engineering from Riyadh University, Saudi Arabia and University of Michigan, Ann Arbor, USA in 1980 and 1984, respectively. He received the Ph.D degree in Electrical Engineering from Virginia Polytechnic, Virginia, USA in 1988. He is an Assistant Professor in the Department of Electrical Engineering at King Saud University. He is a member of IEEE, USA