A GENERAL FRAMEWORK FOR EFFICIENT FPGA IMPLEMENTATION OF MATRIX PRODUCT

Faycal Bensaali[a,*], Abbes Amira[b], Reza Sotudeh[a]

[a] University of Hertfordshire

[b] Brunel University, West London

ABSTRACT

High performance systems are required by the developers for fast processing of computationally intensive applications. Reconfigurable hardware devices in the form of Filed-Programmable Gate Arrays (FPGAs) have been proposed as viable system building blocks in the construction of high performance systems at an economical price. Given the importance and the use of matrix algorithms in scientific computing applications, they seem ideal candidates to harness and exploit the advantages offered by FPGAs. In this paper, a system for matrix algorithm cores generation is described. The system provides a catalog of efficient user-customizable cores, designed for FPGA implementation, ranging in three different matrix algorithm categories: (i) matrix operations, (ii) matrix transforms and (iii) matrix decomposition. The generated core can be either a general purpose or a specific application core. The methodology used in the design and implementation of two specific image processing application cores is presented. The first core is a fully pipelined matrix multiplier for colour space conversion based on distributed arithmetic principles while the second one is a parallel floating-point matrix multiplier designed for 3D affine transformations.

Keywords

Matrix Multiplication, Field Programmable Gate Array, Distributed Arithmetic, 3D Affine Transformations.

1. INTRODUCTION

The nature of scientific computing applications involves performing complex tasks repeatedly on a large set of data, often under real-time requirements. Therefore, high performance systems are required by the developers for fast computations.

Many researchers have begun to recognise the potential of reconfigurable hardware in the form of Filed-Programmable Gate Arrays (FPGA) in accelerating such computationally intensive applications. A close examination of the algorithms used in these applications reveals that many of the fundamental actions involve matrix algorithms.

Several systems have been developed for matrix algorithms implementation. The most important work done so far in this area of research is the RCAMAT system centred around the Virtex-E Xilinx FPGA series [1]. It consists of a Graphical User Interface (GUI) for generating, browsing and mapping the matrix algorithms on the RCMAT coprocessor, a library containing structural and parametrazable VHDL codes and a coprocessor which is basically the Virtex-E FPGA.

A coprocessor based on RISC architecture, has been developed at Adelaide University under the MATRISC project. A range of matrix algorithms such as Singular Value Decomposition (SVD), matrix multiplication and addition, Discrete Fourier Transform (DFT) and QR factorisation can be performed using this coprocessor [2].

Another project (MATCH compiler project) concerning the implementation of matrix algorithms is described in [3]. The objective of this project is to develop various MATLAB libraries on an FPGA board. These functions are developed in Register Transfer Level (RTL) VHDL using the Sinplify logic synthesis tool from Synplicity to generate gate level netlists, and the Alliance Place And Route (PAR) tools from Xilinx. Some of the functions which have been implemented on the FPGA board include matrix addition, matrix multiplication, one-dimensional Fast Fourier Transform (FFT) and Finite Impulse Response (FIR)/Infinite Impulse Response (IIR) filters.

A range of matrix cores have been developed so far as part of an ongoing research project for matrix operations [4, 5, 6], matrix transforms [7, 8] and matrix decompositions [9, 10]. It is the aim of the work presented in this paper to bring together the existing cores under a general environment: Matrix Core Generator (MCG) system to address a range of applications.

The MCG provides the user with a catalogue of optimised, predefined set of building blocks for common matrix algorithms, simplifying design steps and bringing the user design, based on such blocks, to completion faster.

The developed cores can be ranged into two different categories: (i) general purpose cores; and (ii) specific application cores. The methodology used in the design and implementation of two matrix multipliers for two different applications is presented in this paper.

There has been extensive work on designing and implementing matrix multiplication on FPGAs using different design methodologies and implementation approaches for different data types [11, 12, 13, 14]. Most of the proposed architectures are for general purpose used and based on the integer data type. Recent improvements in the computing power of FPGAs have motivated researchers to consider floating-point based matrix multiplication. Very few recent implementations employed floating-point arithmetic [15, 16, 17, 18].

In this paper, the first proposed matrix multiplier is a novel architecture for efficient implementation of an RGB to YCrCb colour space conversion suitable for FPGAs implementation. The proposed Colour Space Converter (CSC) is based on Distributed Arithmetic (DA) principles. It is fully pipelined, platform independent, has a low latency and a high throughput rate. In Addition, the architecture has shown outstanding results in comparison with the existing CSCs available in the market such as the cores from Amphion Ltd. [19], CAST Inc. [20] and ALMA Tech. [21] where the same conversion operation is performed.

*Corresponding author: f.bensaali@herts.ac.uk

The second proposed matrix multiplier is a parallel floating-point architecture designed for accelerating 3D affine transformations on FPGA, which are the fundamental cornerstone of computer graphics. These transformations require large amounts of processing power since they are based on matrix multiplication. The proposed core is based on a new parallelised approach for processing the 3D objects.

The proposed cores were implemented and tested on the RC1000 board [22] equipped with a XCV2000E Virtex-E (bg560-6) FPGA and in the meantime they were synthesised on other FPGA devices in order to make a fairer and consistence comparison with existing cores using the same platform or exploring the additional features of recent FPGA devices.

The composition of the rest of this paper is as follows. The description of the MCG general environment is given in Section 2. In section 3, the mathematical background, the proposed architecture and the hardware implementation for the two selected specific image processing application cores are presented. Finally, section 4 concludes the paper.

2. MATRIX CORE GENERATOR ENVIRONMENT

The MCG provides the user with a catalogue of optimized, predefined set of building blocks for common matrix algorithms, simplifying design steps and bringing the user design, based on such blocks, to completion faster.

The developed cores can be ranged into two different categories: (i) general purpose cores; and (ii) specific application cores. The methodology used in the design and implementation of three general purpose use transforms are described in the next sections.

Figure 1 illustrates the MCG general environment. It consists of three basic parts:

1. A Graphical User Interface (GUI) allowing the user to select and customize a core for generation;
2. A ready-made cores, data sheets and demos libraries; and
3. A generator to generate different input files for synthesis tools to configure the FPGA.

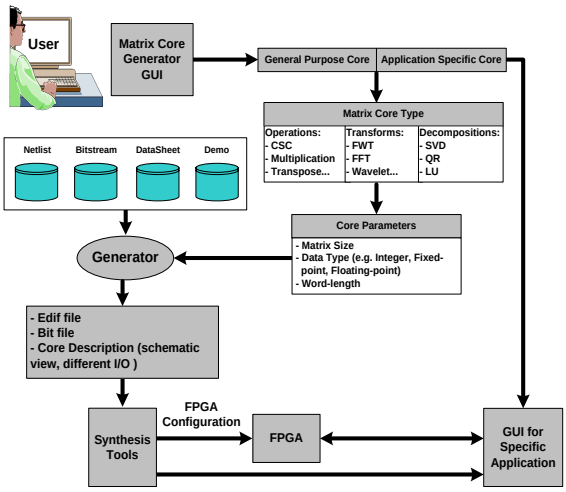

Figure 1. The MCG general environment

The GUI allows the user to select between two main options: General purpose core or application specific core. For each selected option the user has the ability to customize his design to meet the needs of his application.

For each core, the MCG system can deliver the following:

- A customized Electronic Design Interchange Format (EDIF) netlist;
- A bitstream file, to be used for FPGA configuration;
- A datasheet which includes the core's functional information; and
- A demo, if a specific application core is selected for generation.

The option of generating a bitstream file is provided if the user's design requires just the selected core. Thus, the user will be saving the time consumed during the place and route stage. In the case where the user's design consists of several cores, an EDIF file can be generated. Synthesis tools are then invoked for the general bitstream file generation by combining all the EDIF files together.

3. IMPLEMENTATION STRATEGIES OF TWO SPECIFIC APPLICATION CORES

This section describes the methodology used in the design and the implementation of two typical examples. Basically, these are matrix multiplication based DA principles for colour space conversion and floating-point parallel matrix multiplication for 3D affine transformations.

3.1 Matrix Multiplier for Colour Space Conversion

Decomposing an RGB colour image into one luminance image and two chrominance images is the method that has been used in many commercial applications such as face detection [23, 24], as well as the JPEG and MPEG imaging standards [25, 26]. The calculation of RGB colour components from YCrCb components consumes up to 40% of the processing power in a highly optimised decoder [25]. Accelerating this operation would be useful for the acceleration of the whole process. Therefore, techniques which efficiently implement this conversion are desired. This section presents a novel architecture for efficient implementation of a CSC suitable for FPGAs implementation. The proposed architecture is based on DA ROM accumulator principles.

3.1.1 Mathematical Background

A colour in the RGB/YCrCb colour space is converted to the YCrCb/RGB colour space using the following equations:

$$\begin{pmatrix} Y \\ C_r \\ C_b \end{pmatrix} = A \times \begin{pmatrix} R \\ G \\ B \\ 1 \end{pmatrix}, \begin{pmatrix} R \\ G \\ B \end{pmatrix} = B \times \begin{pmatrix} Y \\ C_r \\ C_b \\ 1 \end{pmatrix} \quad (1)$$

Where:

$$A = \begin{pmatrix} 0.257 & 0.504 & 0.098 & 16 \\ 0.439 & -0.368 & -0.071 & 128 \\ -0.148 & -0.291 & 0.439 & 128 \end{pmatrix} \quad (2)$$

$$B = \begin{pmatrix} 1.164 & 1.596 & 0.0 & -222.912 \\ 1.164 & -0.813 & -0.392 & 135.616 \\ 1.164 & 0.0 & 2.017 & -276.8 \end{pmatrix} \quad (3)$$

Figure 2 shows the direct mapping of equations 1.

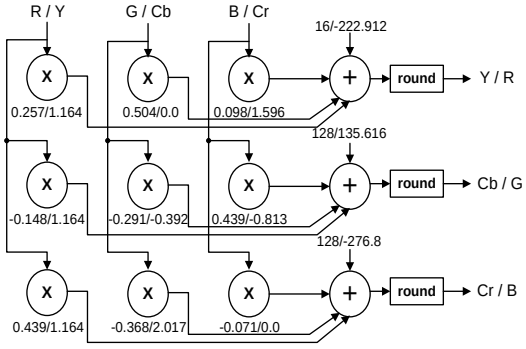

Figure 2. General block diagram for RGB ↔ YCrCb CSC

Consider an $N \times M$ image. Let represent each image pixel by b_{ijk}, $(0 \le i \le N-1, 0 \le j \le M-1, 0 \le k \le 2)$ where:

$$(b_{ij0}, b_{ij1}, b_{ij2}) = (R_{ij}, G_{ij}, B_{ij}) \quad (4)$$

where (R_{ij}, G_{ij}, B_{ij}) are respectively the red, green and blue components of the pixel in row *i* and column *j*.

Since the matrices *A* and *B* are constants, DA principles using ROM ACcumulate (RAC) technique can be applied to compute the elements of the converted image. By following the mathematical development presented in [5], the elements of the converted image c_{ijk} can be computed using the following equation:

$$c_{ijk} = \sum_{l=0}^{7} \left(\sum_{m=0}^{2} A_{km} \times b_{ijm,l} \right) \times 2^l + A_{k3} \quad (5)$$

where $b_{ijm,l}$ is the l^{th} bit of b_{ijm}.

Three ROMs (one for each matrix *A* row) with the size of $2^{N-1} = 2^3 = 8$ are needed in order to store the precompute 2^3 possible partial products values. An input set of 3 bits $(b_{ij0,l}, b_{ij1,l}, b_{ij2,l})$ is used as an address to retrieve the corresponding stored values. Table 1 gives the content of each ROM.

Table 1. Content of the ROM i $(0 \le i \le 2)$

$b_{ij0,l}$	$b_{ij1,l}$	$b_{ij2,l}$	The content of the ROM *i*
0	0	0	0
0	0	1	A_{i2}
0	1	0	A_{i1}
0	1	1	$A_{i1} + A_{i2}$
1	0	0	A_{i0}
1	0	1	$A_{i0} + A_{i2}$
1	1	0	$A_{i0} + A_{i1}$
1	1	1	$A_{i0} + A_{i1} + A_{i2}$

3.1.2 *Proposed Architecture* for CSC

Equation 5 can be mapped into the proposed architecture as shown in Figure 3. It consists of 8 identical Processor Elements (PE_ps) $(0 \le p \le 7)$. Each PE_p comprises three parallel signed integer adders, three *p* right shifters and one Memory Block, which consists of three ROMs.

It is worth noting that the architecture has a latency of 8 and a throughput rate equal to 1. The entire image conversion can be carried out in (*Latency* + (*N *M*) * *Throughput*) = 8 + (*N *M*) clock cycles, while using the standard algorithm, the conversion can be carried out in $(3 \times 4 \times N \times M)$ clock cycles, where (3×4) is the constant matrix *A* or *B* size.

3.1.3 Hardware Implementation and Results

The proposed architecture has been implemented and verified using the Celoxica RC1000 FPGA development board. The RC1000 board used is a standard PCI bus card fitted with the XCV2000E (bg560-6) Virtex-E FPGA.

The proposed architecture can be used for the inverse conversion (YCrCb to RGB) by:

- Duplicating the ROMs and using a selector signal which allows the user to choose the appropriate converter); or
- Setting the contents of the ROMs in advance, depending on the desired conversion.

The precomputed partial products are stored in the ROMs using 13 bits fixed point representation (8 bits for integer part and 5 bits for fractional part). 13-bit arithmetic is used inside the architecture. The inputs and outputs of the architecture are presented using 8 bits and the outputs are rounded. Rounding usually looks at the decimal value and if it is greater than or equal to 0.5, then the result is increased by one. This implies a condition of verifying followed by another arithmetic operation. A more efficient way to round a number is to add 0.5 to the result and truncate the decimal value. This technique has been applied in our proposed architecture. The initial value for each PE's ACC (for the serial architecture) and for the first PEs adder (for the parallel architecture) is set in advance to $(A_{i3} + 0.5)$, where $(0 \le i \le 2)$. The MACs and parallel signed adders have been implemented using Xilinxs CoreGen utility, which contains many efficient designs that can often save time for a programmer [27]. The shifters and ROMs initialization have been implemented using VHDL. All design components have been connected together using Handel-C [28].

The CSC implementation consumes 186 of the total available slices on the XCV2000E chip and runs with a maximum frequency of 277 MHz. In order to make a fairer and consistent comparison with some existing FPGA based CSCs using the same technology [19, 20, 21], the XCV50E-8 FPGA device has been targeted. A comparison with other FPGA platforms utilizing the new available features has been also performed [29, 30]. Table 2 illustrates the performances obtained for the proposed architecture in terms of area consumed and speed which can be achieved.

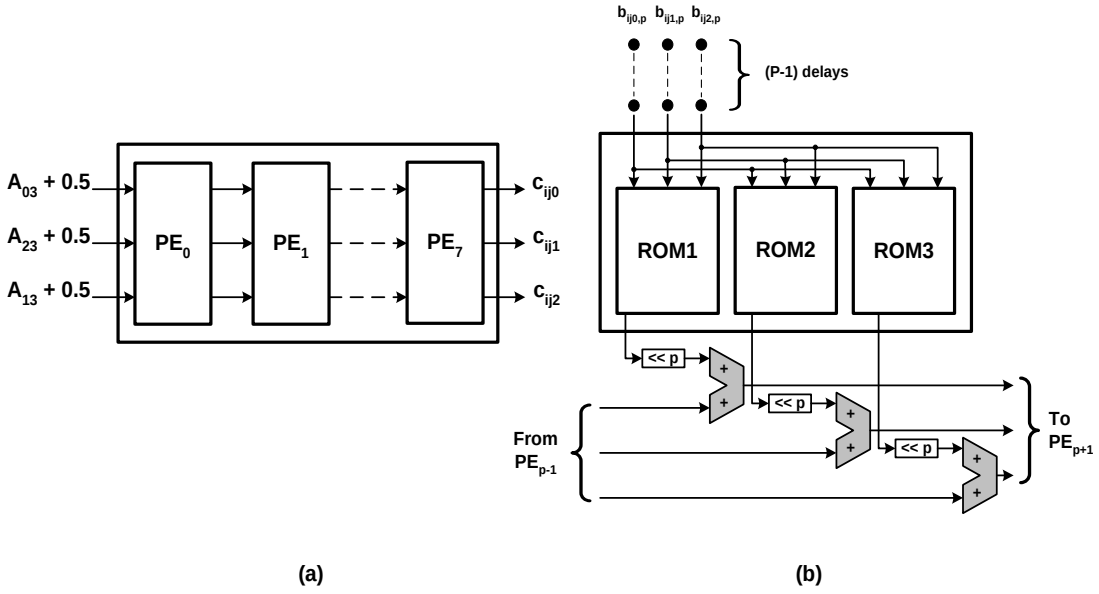

Figure 3. Proposed CSC architecture based on DA principles: (a) Overall structure, (b) Internal structure of the PE

Table 2. Performance comparison with existing CSC

CSC Core	Hardware Platform	Resources	Speed (MHz)
Proposed Architecture	XCV50E-8	186 (Slices)	263
Xilinx [29]	XC2V500-5	131 LUTs + 5 [(18×18) MULTs]	185
Architecture in [30]	Cyclon-II	292 (Logic Elements)	216
CAST. Inc [20]	XCV50E-8	222 (Slices)	112
ALMA. Tech [21]	XCV50E-8	222 (Slices)	105
Amphion Ltd [19]	XCV50E-8	204 (Slices)	90

The proposed architecture shows significant improvements in comparison with the existing implementations [29, 30, 19, 21, 20], which perform the RGB to YCrCb conversion, in terms of the area consumed and the maximum running clock frequency.

3.2 Floating-Point Matrix Multiplier for 3D Affine Transformations

Floating-Point Matrix Multiplication (FPMM) is a basic operation in many scientific computing applications involving large dynamic range. In this section, the floating-point adder and multiplier proposed in [6] are used as basic components for the implementation of a parallel FPMM designed for 3D affine transformations.

3.2.1 Mathematical Background

In computer graphics the most popular method for representing an object is the polygon mesh model. In a simplest case, a polygon mesh is a structure that consists of polygons represented by a list of (x,y,z) coordinates that are the polygon vertices. Thus the information we store to describe an object is finally a list of points or vertices [31]. 3D affine transformations are the transformations that involve rotation, scaling, shear and translation. A matrix can represent an affine transformation and a set of affine transformations can be combined into a single overall affine transformation [31]. Using matrix notation, a Vertex V is transformed to V^* (* denotes the transformed vertex) under translation, scaling and rotation, which are the most commonly used transformations in computer graphics, as:

$$V^* = T \times V \tag{6}$$

$$\begin{pmatrix} x^* \\ y^* \\ z^* \\ 1 \end{pmatrix} = \begin{pmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{7}$$

Consider an object represented with *N* vertices. The New Position (*NP*) of the object when applying a transformation can be calculated as follows [4]:

$$NP = T \times OP \tag{8}$$

where:

- *T* is the matrix transform.
- *OP* is a *(4,N)* matrix contains the Old vertices Position.
- *NP* is a *(4,N)* matrix contains the New vertices Position.

$$\begin{pmatrix} x_0^* & x_1^* & \cdots & x_{N-1}^* \\ y_0^* & y_1^* & \cdots & y_{N-1}^* \\ z_0^* & z_1^* & \cdots & z_{N-1}^* \\ 1 & 1 & \cdots & 1 \end{pmatrix} = \begin{pmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x_0^* & x_1^* & \cdots & x_{N-1}^* \\ y_0^* & y_1^* & \cdots & y_{N-1}^* \\ z_0^* & z_1^* & \cdots & z_{N-1}^* \\ 1 & 1 & \cdots & 1 \end{pmatrix} \quad (9)$$

3.2.2 *Proposed Architecture*

Equation 9 can be mapped into the proposed architecture shown in Figure 4.

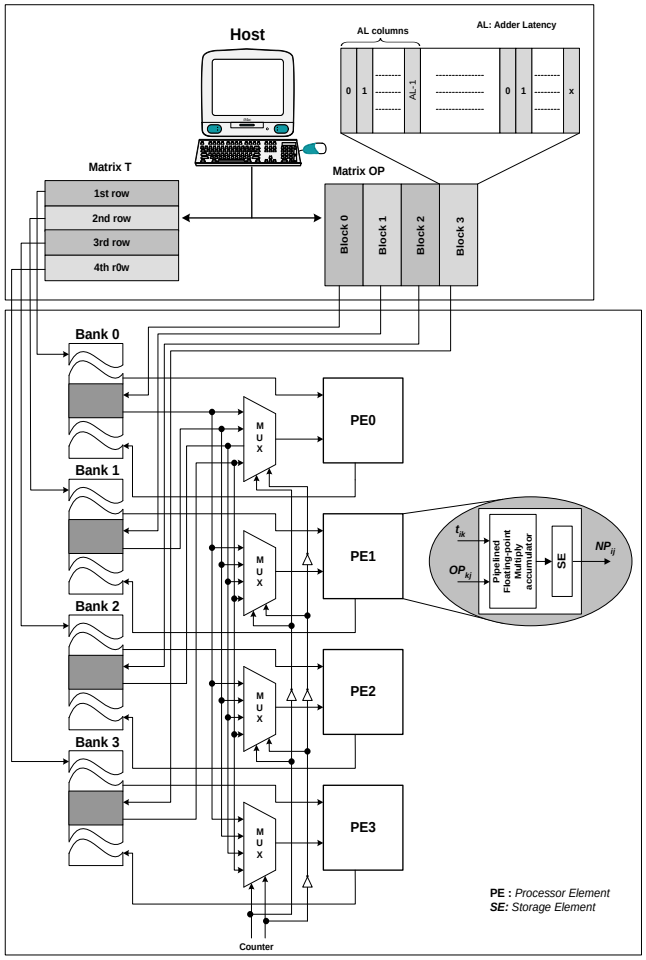

Figure 4. The proposed parallel FPMM architecture for 3D affine transformations

The multiplier consists of four identical Processor Elements (PEs). Each PE comprises a pipelined floating-point Multiply ACcumulator (MAC) and a register for final result storage. The vertices coordinates are represented using the IEEE single-precision real numbers. The MAC has been implemented using two different approaches:

- The pipelined floating-point library from Celoxica [32]. It is a platform-independent core. It allows the programmer to perform floating-point operations in a pipelined manner on single precision floating-point numbers.
- The proposed floating-point adder and multiplier described in the [6]. The floating-point multiplier used is the Xilinx CoreGen based approach.

For both approaches, the floating-point adders and multipliers used are pipelined and have different latencies. The floating-point Adder and Multiplier Latencies from Celoxica's library are $AL=10$ and $ML=10$ respectively. The input transform matrix *T* is partitioned into four rowwise blocks, which gives one row per block. Each block is stored in one of the four available banks. The matrix *OP* is partitioned into four columnwise blocks, likewise matrix *T*, each block is stored in one of the banks. In addition, due to *AL* value, each block of the matrix *OP* is partitioned into columnwise sub-blocks. Each subblock contains *AL* columns and the last one is padded with columns of zeros if $N \bmod AL \neq 10$ (*N* is the number of vertices).

Figure 5 illustrates the timing diagram when performing a multiplication of one row of the transform matrix *T* with one sub-block of the matrix *OP* as shown in equation 10 in the case of $AL=10$:

$$\begin{aligned} \begin{pmatrix} NP_{i0} & NP_{i1} & \cdots & NP_{i9} \end{pmatrix} = \begin{pmatrix} t_{i0} & t_{i1} & t_{i2} & t_{i3} \end{pmatrix} \times \\ \begin{pmatrix} OP_{00} & OP_{01} & \cdots & OP_{09} \\ OP_{00} & OP_{11} & \cdots & OP_{19} \\ OP_{00} & OP_{21} & \cdots & OP_{29} \\ OP_{00} & OP_{31} & \cdots & OP_{39} \end{pmatrix} \end{aligned} \quad (10)$$

The number of clock cycles required for the entire computation of the matrix *NP* is:

$$C = (N/(p \times AL)) \times ((AL + ML - 1) + (4 \times AL)) \quad (11)$$

where:

- *p* Number of PEs
- *AL* is the Adder Latency
- *ML* is the Multiplier Latency
- $(4 \times AL)$ is the size of the *OP* sub-matrices

3.2.3 *Hardware Implementation and Results*

The proposed architecture has been implemented and tested on the RC1000 prototyping board. The implementation based Celoxica's floating-point library consumes 99% of the available FPGA area and runs with a maximum frequency of 50 MHz, while the implementation based on the proposed floating-point adder and multiplier consumes 70% of the target FPGA area and can be run at a maximum clock frequency of 85 MHz. The parallel FPMM architecture has been synthesized on Virtex-II Pro FPGA in order to exploit the additional features and resources available on this device. Table 3 illustrates the performance obtained for the proposed architectures when using the two different design approaches for the MAC implementation.

Results obtained show that the parallel FPMM based on our proposed floating-point addition and multiplication cores gives better performance in comparison with the one based on the pipelined floating-point library from Celoxica. This mainly, because of the suitability of the different components used in our implementation for the targeted FPGA device.

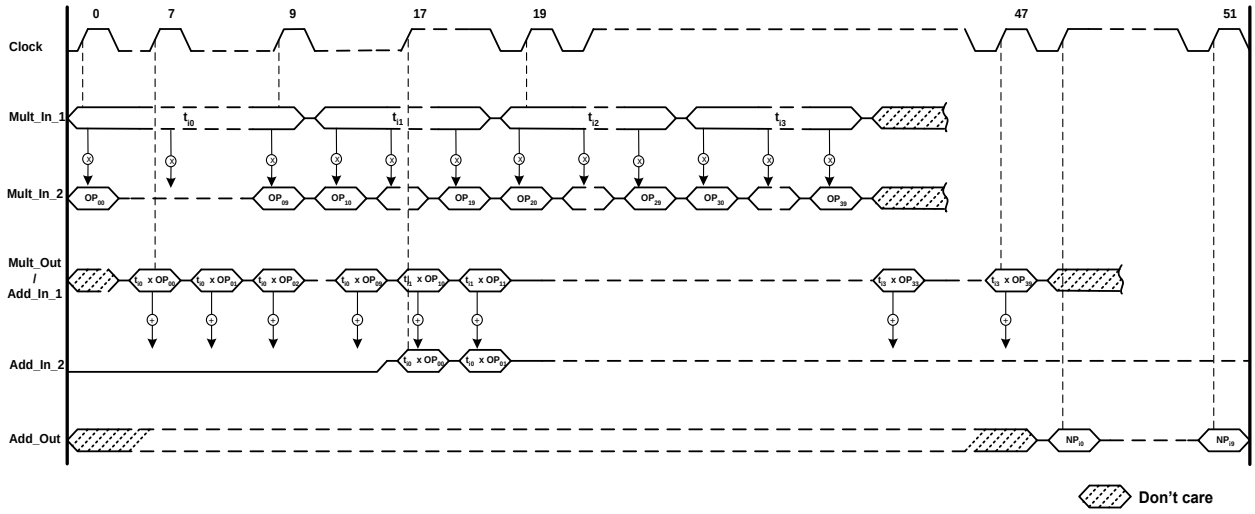

Figure 5. Timing diagram when performing a multiplication of one row of the transform matrix *T* with one sub-block of the matrix *OP*

Table 3. Area/Speed Implementation report for the proposed FPMM-Target FPGA XC2VP125 (ff1704-7)

MAC used	Area		Speed (MHz)
	Logic (%)	18×18 Mults	
Proposed floating-point adder / multiplier	39	16	215
Celoxica pipelined floating-point library	43	64	119

It is worth noting that, due to the application requirements, the sizes of the manipulated matrices *T* and *OP* are (4×4) and $(4\times N)$ respectively. The maximum value of *N* depends only on the available off-chip storage resources.

On the RC1000 board, *N* can be any value up to 2^{18}. Other sizes can be performed using the proposed architecture by applying a different partitioning strategy on the matrices *T* and *OP* at the host level.

In [15], which is the most recent work concerning 32-bit floating-point matrix multiplication, two implementations are presented. The results presented are only the one obtained for 64-bit input data. Therefore, a fair comparison can not be made with our implementation. The work presented in [16], studied the implementation of floating-point arithmetic and used matrix multiplication as an example application. Results obtained show that a 96×96 FPMM, implemented on a Virtex XC4044XL FPGA device, can achieve a maximum running frequency of 50 MHz. In [18], the authors investigated the influence of the floating-point MACs on the performance of a matrix multiplication algorithm. Results obtained, show that the maximum and best running frequency for a 1024×1024 FPMM is 33 MHz on the Annapolis MicroSystems board.

4. CONCLUSION

This paper describes a system for matrix algorithm cores generation used in image processing applications. The system provides a catalogue of user-customisable cores ranging in three different matrix algorithm categories: (i) matrix operations, (ii) matrix transforms and (iii) matrix decomposition.

The system includes a GUI to help the users customize the cores to be generated to meet the requirements of their applications.

The design process for the core generation can be broken down into four main steps:

1. Selecting the algorithm category;
2. Selecting the operation, transform or decomposition; depending on the selected category;
3. Setting the design parameters; and
4. Generating the core.

The output from the system can be an EDIF or a bitstream file.

The developed cores can be ranged into two different categories: (i) general purpose cores; and (ii) specific application cores. The methodology used in the design and implementation of two specific application cores is presented. The first core is a fully pipelined matrix multiplier for colour space conversion, which is used in many commercial applications such as face detection as well as the JPEG and MPEG imaging, based on distributed arithmetic principles while the second one is an efficient parallel matrix multiplier designed for 3D affine transformations used in computer graphics applications such as entertainment (e.g. movies and computer games), Computer Aided Design (CAD), medical visualisation (e.g. MRI/CT and virtual surgery).

The performance in terms of the area used and the maximum clock frequency has been assessed for the CSC and has shown that it can be run with a higher frequency and consumes less area when compared with existing systems. Result obtained for the core dedicated for 3D affine transformations has shown that FPGA implementation can achieve the performance of a graphics card.

REFERENCES

[1] A. Amira, "A Custom Coprocessor for Matrix Algorithm", PhD thesis, School of Computer Science, The Queen's University of Belfast, 2001.

[2] A. Beaumont-Smith, M. Liebelt, C. C. Lim and K. To, "A Digital Signal Multi-Processor for Matrix Application", Proceedings of the 14th Australian Microelectronics Conference, Melbourne, October 1997.

[3] S. P. Periyacheri, A. Jones, A. Nayak, D. Zaretsky, P. Banerjee, N. Shenoy and A. Choudhary, "A MATLAB Compiler for Distributed, Heterogeneous, Reconfigurable Computing Systems", Proceedings of the 11th International Conference Parallel and Distributed Computing and Systems, Cambridge, MAUSA, November 1999.

[4] F. Bensaali, A. Amira and A. Bouridane, "Accelerating Matrix Product on Reconfigurable Hardware for
Image Processing Applications", IEE Proceedings on Vision, Image and Signal Processing-Special Issue on Rapid Prototyping of Signal Processing Algorithms, Vol. 153, Issue 6, pp 739-746, December 2006.

[5] F. Bensaali and A. Amira, "Accelerating Colour Space Conversion on Reconfigurable Hardware", Image and Vision Computing, Elsevier, Vol. 23, Issue 11, pp 935-942, October 2005.

[6] F. Bensaali, A. Amira and R. Sotudeh, "Floating-Point Matrix Product on FPGA", ACS/IEEE International Conference on Computer Systems and Applications, Jordan, May 2007.

[7] I. S. Uzun, A. Amira, A. Ahmedsaid and F. Bensaali, "Towards A General Framework for an FPGA-based FFT Coprocessor", Preceedings of the Seventh IEEE International Symposium on Signal Processing and its Application, Vol. 1, pp. 617-620, Paris, France, July 2003.

[8] I. S. Uzun and A. Amira, "Design and FPGA Implementation of Non-Separable 2-D Biorthogonal Wavelet Transforms for Image/Video Coding", Proceedings of the IEEE International Conference on Image Processing, Vol.4, pp. 2825-2828, Singapore, October 2004.

[9] A.Ahmedsaid and A.Amira, "Accelerating SVD On Reconfigurable Hardware For Image Denoising", Proceedings of the IEEE International Conference on Image Processing, Vol. 1, pp. 259-262, Singapore, October 2004.

[10] A. Ahmedsaid, A. Amira and A. Bouridane, "Improved SVD Systolic Array and Implementation on FPGA", Proceedings of the IEEE International Conference on Field-Programmable Technology, pp. 35-42, Tokyo, Japan, December 2003.

[11] O. Mencer, M. Morf, M. J. Flynn, "PAM-Blox: High Performance FPGA Design for Adaptive Computing", Proceedings of the IEEE Symposium on Field-Programmable for Custom Computing Machines, Napa, California, pp. 167-174, April 1998.

[12] R. S. Grover, W. Shang, Q. Li, "An Improved Architecture for Bit-Level Matrix Multiplication", Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Vol. IV, Las Vegas, Nevada, pp. 2257-2264, June 2000.

[13] L. Jianwen and J. C. Chuen, "Partially Reconfigurable Matrix Multiplication for Area and Time Efficiency on FPGAs", Proceedings of the EUROMICRO Systems on Digital system Design, pp. 244-248, Rennes, France, August 2004.

[14] C. R. Wan and D. J. Evans, "Nineteen Ways of Systolic Matrix Multiplication", International Journal in Computer Mathematics, Vol. 69, pp. 39-69, 1998.

[15] L. Zhuo and V. K. Prasanna, "Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on FPGAs", Proceedings of the 18th International Parallel and Distributed Processing Symposium, pp. 92- 1001, New Mexico, USA, April 2004.

[16] I. Sahin, C. Gloster, and C. Doss "Feasibility of Floating-Point Arithmetic in Reconfigurable Computing Systems", Proceedings of the 3rd Military and Aerospace Applications of Programmable Devices and Technology Conference, Maryland, USA, September 2000.

[17] Y. Dou, S. Vassiliadis, G. Kuzmanov and G. Gaydadjiev, "64-bit Floating-Point FPGA Matrix Multiplication", Proceedings of the 13th international Symposium on Field-Programmable Gate Arrays, California, February 2005.

[18] W. B. Ligon III, S. McMillan, G. Monn, K. Schoonover, F. Stivers and K. D. Underwood, "A Re-Evaluation of the Practicality of Floating- Point Operations on FPGAs", Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 206-215, Napa, California, April 1998.

[19] Datasheet (www.amphion.com), "Color Space Converters", Amphion semiconductor Ltd, DS6400 V1.1, April 2002.

[20] Application Note (www.cast-inc.com), "CSC Color Space Converter", CAST Inc, April 2002.

[21] Datasheet (www.alma-tech.com), "High Performance Color Space Converter", ALMA Technologies, May 2002.

[22] Datasheet, "RC1000 Development Platform Product
Brief", v1.1, Celoxica Ltd., August 2002.

[23] A. Albiol, L. Torres and E.J. Delp, "An unsupervised
color image segmentation algorithm for face detection applications", Proceedings of the International Conference on Image Processing, pp. 681-684, Vol. 2, October 2001.

[24] P. Kuchi, P. Gabbur, P. S. Bhat and S. David, "Human Face Detection and Tracking using Skin Color Modelling and Connected Component Operators", The IETE Journal of Research, Special issue on Visual Media Processing, Vol. 38, No. 3 & 4, pp. 289-293, May 2002.

[25] M. Bartkowiak, "Optimizations of Color Transformation for Real Time Video Decoding", Digital Signal Processing for Multimedia Communications and Services, EURASIP ECMCS 2001, Budapest, September 2001.

[26] J. L. Mitchell and W. B. Pennebaker, "MPEG Video Compression Standard", Chapman & Hall, 1996.

[27] Application Note, "Xilinx CoreGen and Handel-C", AN 58 v1.0, 2001.

[28] Manual , "Handel-C Language Reference Manual", RM-1003-4.2, Celoxica Ltd., 2004.

[29] L. Pillai, "Color Space Converter: Y'CrCb to R'G'B"', Application Note, XAPP283 (v1.3.1), Xilinx Inc., March 2005.

[30] Application Note, "Color Space Converter: MegaCore Function User Guide", (v.2.2.0) Altera, June 2004.

[31] A. Watt, "3D Computer Graphics," Addison-Wesley, 2000.

[32] Manual, "Pipelined Floating-Point Library Manual", Celoxica Ltd., 2005.