INTEGRATION, the VLSI journal 42 (2009) 346–355

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

An embedded, FPGA-based computer graphics coprocessor with native geometric algebra support

Silvia Franchini [a], Antonio Gentile [a], Filippo Sorbello [a], Giorgio Vassallo [a], Salvatore Vitabile [b,*]

[a] Dipartimento di Ingegneria Informatica, Università di Palermo, V.le delle Scienze (Edificio 6), 90128 Palermo, Italy
[b] Dipartimento di Biotecnologie Mediche e Medicina Legale, Università di Palermo, Via del Vespro, 90127 Palermo, Italy

ARTICLE INFO

Article history:
Received 19 March 2008
Received in revised form
16 September 2008
Accepted 17 September 2008

Keywords:
Clifford algebra
Computational geometry
Embedded coprocessors
Application-specific processor
FPGA-based prototyping

ABSTRACT

The representation of geometric objects and their transformation are the two key aspects in computer graphics applications. Traditionally, computer-intensive matrix calculations are involved in modeling and rendering three-dimensional (3D) scenery. Geometric algebra (aka Clifford algebra) is attracting attention as a natural way to model geometric facts and as a powerful analytical tool for symbolic calculations. In this paper, the architecture of Clifford coprocessor (CliffoSor) is introduced. CliffoSor is an embedded parallel coprocessing core that offers direct hardware support to Clifford algebra operators. A prototype implementation on a programmable gate array (FPGA) board is detailed. Initial test results show the potential to achieve a 20× speedup for 3D vector rotations, a 12× speedup for Clifford sums and differences, and more than a 4× speedup for Clifford products, compared to the analogous operations in GAIGEN, a standard geometric algebra library generator for general-purpose processors. An execution analysis of a raytracing application is also presented.

1. Introduction

Many research fields, such as machine vision, robotics, and computer graphics, rely heavily on geometric models of external reality. Software packages and graphics hardware accelerators have been designed to deal with the representation of points, lines, and planes and their transformations in three-dimensional (3D) space, such as rotations, translations, reflections, and projections.

Traditionally, computer graphics packages are implemented using homogeneous coordinates and a matrix package [1]. This approach creates a separation between geometric reasoning and matrix-based processing. Programming geometric models requires reasoning in affine and projective geometries; whereas the implementation of graphic objects and their transformations using a matrix formulation require a geometric interpretation of vector calculations that is often problematic and prone to error.

Geometric algebra, which has been studied for more than a century, is attracting attention in recent years as a powerful new computing paradigm for a number of research fields. It offers a natural way to model geometric objects independent of their coordinates with its powerful and simple symbolic formalism [2]. Also known as Clifford algebra (from its main contributor, W. K. Clifford [3]), this type of algebra offers an integrated approach to geometric modeling and algorithms. It permits the specification of geometric objects at a coordinate-free, unified level, where points, lines, and planes become basic elements of computation and their transformations are executed directly [2,4,5].

The exploitation of geometric algebra's symbolic computing power requires efficient support for its powerful operators and data types. Previous implementations had to resort to changing complex translations into matrix equations and standard matrix libraries. Many software libraries (CLU [6], GluCat [7]), packages for symbolic and numerical environments (Maple [19,20], Mathematica, MatLab), and stand-alone programs (CLUit [6], CLICAL [8]) have been developed to solve geometric algebra expressions directly on general-purpose processors. In these implementations, a hierarchy of abstraction layers is used to program geometric algebra expressions, lowering their computational advantage through the introduction of significant overhead. These approaches address general *n*-dimensional Clifford algebra, whereas in the case of computer graphics and machine vision applications, an optimized four-dimensional (4D) implementation should be sought as a good compromise between complexity and benefits. The GAIGEN library generator [9] is a software library that addresses this need.

This paper examines the direct support of geometric algebra data types and operators in hardware. Related research [10] presents a coprocessor for native geometric product execution, for algebras of dimension up to 8. A geometric product is executed by computing the components of its basis blades in a pipeline

* Corresponding author. Fax: +390916529124.
E-mail address: vitabile@unipa.it (S. Vitabile).

0167-9260/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2008.09.010

fashion. No other geometric algebra operation is supported directly by the coprocessor, with the exception of a prototype implementation on a field-programmable gate array (FPGA) board.

This paper presents a novel coprocessing architecture, the Clifford coprocessor (*CliffoSor*). Aimed at embedded machine vision and computer graphics applications, either as part of robotic platforms or dedicated graphic boards, this architecture directly supports 4D homogeneous operands (namely scalars, vectors, bivectors, trivectors, and pseudoscalars) and their operations (geometric products, outer products, left and right contractions, sums and differences). Clifford numbers, their expressions in terms of homogeneous numbers, and related operations are described in Section 2. The CliffoSor architecture is currently implemented as a coprocessor core hosted on a PCI-based FPGA board. The core is described using a mixed hardware description language design that incorporates both Handel-C and VHDL hardware description languages. An early version of the CliffoSor architecture was presented in [11,12]. Test results are presented for 4D geometric products, 4D sums/differences, and 3D rotations on CliffoSor, compared to the same operations implemented for general-purpose processors by the 4D library generated by the GAIGEN library generator [9]. In addition, a raytracer application and 3D model rotation have been implemented and executed on CliffoSor to prove its effectiveness on its application domain. These results suggest that there is great potential for speedup in comparison with the execution on a traditional processor. To realize this potential, however, the core must be placed as close as possible to the memory subsystem, with dedicated DMA access. The potential cycle speedup is demonstrated for products ($4\times$), sums ($12\times$), and rotations ($20\times$).

The paper is organized as follows. Section 2 briefly introduces geometric algebra and its implications for computer graphics. In Section 3, design considerations are discussed along with their implications for efficient hardware implementation of operand types and operations. Section 4 details the architecture of CliffoSor and its implementation of FPGA. Experimental results are described in Section 5, and Section 6 summarizes the conclusions from this work.

2. Geometric algebra and computer graphics

Geometric algebra (Clifford algebra) is a powerful mathematical tool for symbolic calculations. It is applied to different fields of research such as computer graphics, CAD/CAM, robotics, physics, and any application in which description and manipulation of geometric entities are very important. As a unifying mathematical language, geometric algebra comprises a number of mathematical descriptions widely used in computer graphics, such as quaternions to represent rotations and Lie algebras for rigid body motion descriptions [13]. A brief introduction to the theory is given in the following sections. Interested readers may find further details in [2,4,5,14,17,18,22].

2.1. Theoretical background

Clifford algebra expands classical linear algebra concepts, such as scalars and vectors, by introducing two-, three-, or higher-dimensional subspaces, called blades. In Clifford algebra, vectors can be combined using the outer product to obtain higher-dimensional entities, such as bivectors (representing planes) and trivectors (representing 3D subspaces). For example, if $\mathbf{a}$ and $\mathbf{b}$ are vectors, their outer product $\mathbf{a}\wedge\mathbf{b}$ is a blade of *grade* 2 (or a 2-*blade*), which represents the two-dimensional oriented subspace that contains $\mathbf{a}$ and $\mathbf{b}$. Such blades of grade 2 are called *bivectors*. The outer product of three vectors results in a blade of grade 3 (also called a *trivector*), and so on. According to this formalism, vectors are 1-blades, and scalars are 0-blades. Geometric algebra also defines the operators with which these subspaces can be manipulated. It is possible, in fact, to add and subtract subspaces of different dimensions by means of composition, and even to multiply them, resulting in powerful expressions that can express many geometric relations and concepts.

Considering an orthonormal basis $\{\boldsymbol{e}_1,\boldsymbol{e}_2,\ldots,\boldsymbol{e}_n\}$ of R^n, a possible basis of the n-dimensional Clifford algebra, Cl_n, consists of all k-dimensional subspaces with $k\leqslant n$, as listed in Table 1. The total number of these subspaces is 2^n, while the number of k-dimensional subspaces is given by the binomial coefficient $n!/(k!(n-k)!)$. The basis blade of the highest dimension is called the *pseudoscalar*. A generic number in Cl_n is called a *multivector*, which is a linear combination with real coefficients of the previously mentioned basis elements.

The most important operator of Clifford algebra is the *geometric product* (also called the *Clifford product*). The geometric product between two multivectors is performed by "multiplying" each component of an operand individually with each component of the other operand, and simplifying the resulting terms by means of the set of axioms listed in Table 2.

A particular type of multivector is the *rotor*, formed by a scalar element and a bivector element. Rotors are used to perform rotations of subspaces of any dimension. If $\boldsymbol{R}$ denotes a rotor and $\boldsymbol{v}$ denotes a vector, the rotated vector $\boldsymbol{v}'$ can be calculated as $\boldsymbol{v}'=\boldsymbol{R}\boldsymbol{v}\boldsymbol{R}^{\dagger}$, where $\boldsymbol{R}^{\dagger}$ is the reverse of $\boldsymbol{R}$. Interestingly, a generic rotor can be derived from the geometric product of two unitary vectors, where the rotation is in the plane determined by the two vectors, and the rotation angle is twice the angle between them.

2.2. 4D Clifford homogeneous numbers

In the case of computer graphics and machine vision applications, 4D geometric algebra is a reasonable compromise between complexity and benefits. A generic Clifford algebra multivector is a linear combination with real coefficients of the basis elements of Clifford space; in four dimensions, a generic multivector can be written as

$$a_0+a_1\boldsymbol{e}_1+a_2\boldsymbol{e}_2+a_3\boldsymbol{e}_3+a_4\boldsymbol{e}_4+a_{12}\boldsymbol{e}_1\boldsymbol{e}_2+a_{13}\boldsymbol{e}_1\boldsymbol{e}_3+a_{14}\boldsymbol{e}_1\boldsymbol{e}_4+a_{23}\boldsymbol{e}_2\boldsymbol{e}_3+a_{24}\boldsymbol{e}_2\boldsymbol{e}_4+a_{34}\boldsymbol{e}_3\boldsymbol{e}_4+a_{123}\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3+a_{124}\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_4+a_{134}\boldsymbol{e}_1\boldsymbol{e}_3\boldsymbol{e}_4+a_{234}\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_4+a_{1234}\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_4$$

It is possible to collect generic multivector basis elements of the same grade to build 4D homogeneous Clifford numbers: *scalar*, *vector*, *bivector*, *trivector*, and *pseudoscalar* as shown in Table 3.

Table 1
Basis elements (blades) in n-dimensional Clifford algebra.

Dimension	Element	Blade
0	Scalar	1
1	Vector	$\boldsymbol{e}_i$
2	Bivector	$\boldsymbol{e}_i\boldsymbol{e}_j$
3	Trivector	$\boldsymbol{e}_i\boldsymbol{e}_j\boldsymbol{e}_k$
…	…	…
n	Pseudoscalar	$\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\ldots\boldsymbol{e}_n$

Table 2
Clifford product axioms.

$\boldsymbol{e}_i\boldsymbol{e}_i=\pm1$	$i=1,2,\ldots,n$
$\boldsymbol{e}_j\boldsymbol{e}_i=-\boldsymbol{e}_i\boldsymbol{e}_j$	$i\neq j=1,2,\ldots,n$
$\lambda\boldsymbol{e}_i=e_i\lambda$	$i=1,2,\ldots,n$

Table 3
4D homogeneous geometric algebra elements.

Homogeneous element (symbol)	Grade	Analytic expression
Scalar (s)	0	a_0
Vector (v)	1	$a_1\boldsymbol{e}_1+a_2\boldsymbol{e}_2+a_3\boldsymbol{e}_3+a_4\boldsymbol{e}_4$
Bivector (b)	2	$a_{12}\boldsymbol{e}_1\boldsymbol{e}_2+a_{13}\boldsymbol{e}_1\boldsymbol{e}_3+a_{14}\boldsymbol{e}_1\boldsymbol{e}_4+a_{23}\boldsymbol{e}_2\boldsymbol{e}_3+a_{24}\boldsymbol{e}_2\boldsymbol{e}_4+a_{34}\boldsymbol{e}_3\boldsymbol{e}_4$
Trivector (t)	3	$a_{123}\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3+a_{124}\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_4+a_{134}\boldsymbol{e}_1\boldsymbol{e}_3\boldsymbol{e}_4+a_{234}\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_4$
Pseudoscalar (p)	4	$a_{1234}\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_4$

A homogeneous number is an element that contains blades of only the same grade.

In terms of homogeneous elements, a generic multivector m can be rewritten as an ordered five-element tuple $m=(s,v,b,t,p)$.

3. A hardware implementation of 4D geometric algebra

From the previous section, the implementation of 4D geometric products on two generic multivectors will result in a large number of multiplications between blade coefficients, and accumulation of partial products into the proper result blade coefficient. Since generic 4D multivectors consist of up to $2^4=16$ blades, a geometric product will require up to $(2^4)^2=2^8=256$ multiplications between coefficient pairs and $2^4(2^4-1)=240$ additions. Since in most cases many coefficients are zero, thoughtful processing can avoid wastage of resources. This computational complexity is indeed the most significant impediment to a wide-spread use of geometric algebra, which hides such complexity behind its powerful operators.

Two observations lead to the design choice to implement native support for Clifford operations on 4D homogeneous numbers. First, it has been widely observed that in most of the common applications of geometric algebra, Clifford numbers appear in the form of either a single homogenous number or simple compositions of homogeneous parts. Second, the distributive property holds for operations on homogeneous Clifford numbers, thus allowing operations on generic multivectors to be sequenced as multiple operations on their homogeneous parts. This choice would make the worst case scenario a product of two bivectors for a total of 36 multiplications and 28 additions, considerably better than the general case. A closer look at the application profiles reveals that the most frequent situation is the case of vector–bivector (or trivector–bivector) products, which would require 24 multiplications and 16 additions. CliffoSor is therefore designed to make the most frequent case faster, resorting to a higher-level Application Programming Interface (API) to solve more complex and less frequent operations with multiple calls to the specialized hardware.

3.1. Bit-mask encoding of homogeneous element blades

As described in Table 3, 4D homogeneous elements carry a variable number of blades, where each blade is a basis element-coefficient pair. In n dimensions, given an orthonormal basis $B=\{\boldsymbol{e}_1,\boldsymbol{e}_2,\boldsymbol{e}_3,\ldots,\boldsymbol{e}_n\}$ of a Euclidean vector space R^n, the corresponding basis of the Clifford space Cl_n is defined by 2^n elements, namely all possible combinations of basis elements. To represent them efficiently in hardware, an n-bit mask is associated with each blade, where each bit is associated with a basis element $\boldsymbol{e}_i$, $i\in[1,n]$, with $\boldsymbol{e}_1$ the least significant bit. In four dimensions, each one of the 2^4 basis blades is associated with a four-bit mask, as described in Table 4, where the bit masks associated with each blade are listed below the corresponding coefficient.

Table 4
Homogeneous element format.

	Tag	Blade coefficients and bit masks					
		A	B	C	D	E	F
Scalar	000	a_0 0000	0	0	0	0	0
Vector	001	a_1 0001	a_2 0010	a_3 0100	a_4 1000	0	0
Bivector	010	a_{12} 0011	a_{13} 0101	a_{14} 1001	a_{23} 0110	a_{24} 1010	a_{34} 1100
Trivector	101	a_{234} 1110	a_{134} 1101	a_{124} 1011	a_{123} 0111	0	0
Pseudoscalar	100	a_{1234} 1111	0	0	0	0	0

Each homogeneous element is represented by a vector of seven elements: a tag specifying the element, and six elements to store blade coefficients. Bit masks associated to each blade are listed below the corresponding coefficient. Shaded cells are non-used vector elements.

This mapping exhibits an interesting bit-inversion property between basis elements belonging to dual homogeneous elements, such as scalar–pseudoscalar, and vector–trivector pairs. This property will be usefully exploited in the geometric product implementation details in Section 4.4.2.2). A single format is used to represent all five types of homogeneous elements, as described in Table 4.

Each homogeneous element is represented by a vector of seven elements, namely a three-bit *tag*, and six 32-bit elements (A–F) to store the blade coefficients. The *tag* is encoded such that dual elements differ in the most significant bit. While choosing a fixed format causes some storage inefficiencies, as only the bivector uses all seven elements, this choice enables a simpler design, and is thus preferable to the more compact blade representation given in [10].

3.1.1. Computing the bit mask of results

Based on the approach proposed in [15], the following technique has been developed to calculate geometric products, outer products, and left or right contractions between two blades. These operations are performed by a three-step process. First, for all products, the scalar values associated with the input blades are multiplied to yield the coefficients of the result. Second, the blade bit mask of the result is computed, according to the following cases:

- for geometric products, the result blade bit mask is the XOR of the two input blade bit masks;
- for outer products, the result blade bit mask is the XOR of the two input blade bit masks, with the exception that the result will be zeroed if the bitwise AND of the two bit masks is different from zero.
- for left (right) contractions, the result blade bit mask is the XOR of the two input blade bit masks, with the exception that the result will be zeroed if the bitwise AND between the first (second) bit mask and the complemented second (first) bit mask is different from zero.

Third, the sign of the result depends on the input blades according the axioms listed in Table 2, and is implemented with a look-up table.

3.2. Operations on homogeneous numbers

The subdivision of a Clifford multivector into homogeneous elements allows these elements to be treated as input operands of Clifford algebra operations. A Clifford operation on generic multivectors $\boldsymbol{m} = (s, v, b, t, p)$ can be expressed as a sequence of binary operations on their homogeneous elements. In particular, operations that are supported natively in CliffoSor are additions/subtractions, geometric products, left and right contractions, and outer products.

Operations on non-homogeneous numbers are executed by the software API, which translates them into the appropriate sequence of binary operations on homogeneous elements. The only exception is the case of 3D rotations, which is treated separately (see Section 3.3 below).

The result type of a given operation between two homogeneous operands can be determined from the types of the input operands. The outer product, left contraction, and right contraction generate a single homogeneous number as a result, while the geometric product, sum, and difference generate two homogeneous numbers as a result. The full geometric product between two bivectors generates three homogeneous numbers (namely a scalar, a bivector, and a pseudoscalar), and is executed by the API with multiple calls to CliffoSor.

Other Clifford operations on homogeneous numbers, such as the unary operators dual, reverse, conjugate, grade involution, inverse, and the binary operators meet and join, are implemented in the API as special cases of the supported operations.

3.2.1. Considerations of product operations

With proper handling of coefficients, different product operations can re-use the same functional unit, leading to a simplified design. For example, both (vector∗vector) and (vector∗trivector) products can be executed by the same unit, as illustrated below:

Vector∗vector case:

$$vect_1 = A_1\boldsymbol{e}_1 + B_1\boldsymbol{e}_2 + C_1\boldsymbol{e}_3 + D_1\boldsymbol{e}_4$$
$$vect_2 = A_2\boldsymbol{e}_1 + B_2\boldsymbol{e}_2 + C_2\boldsymbol{e}_3 + D_2\boldsymbol{e}_4$$

$$\begin{aligned} vect_1 * vect_2 &= (A_1A_2 + B_1B_2 + C_1C_2 + D_1D_2) \\ &+ (A_1B_2 - B_1A_2)\boldsymbol{e}_1\boldsymbol{e}_2 + (A_1C_2 - C_1A_2)\boldsymbol{e}_1\boldsymbol{e}_3 \\ &+ (B_1C_2 - C_1B_2)\boldsymbol{e}_2\boldsymbol{e}_3 + (A_1D_2 - D_1A_2)\boldsymbol{e}_1\boldsymbol{e}_4 \\ &+ (B_1D_2 - D_1B_2)\boldsymbol{e}_2\boldsymbol{e}_4 + (C_1D_2 - D_1C_2)\boldsymbol{e}_3\boldsymbol{e}_4 \end{aligned}$$

Vector∗trivector case:

$$vect_1 = A_1\boldsymbol{e}_1 + B_1\boldsymbol{e}_2 + C_1\boldsymbol{e}_3 + D_1\boldsymbol{e}_4$$
$$trivect_2 = A_2\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_4 + B_2\boldsymbol{e}_1\boldsymbol{e}_3\boldsymbol{e}_4 + C_2\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_4 + D_2\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3$$

$$\begin{aligned} vect_1 * trivect_2 &= (A_1A_2 - B_1B_2 + C_1C_2 - D_1D_2)\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_4 \\ &+ (C_1D_2 + D_1C_2)\boldsymbol{e}_1\boldsymbol{e}_2 + (-B_1D_2 + D_1B_2)\boldsymbol{e}_1\boldsymbol{e}_3 \\ &+ (A_1D_2 + D_1A_2)\boldsymbol{e}_2\boldsymbol{e}_3 + (-B_1C_2 - C_1B_2)\boldsymbol{e}_1\boldsymbol{e}_4 \\ &+ (A_1C_2 - C_1A_2)\boldsymbol{e}_2\boldsymbol{e}_4 + (A_1B_2 + B_1A_2)\boldsymbol{e}_3\boldsymbol{e}_4 \end{aligned}$$

By comparing the two above algebraic expressions, one can observe that the blade coefficient calculation for the (*vector∗trivector*) case is obtained from the case of (*vector∗vector*) by applying the following coefficient-swapping rules:

$$\boldsymbol{e}_1\boldsymbol{e}_2 \leftrightarrow \boldsymbol{e}_3\boldsymbol{e}_4$$
$$\boldsymbol{e}_1\boldsymbol{e}_3 \leftrightarrow \boldsymbol{e}_2\boldsymbol{e}_4$$
$$\boldsymbol{e}_2\boldsymbol{e}_3 \leftrightarrow \boldsymbol{e}_1\boldsymbol{e}_4$$

3.3. Operations on non-homogeneous numbers—3D rotations

As described earlier, operations on multivectors are executed in API by translating them into sequences of supported operations on homogeneous parts of the multivectors.

The only operations on non-homogeneous elements that are supported natively on CliffoSor are 3D rotations, as they are important and frequent operations in computer graphics. 3D rotations require a chain of products involving non-homogeneous elements, such as *rotors* $\boldsymbol{R}$, generically composed of a scalar and a bivector:

$$\boldsymbol{R} = (s, b) = q_0 + q_1\boldsymbol{e}_{12} + q_2\boldsymbol{e}_{13} + q_3\boldsymbol{e}_{23}.$$

A 3D rotation of a generic vector $\boldsymbol{v} = a_1\boldsymbol{e}_1 + a_2\boldsymbol{e}_2 + a_3\boldsymbol{e}_3$ can be expressed as

$$\boldsymbol{v}' = \boldsymbol{R}\boldsymbol{v}\boldsymbol{R}^{\dagger}$$

where $\boldsymbol{v}'$ is the rotated vector and $\boldsymbol{R}^{\dagger}$ is the reverse of $\boldsymbol{R}$, $\boldsymbol{R}^{\dagger} = (s, -b) = q_0 - q_1\boldsymbol{e}_{12} - q_2\boldsymbol{e}_{13} - q_3\boldsymbol{e}_{23}$. A dedicated functional unit is implemented in CliffoSor to handle 3D rotations.

4. CliffoSor architecture

CliffoSor is an embedded coprocessor core that offers direct hardware support for Clifford algebra operations. CliffoSor performs operations between homogeneous elements (scalars, vectors, bivectors, trivectors, pseudoscalars) coded using 32-bit integers in two's complement representation. A prototype CliffoSor has been implemented on an FPGA board containing a Xilinx FPGA Virtex2000E hosted on the Celoxica Ltd. RC 1000 PCI board, pictured in Fig. 1. The board uses the PCI bus to interface with the host CPU and is equipped with 8 MB of SRAM for dynamic data storage, which is accessible to both the host CPU and the FPGA. The operating frequency of CliffoSor is 50 MHz, as required by the SRAM read/write access cycle.

4.1. System overview

Fig. 2 depicts the system architecture. CliffoSor is hosted on the FPGA housed in the Celoxica RC1000 board. Computer graphics applications (such as a raytracer and a 3D modeler) are implemented in a C++ host program.

An API controls the execution of Clifford operations, and translates operations on generic multivectors into sequences of operations on their homogeneous components, instantiating them to the CliffoSor.

Instructions and data are transferred through the PCI bus to the RC1000 SRAM area. Results are collected once CliffoSor signals execution completion. The Clifford Interface unit controls data exchanges between the PCI bus, the SRAM, and CliffoSor. The resource utilization of CliffoSor is detailed in Table 5.

Although the actual design would allow for a dual core in the same FPGA device, the limiting factor is the low bandwidth of the PCI interface currently used. Future CliffoSor designs will target PCI Express buses with higher transfer rates (in excess of 400 MB/s), capable of sustaining multiple core parallel operations. The phases of the system operation are described in the subsequent section.

4.2. System operation

The system operation is organized in six phases: the first phase is host–coprocessor synchronization (the handshaking phase); the second phase is instruction and data transfers from the host to the SRAM banks (the host-writing phase); the third phase is instruction and data transfers from the SRAM to CliffoSor (the

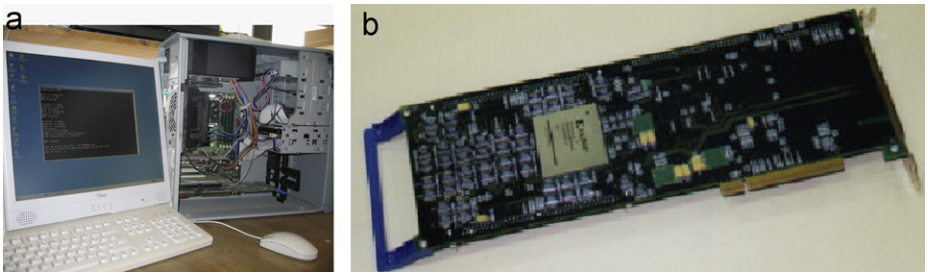

Fig. 1. (a) System setup and (b) Celoxica Ltd. RC 1000 PCI board.

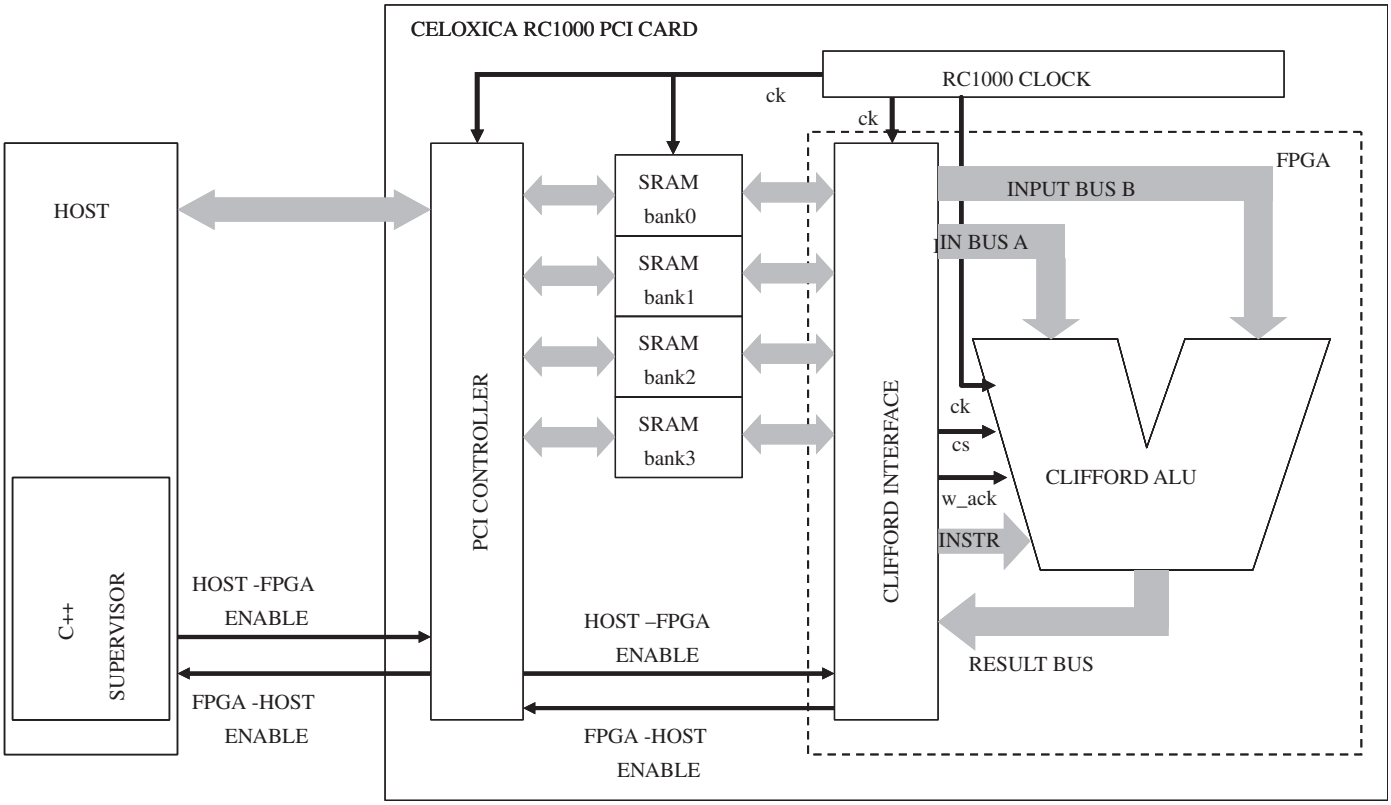

Fig. 2. CliffoSor system architecture.

coprocessor-reading phase); the fourth phase is instruction execution on CliffoSor (the execution phase); the fifth phase is the transfer of results from CliffoSor to the SRAM (the coprocessor-writing phase); the sixth phase is the transfer of data from SRAM to host (the host-reading phase). The system operation phases are depicted in Fig. 3.

4.3. Instruction format

A single instruction format is currently used, containing a 32-bit INSTRUCTION word followed by two 7×32-bit operands, for a total of 15×32-bit words (480 bits). The format is described in Table 6.

The INSTRUCTION word, illustrated in Table 7, contains a 4-bit OPCODE, specifying the operation to be executed on the two operands, followed by three bytes containing the ID for the result, operand A, and operand B, respectively. These IDs are used in the API to identify homogeneous numbers that are part of the same composite multivector.

The OPCODE field in the INSTRUCTION word specifies the Clifford operation to be performed. Each operand (see also Section 3.1) is represented by a 32-bit HEADER word followed by six 32-bit blade coefficients, in two's complement representation, as described in Table 8.

Table 9 details the operand HEADER word format. As described in Table 4, some of the coefficients may be zeros, according to the TAG field in the HEADER word. The content of the TAG field is described in Table 4. The operand HEADER word contains also the OPERAND ID byte, used to associate a homogeneous number with a specific multivector. CliffoSor is designed to perform operations with homogenous numbers, which may result in one or two homogeneous parts. The result format is described in Table 10.

4.4. CliffoSor architecture

Fig. 2 shows the CliffoSor architecture, which is composed of two main units, namely the Clifford Interface and the Clifford ALU.

The Clifford Interface unit controls data and signals for instructions and data transfers between the ALU and both the memory and bus infrastructure. A Celoxica proprietary host–board handshake protocol is used for the component synchronization and data/instructions transfer. A 32-bit data transfer occurs at a PCI bus frequency of 33 MHz, with a transfer rate of 132 MB/s. The SRAM memory banks allow for a single read/write cycle of four 32-bit words. The Clifford ALU unit natively executes Clifford algebra operations. Since the design targets computer graphics applications, the ALU has been optimized to execute Clifford 4D additions, subtractions, multiplications, and 3D rotations.

4.4.1. Clifford Interface

The Clifford Interface unit handles PCI–SRAM–CliffoSor data transfers. It is implemented using the Handel-C language, and uses the Celoxica proprietary handshake protocol for RC1000–PCI bus communications. It executes two main tasks: *SRAM read*, consisting of four transfers of four 32-bit words to transfer the 15×32-bit instruction vector from SRAM to the ALU instruction vector register and *SRAM write*, consisting of four transfers of four 32-bit words to write back results (a single 15×32-bit vector) from the ALU result vector register to SRAM.

Host–RC1000 synchronization occurs by exchanging a control byte and a status byte through the two homonymous ports. A SRAM read operation initiates as soon as the control byte from the host signals that SRAM data transfer from the host is complete. A SRAM write operation initiates once the Clifford ALU sets the *w_ack* signal. Once the four four-word transfers are completed, a status byte is sent back to the host to indicate result availability.

Table 5
CliffoSor resource utilization on a XILINX xcv2000e-6bg560

Operation	Slices	FF	4-LUT	IOB	Gates
Clifford multiplier	4911 25.5%	1715 4.5%	9271 24.1%	–	–
ALU controller	38 0.2%	15 0.03%	70 0.2%	–	–
Clifford adder	673 3.5%	827 2.1%	982 2.6%	–	–
Clifford rotator	2991 15.5%	430 1.1%	5777 15.1%	–	–
Clifford ALU	8444 43.9%	2945 7.7%	16,027 41.7%	–	211,278 8.33%
Clifford interface	958 4.9%	550 1.4%	683 1.8%	93 23%	12,512 0.49%
CliffoSor	9402 48.9%	3495 9.1%	16,710 43.5%	93 23%	228,802 9.01%

4.4.2. Clifford ALU

The Clifford ALU is composed of three functional units dedicated to native Clifford operation execution:

- a multiplier unit, for geometric products, outer products, left and right contractions;
- an adder unit, for sums and subtractions;
- a 3D rotation unit, for 3D rotations.

A controller unit decodes the instruction and selects the appropriate functional unit for its execution. Fig. 4 shows the Clifford ALU internal architecture. The RC1000 clock synchronizes the ALU operations.

4.4.2.1. Controller unit design. The controller unit supervises the ALU operation. After decoding the instruction, it enables the appropriate functional unit by setting the corresponding chip enable signal (namely *product_CE*, *adder_CE*, and *rotator_CE*).

After the enable signal is set for the currently active functional unit, the instruction execution is completed and the acknowledgement signal *w_ack* is set to signal execution completion for the Clifford Interface. As an example, the timing diagram for a product operation is illustrated in Fig. 5.

Table 6
CliffoSor instruction format

480b		
32b	7×32b	7×32b
INSTRUCTION	OPERAND A	OPERAND B

Table 7
INSTRUCTION field format

4b	8b	8b	8b	4b
31 28	27 20	19 12	11 4	3 0
N/U	OPERAND B ID	OPERAND A ID	RESULT ID	OPCODE

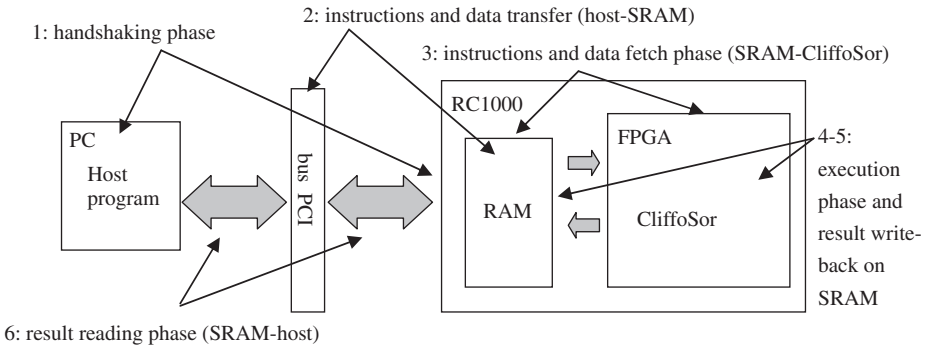

Fig. 3. System operation cycle.

Table 8
Operand format

32b	BLADES					
	32b	32b	32b	32b	32b	32b
HEADER	A	B	C	D	E	F

Table 9
Operand header format

21b		8b		3b	
31	11	10	3	2	0
NOT USED		OPERAND ID		TAG	

Table 10
Result format

32b	BLADES					
	32b	32b	32b	32b	32b	32b
TAG1	A1	B1	C1	D1	E1	F1
TAG2	A2	B2	C2	D2	E2	F2

4.4.2.2. Multiplier unit design. The *multiplier* functional unit executes the geometric product of two homogeneous elements. The result of a geometric product operation is a Clifford number composed of one or two homogeneous elements, whose coefficients are given as sums of intermediate sub-products. The sub-products are all the possible products between operand coefficients, and a $24\times$ multiplier bank is used to parallel the computation of all 24 sub-products.

A *mask generator* look-up table is designed to compute the appropriate sign for each sub-product as the result of three contributions: operand coefficient signs, algebra signature,[1] and blade multiplication. The LUT input is determined by the two operand tags. The LUT output is fed to a 24-bit register: if the *n*th bit of the mask register is set, then the *n*th sub-product must be sign inverted, or else left unchanged. Signed sub-products are then routed and added to the appropriate coefficient in the result register.

The symmetries observed in the geometric product operations for all the 24 possible combinations of the input operands led to the design of only three sub-units, namely *scalar unit*, *vector unit*, and *bivector unit*, in which all the route and add operations are performed.

The *scalar* unit is used for the following geometric product operations: scalars∗others, pseudoscalars∗others, others∗scalars, and others∗pseudoscalars. It performs no sums, but only field-to-field routing.

The *vector* unit is used in the following cases: vector∗vector, vector∗trivector, trivector∗vector, and trivector∗trivector; in all considered cases, operands are formed by four fields. The result is composed of two homogeneous numbers: the first one is a scalar or a pseudoscalar (with only one field filled) and the second is a bivector (with all six fields filled).

[1] The signature of the algebra is the pair of integers (p, q) where p is the number of basis elements which square to +1 and q is the number of basis elements which square to −1.

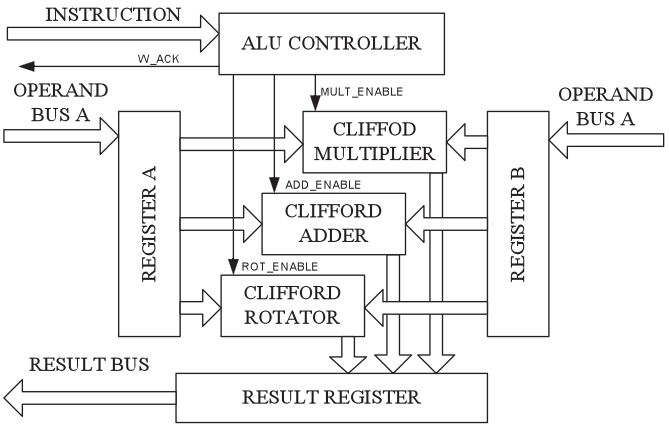

Fig. 4. Clifford ALU internal architecture.

The *bivector* unit is used in the following cases: bivector∗vector, bivector∗trivector, vector∗bivector, and trivector∗bivector; in all considered cases, one operand is formed by six fields (bivector), and the other operand is a four-field homogeneous number (vector or trivector). The result is composed of two homogeneous numbers: a vector (with four fields filled) and a trivector (with four fields filled).

This hardware organization requires appropriate pre-swapping and post-swapping of the input operand elements and the output result elements.

Pre-swapping of input operands is performed before multiplication when either the first operand is a bivector or the first operand is a vector (or a trivector) and the second operand is a scalar (or a pseudoscalar). In fact, in all 24 possible cases, a geometric product involves at most a four-field operand and a six-field operand, hence the use of four 32-bit registers for the first operand and six 32-bit registers for the second one. In the particular case in which the first operand is a bivector (formed by six fields), pre-swapping is performed, storing it to the second six-word register. Pre-swapping is also performed when the second operand is a scalar or pseudoscalar (with only one field filled). In the pre-swapping phase, the first operand is sent to the six-field register, while the first four fields of the second operand are sent to the four-field register. The last two fields of the second operand are discarded since they are empty.

Post-swapping of the output result elements is performed after the sum and route operations in the following cases: vector∗trivector, trivector∗vector, bivector∗pseudoscalar, and pseudoscalar∗bivector. In the post-swapping phase, therefore, the resulting first field is swapped with the sixth one, the second field with the fifth one, and the third field with the fourth one (see Section 3.2.1).

As the results of other product operations (the outer product, left contraction, right contraction) are subsets of the results of the geometric product, they are always performed in CliffoSor by taking the appropriate parts of a geometric product result. Fig. 6(a) shows the dataflow diagram for the *multiplier* functional unit.

4.4.2.3. Adder unit design. The *adder* functional unit performs algebraic sums between two homogeneous elements. If the two homogeneous elements are of the same type, then the result is a homogeneous element of the same type (with fields that are given by a field-to-field sum); otherwise, the result is simply the composition of the two input homogeneous elements. For a subtraction operation, the second operand is sign inverted before the sum

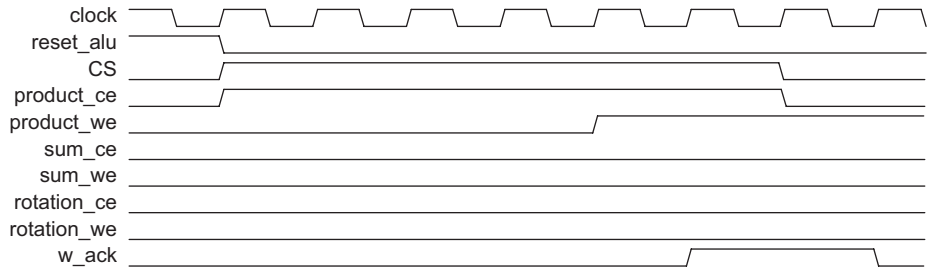

Fig. 5. Timing diagram for a product operation.

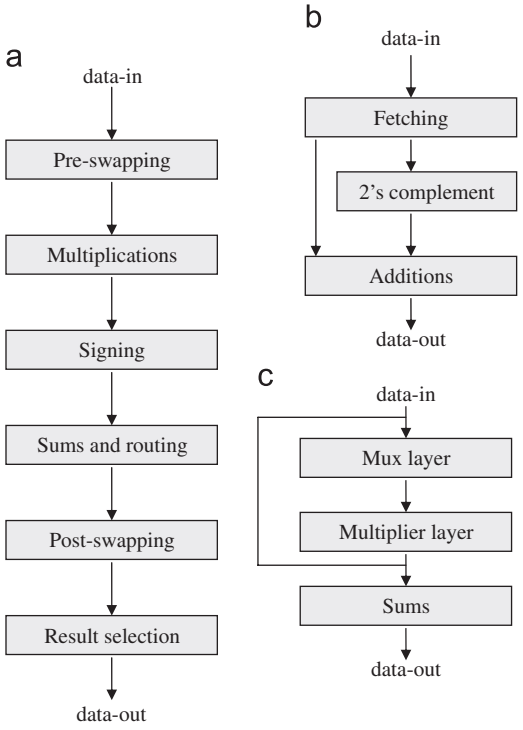

Fig. 6. Dataflow diagram for the three Clifford ALU functional units. (a) Dataflow diagram for the multiplier functional unit. (b) Dataflow diagram for the adder functional unit. (c) Dataflow diagram for the 3D rotation functional unit.

operation. Fig. 6(b) depicts the dataflow diagram for the *adder* unit.

4.4.2.4. 3D rotation unit design. The *3D rotation* functional unit performs 3D vector rotations. The input operands are a 3D vector and a rotor, which is a particular multivector of Clifford algebra formed by a scalar part and a bivector part, as seen in Section 3.3. Since the rotation occurs in three dimensions, the bivector has only three non-null components.

The result element is again a 3D vector. The rotation operation is performed in two phases. In the first one, a collection of coefficients are built from the rotor values in parallel using a bank of multipliers. In the second phase, the result vector components are evaluated as sums of products between input vector components and previous coefficients. These latter multiplications are performed by the same multiplier bank, saving hardware resources. Fig. 6(c) depicts the dataflow diagram for the *rotation* functional unit.

5. Experimental results

Several experimental tests were performed to evaluate the performance of CliffoSor and compare it with existing Clifford algebra software implementations. To this end, the GAIGEN geometric algebra software library generator [9] was used to generate a 4D library.

This library implements 4D homogeneous Clifford algebra on a general-purpose processor, with methods to handle homogeneous numbers (scalars, vectors, bivectors, trivectors, and pseudoscalars) and to perform operations on them.

5.1. Performance tests

Three performance tests of this implementation are presented. In the first test, 500,000 product operations were performed using randomly generated homogeneous numbers. A second test performed 500,000 additions, and a third test executed 500,000 vector rotations. The product operations, sum operations, and vector rotation operations were executed on a traditional general-purpose CPU and on CliffoSor. CliffoSor runs on a FPGA board driven by a 50 Mhz clock, while the GAIGEN software library runs on a 2 GHz Pentium4 CPU. The latencies for each operation were calculated in clock cycles, to suggest the potential speedup. However, the different clock cycle between the highly optimized Pentium4 and the slower-paced CliffoSor do not allow for a direct comparison. By measuring the latencies inside CliffoSor, from when the data is available for reading in SRAM until the results are written back in the SRAM, a *potential* for speedup can be achieved by placing the CliffoSor core on the motherboard, with DMA access to the memory subsystem.

Table 11 lists the clock cycle latencies for products, additions, and 3D rotations. Note that the majority of the total latency is due to cycles spent in the Clifford Interface unit.

Fig. 7 compares the execution of the same product, sum, and 3D rotation operations both in software and using CliffoSor. The full-software implementation uses the highly optimized 4D library generated using GAIGEN. While direct comparison of clock cycles is not meaningful due to the diverse implemented system architecture, the results suggest a potentially significant speedup. The product, sum, and rotation latencies using the GAIGEN-generated library are 264, 640, and 1094 (2 GHz Pentium) clock cycles, respectively. Comparing them with the total CliffoSor clock cycles (at 50 MHz), we see that CliffoSor could potentially achieve $4\times$, $12\times$, and $20\times$ speedups for products, sums, and rotations, respectively.

5.2. Raytracing

To demonstrate the full power of the geometric algebra unifying language, authors Fontjine and Dorst have implemented a full raytracing application using the GAIGEN library generator [16]. Calls to homogeneous number operations were trapped and redirected to the CliffoSor API, to be executed on CliffoSor. Fig. 8 shows both the full-software-rendered scene (a) and the CliffoSor-rendered scene (b). The artifacts in the two sphere surfaces indicate the effects of rounding errors in the CliffoSor execution.

Table 11
CliffoSor Interface and ALU latencies for products, additions, and 3D rotations on random generated homogeneous numbers

Operation	Interface cycles	ALU cycles	Total cycles
Product	49	7	56
Sum, difference	49	5	54
3D rotation	49	7	56

Latencies are expressed in clock cycles at 50 MHz.

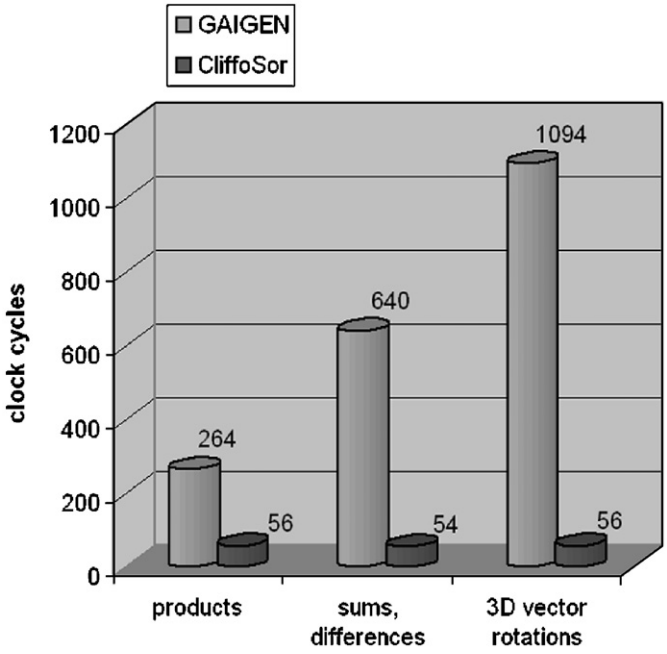

Fig. 7. Product, sum, and 3D rotation operations comparison between full-software implementation on a 2 GHz Pentium and CliffoSor. The comparison is intended for potential speedup estimation only.

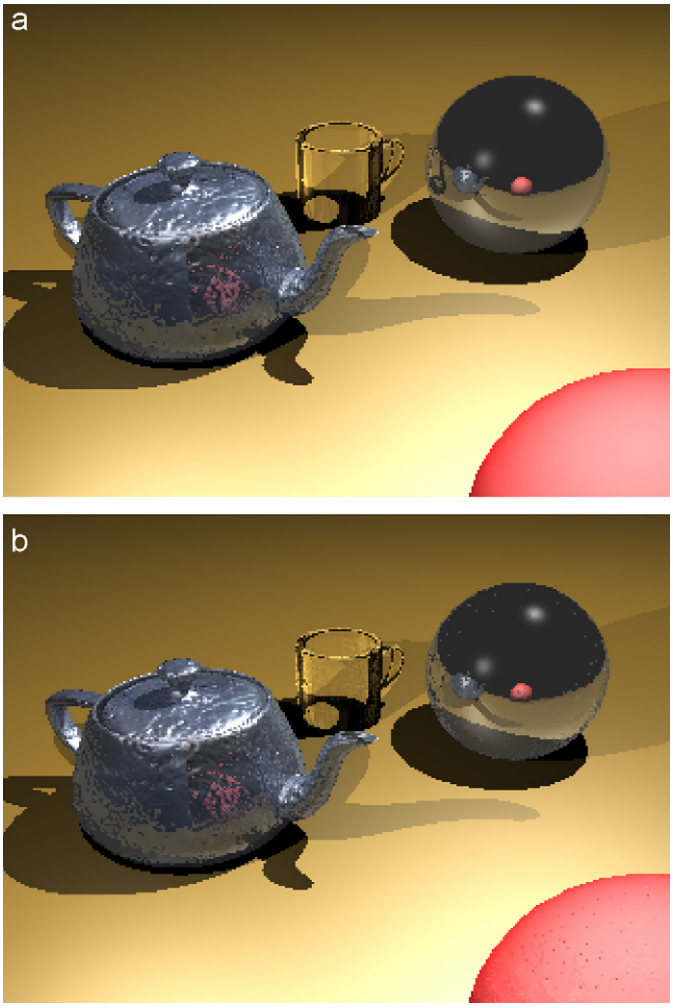

Fig. 8. Sample scene rendered using a GAIGEN-based raytracing application: (a) full-software-rendered scene and (b) CliffoSor-rendered scene.

The raytracer is used to render a 320×240 pixel image with a recursion factor of 5. The application profile exhibits about 2 million *sums/subtractions* between homogeneous numbers and about 11 million *left contractions* between vectors and bivectors. The image is rendered in about 87 s using the full-software raytracer, while the execution time increases to over 2 h once CliffoSor is used. This result was expected as an effect of the very slow interface between the FPGA and the PCI. Accurate profiling of the application shows that most of the time (over 98%) is taken by the PCI handling and data transfer, while the remaining 2% is divided between data preparation (conversion to fixed point and assembly, for 0.7%) on the software side, and CliffoSor operation (between CliffoSor reads and writes to SRAM, for 1.3%).

Interestingly enough, the profiling confirms execution times of 0.1 and 0.149 μs (five cycles and seven cycles at 50 MHz, respectively) for *sums/subtractions* and *left contractions*, respectively, while showing that about 7.1 μs (or 353 cycles at 50 MHz) are spent in the *Clifford Interface* unit. This latter result accounts for both the SRAM control latency discussed earlier, and the Celoxica handshake protocol between RC1000 and PCI bus.

5.3. 3D rotation

3D rotations are executed in real time, and are natively supported on CliffoSor. A sample video is available at the IEEE Manuscript Central, which shows rotation of a 3D model of a teapot with basic rendering.

6. Conclusions

Geometric algebra is a powerful analytical tool that offers an integrated approach to geometric modeling. This paper examined native support of geometric algebra objects and operators directly in hardware. This paper presented the Clifford coprocessor (CliffoSor) system architecture, and a prototype system was implemented on a FPGA board. Test results were presented for 4D geometric products, 4D sums/differences, and 3D rotations on CliffoSor, and compared against the same operations implemented for general-purpose processors by the 4D library generated by the GAIGEN library generator. In addition, a raytracer application and 3D model rotation were successfully implemented and executed on CliffoSor to prove its effectiveness on its application domain.

These results suggest that there is great potential for speedup in comparison with the execution on a traditional processor, but to realize this potential, the core must be placed as close as possible to the memory subsystem, with dedicated DMA access. The potential cycle speedups demonstrated are $4\times$ for products, $12\times$ for sums, and $20\times$ for rotations.

Acknowledgments

The authors would like to thank Dr. Ing. Salvatore Segreto and Dr. Ing. Vincenzo Vullo for their support of the FPGA coprocessor implementation and testing. This work was supported in part by a donation by XILINX, by Project ICT.P04.003 of the Italian National Research Council, and by Programma di Ricerca Ordinario (ex 60%) of the Università degli studi di Palermo, Contract no. ORPA049852, FY 2004 and ORPA051745, FY 2005.

References

[1] S. Mann, N. Litke, T. DeRose, A coordinate-free geometry ADT, University of Waterloo Research Report CS-97-15.
[2] S. Franchini, G. Vassallo, F. Sorbello, A brief introduction to Clifford algebra, Università degli Studi di Palermo Technical Report, 〈http://www.dinfo.unipa.it/files/CliffTechRep.pdf〉.
[3] W.K. Clifford, On the classification of geometric algebras, in: R. Tucker (Ed.), Mathematical Papers, Macmillian, London, 1882, pp. 397–401.
[4] L. Dorst, S. Mann, Geometric algebra: a computational framework for geometrical applications (part 1: algebra), IEEE Comput. Graphics Appl. 22 (4) (2002) 58–67.
[5] L. Dorst, S. Mann, Geometric algebra: a computational framework for geometrical applications (Part 2: applications), IEEE Comput. Graphics Appl. 22 (3) (2002) 24–31.
[6] C. Perwass, The CLU Project web page, 〈http://www.perwass.de/cbup/clu.html〉.
[7] P. Leopardi, The GluCat Home Page, 〈http://glucat.sourceforge.net/〉.
[8] P. Lounesto, The CLICAL Home Page, 〈http://www.helsinki.fi/~lounesto/CLICAL.htm〉.
[9] D. Fontijne, T. Bouma, L. Dorst, GAIGEN: A Geometric Algebra Implementation Generator, University of Amsterdam, Netherlands, 2002.
[10] C. Perwass, C. Gebken, G. Sommer, Implementation of a Clifford algebra coprocessor design on a field-programmable gate array, in: Rafal Ablamowicz (Ed.), Clifford Algebras—Applications to Mathematics, Physics, and Engineering, Progress in Mathematical Physics Series, vol. 34, 2004.
[11] A. Gentile, S. Segreto, F. Sorbello, G. Vassallo, S. Vitabile, V. Vullo, CliffoSor, an innovative FPGA-based architecture for geometric algebra, in: Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA 2005, 2005, pp. 211–217.
[12] A. Gentile, S. Segreto, F. Sorbello, G. Vassallo, S. Vitabile, V. Vullo, CliffoSor: a parallel embedded architecture for geometric algebra and computer graphics, in: Proceedings of the IEEE International Workshop on Computer Architecture for Machine Perception, CAMP 2005, IEEE Computer Society Press, Silver Spring, MD, 2005, pp. 90–95.
[13] D. Hildenbrand, D. Fontijne, C. Perwass, L. Dorst, Geometric algebra and its application to computer graphics, Tutorial 3, in: Interacting with Virtual Worlds, Proceedings of the 25th Annual Conference of the European Association for Computer Graphics, Grenoble, France, INRIA and Eurographics Association, ISSN: 1017-4656.
[14] L. Dorst, The inner products of geometric algebra, in: Dorst, Doran, Lasenby (Eds.), Applications of Geometric Algebra in Computer Science and Engineering, Birkhauser, Basel, 2002, pp. 34–46.
[15] L. Dorst, Honing geometric algebra for its use in the computer sciences, in: G. Sommer (Ed.), Geometric Computing with Clifford Algebra, Springer, Berlin, ISBN 3-540-41198-4, 2001.
[16] D. Fontijne, L. Dorst, Modeling 3D Euclidean geometry, IEEE Comput. Graphics Appl. 23 (2) (2003) 68–78 〈http://www.science.uva.nl/ga/gaigen/〉.
[17] D. Hestenes, New Foundations For Classical Mechanics, Kluwer Academic Publishers, Dordrecht, 1986.
[18] D. Hestenes, G. Sobcyk, Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics, Kluwer Academic Publishers, Dordrecht, 1987.
[19] R. Ablamowicz, Clifford algebra computations with Maple, in: W.E. Baylis (Ed.), Proceedings of the CAP Summer School in Theoretical Physics, Geometric (Clifford) Algebras in Physics, Banff, Alberta, Canada as CLIFFORD (GEOMETRIC) ALGEBRAS with applications in Physics, Mathematics, and Engineering. Birkhauser, Boston, 1996, pp. 463–502.
[20] R. Ablamowicz, B. Fauser, Clifford ver. 5 Home Page, 〈http://math.tntech.edu/rafal/cliff5/index.html〉.
[22] J. Browne, The Grassmann Algebra Book Home Page, 〈http://www.ses.swin.edu.au/homes/browne/grassmannalgebra/book/〉.

Silvia Franchini received the Laurea degree "cum laude" in Electronic Engineering from the University of Palermo, Italy, in 2004. She is a Ph.D. candidate in Computer Engineering at the Dipartimento di Ingegneria Informatica of the University of Palermo, Italy. Her research interests include innovative computer architectures, embedded coprocessors, Clifford algebra and its applications in computer graphics, computer vision, and robotics.

Antonio Gentile received the Laurea degree in electrical engineering and the doctoral degree in computer science from the Universita' degli studi di Palermo, Italy, in 1992 and 1996, respectively. He also received the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology in 2000. He is an associate professor in the Dipartimento di Ingegneria Informatica, Universita' degli studi di Palermo. His research interests include high throughput portable processing systems, image and video processing architectures, embedded systems, speech processing, human–computer interfaces. He is a senior member of the IEEE, of the IEEE Computer Society, and of the AEIT. He is also an associate editor of Integration—the VLSI Journal.

Filippo Sorbello is currently Full Professor in the area of Computer engineering (ING/INF-05 code) with the Dipartimento di Ingegneria Informatica of University of Palermo, Italy. He was the first head of the same Department. He received the "Laurea" degree in Electronic Engineering from the University of Palermo, Palermo, Italy. Since 1970 he has participated in several projects carried out by the institutes and departments in which he was involved. He was a CNR (Italian National Research Council) scholarship holder, regular assistant, and delegate professor. Afterwards from 1982 he was Associate Professor of Computer Science at the University of Palermo and successively Full professor. His research interests are in the field of innovative computer architecture, neural networks applications, real-time image processing, biometric authentication systems, and multi-agent system security. Prof. Sorbello is an IEEE, ACM, AIIA, and AICA member. He is the author of above 120 international scientific papers.

Giorgio Vassallo graduated "cum laude," in Physics at the University of Palermo in 1982. He has worked in the VLSI and Neural Network Lab of CRES (Centro per la Ricerca Elettronica in Sicilia) of Monreale (PA). Currently he is a research scientist at DINFO (Dipartimento di ingegneria INFOrmatica), University of Palermo, Italy. His research interests are in the field of neural networks, geometric techniques for data mining, and natural language processing.

Salvatore Vitabile is an assistant professor with the Department of Medical Biotechnologies and Forensic Medicine, University of Palermo, Italy. He received the Dr. Ing. degree (M.S.E.E.) in Electronic Engineering and the doctoral degree in Computer Science from the University of Palermo in 1994 and 1999, respectively. Dr. Vitabile has participated in several research projects funded by industries and research institutes relative to multi-agent system security, time series analysis and forecasting, and real-time video processing. Dr. Vitabile has co-authored more than 70 scientific papers in referred journals and conferences. He has served on organizing and program committees of international conferences, symposia, and workshops and he is a reviewer for several scientific journals. He is a member of the Board of Directors of SIREN (Italian Society of Neural Networks), of IEEE, and IEEE Engineering in Medicine and Biology Society. His research interests include neural network applications, real-time image and video processing, application-specific processors design and prototyping, biometric authentication systems, multi-agent system security, and medical image processing.