11

Rapid Prototyping of Quaternion Multiplier: From Matrix Notation to FPGA-Based Circuits

Marek Parfieniuk[1], Nikolai A. Petrovsky[2] and Alexander A. Petrovsky[3]
[1]*Bialystok University of Technology*
[2]*Belarusian State University of Informatics and Radioelectronics*
[3]*Bialystok University of Technology*
[1,3]*Poland*
[2]*Belarus*

1. Introduction

In recent years, hypercomplex numbers called quaternions attract attention of many researchers of the fields of digital signal processing (DSP), control, computer graphics, telecommunications, and others. By using hypercomplex arithmetic, known algorithms can be improved or extended to 4 dimensions so as to find new applications (Alexiadis & Sergiadis, 2009; Chan et al., 2008; Denis et al., 2007; Ell & Sangwine, 2007; Karney, 2007; Marion et al., 2010; Parfieniuk & Petrovsky, 2010a; Seberry et al., 2008; Took & Mandic, 2010; Tsui et al., 2008; Zhou et al., 2007). Current research is mainly focused on theoretical development of quaternion-based algorithms, but one can expect that, in the course of time, engineers and scientists will implement them in hardware, and thus will need building blocks, design insights, methodologies, and tools.

In known algorithms that use hypercomplex arithmetic, the key operation is quaternion multiplication, whose efficiency and accuracy obviously determines the same properties of the whole computational scheme of a filter or transform. Even though the operation has been thoroughly investigated from a mathematical point of view (Howell & Lafon, 1975), rather little is known about the practical aspects of implementing it in hardware as a dedicated digital circuit. To the best of our knowledge, only two research groups reported development of fixed-point quaternion multipliers. In (Delosme & Hsiao, 1990; Hsiao & Delosme, 1996; Hsiao et al., 2000; Parfieniuk & Petrovsky, 2010b; Petrovsky et al., 2001; Verenik et al., 2007), they considered various approaches to computing constant-coefficient multiplication using only binary shifts and additions: CORDIC, lifting, and Distributed Arithmetic (DA), but there is no review of the developed computational schemes, which would allow them to be compared and would inspire further research.

The present chapter has two aims. The first one is to briefly review known facts and achievements related to quaternion multipliers and, by presenting a novel CORDIC-Inside-Lifting architecture, to show that there is much to do in this field. The second aim is to present our methodology and design results related to rapid prototyping of different multiplier schemes using a Xilinx Virtex FPGA device. The chapter contents should be useful to persons interested in implementing hypercomplex computations, as it should allow readers

to select the architecture that is best suited to their needs and to prototype hardware faster than working from scratch.

The rest of the chapter is organized as follows. In Section 2, the properties of quaternion multiplication are briefly reviewed with particular emphasis on its matrix notation and noncommutativity. Section 3 presents possible architectures for multiplying a hypercomplex variable by a constant coefficient using only binary shifts and additions: DA, CORDIC approach, lifting scheme, and a novel combination of the last two schemes. Their general principles are discussed and design trade-offs they offer are considered so as to form a basis for developing digital circuits. Section 4 begins by presenting our rapid prototyping methodology, development platform and tools so as to show how we use Matlab and VHDL-FPGA tools to develop multiplier units. Then, finally, design experiments are reported for different architectures of quaternion multipliers.

Throughout this chapter, the following notation is used. Matrices and column vectors are denoted by upper- and lower-case bold faced characters, respectively. The superscript T denotes transposition. $\mathbf{I}_2$ and $\mathbf{J}_2$ denote the 2×2 identity and reversal matrices, respectively. $\mathbf{\Gamma}_2 = \mathrm{diag}\,(1, -1)$. The set of natural numbers is denoted by $\mathbb{N}$ and is assumed to include zero.

2. Quaternion multiplication

Quaternions are four-dimensional hypercomplex numbers whose rectangular form comprises a real part and three imaginary parts:

$$q = q_1 + q_2 i + q_3 j + q_4 k, \quad q_1, q_2, q_3, q_4 \in \mathbb{R} \tag{1}$$

the latter of which are related to three imaginary units, i, j, and k. Because these units depend on each other in the following way

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2a}$$

$$ij = -ji = k \tag{2b}$$

$$jk = -kj = i \tag{2c}$$

$$ki = -ik = j \tag{2d}$$

quaternion multiplication is noncommutative, unless one of its operands is a real number or both are complex numbers.

The noncommutativity of quaternion multiplication becomes evident when hypercomplex numbers are identified with vectors, and the operation is written in matrix notation:

$$qx \quad \Leftrightarrow \quad \underbrace{\begin{bmatrix} q_1 & -q_2 & -q_3 & -q_4 \\ q_2 & q_1 & -q_4 & q_3 \\ q_3 & q_4 & q_1 & -q_2 \\ q_4 & -q_3 & q_2 & q_1 \end{bmatrix}}_{\mathbf{M}^+(q)} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{x}} \tag{3a}$$

but

$$xq \quad \Leftrightarrow \quad \underbrace{\begin{bmatrix} q_1 & -q_2 & -q_3 & -q_4 \\ q_2 & q_1 & q_4 & -q_3 \\ q_3 & -q_4 & q_1 & q_2 \\ q_4 & q_3 & -q_2 & q_1 \end{bmatrix}}_{\mathbf{M}^-(q)} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{x}} \tag{3b}$$

Namely, the left-operand multiplication matrix, $\mathbf{M}^+(\cdot)$, differs from the right-operand one, $\mathbf{M}^-(\cdot)$.

It should be emphasized that our attention is focused on multiplications in which x is an input variable, and q is a constant coefficient. The latter determines the multiplication matrix and thus the linear transformation of input data that has to be implemented in hardware. It is also assumed that both operands are unit-norm quaternions, i.e. $|q| = \sqrt{q_1 + q_2 + q_3 + q_4} = 1$.

It is easy to show (Parfieniuk & Petrovsky, 2010b) that the multiplication matrices are related in the following way

$$\mathbf{M}^{\mp}(q) = \mathbf{D}_{\mathrm{C}} \mathbf{M}^{\pm}(\overline{q}) \mathbf{D}_{\mathrm{C}} \tag{4}$$

where $\overline{q} = q_1 - q_2 i - q_3 j - q_4 k$ denotes the conjugate quaternion, and $\mathbf{D}_{\mathrm{C}} = \mathrm{diag}(1, -\mathbf{I}_3)$ is the matrix representation of the hypercomplex conjugate operator, as $\overline{\mathbf{q}} = \mathbf{D}_{\mathrm{C}}\mathbf{q}$. Thus a multiplier that computes one variant of quaternion multiplication can be used to compute the other variant. It is only necessary to change the signs of some components of the input, output, and coefficient. From a complexity point of view, sign changes can be neglected, and thus both variants can be considered equivalent.

3. Multiplierless circuits for multiplying quaternions

3.1 Motivation

The straightforward approach to compute quaternion products is to use (3) without exploiting the special structures of the multiplication matrices. It obviously requires doing 16 multiplications and 12 additions of real numbers, but its actual performance highly depends on whether and how vector processing is supported by the hardware and used in implementation. In particular, quaternion computations can be accelerated on PC processors with SIMD extensions (Leiterman, 2003).

On the other hand, for FPGA and ASIC, it is often not desirable, or even possible, to implement real multipliers, because they occupy too much chip area and consume too much energy, offering unsatisfactory throughput. The preferred approach is to replace constant-coefficient multipliers with simple shifts and additions, the former of which can be hardwired.

There are several different approaches to implement quaternion multiplication using only binary shifts and additions. In the next three subsections, they are described from a conceptual point of view, whereas practical design results and insights are presented in Section 4.

3.2 Distributed arithmetic

3.2.1 General principle

Distributed Arithmetic (DA) is a technique that allows vector inner products (sums of products) to be computed without multiplications (White, 1989). The fundamental assumption is that one vector comprises fixed coefficients, whereas the elements of the other one are variables. The corresponding bits of the variables are used as an address into a read-only memory (ROM) that contains precomputed partial results: all possible

combinations of elements of the coefficient vector. The multiplication result is computed by accumulating all results pointed to by given values of the variables.
The matrix-by-vector product (3), which describes quaternion multiplication, corresponds to four inner-products, each of which can be realized using DA (Petrovsky et al., 2001; Verenik et al., 2007). The inner products are related to different constant vectors, which correspond to rows of the multiplications matrix, but use the same variable vector.
Assuming that variable data are fractions, $|x_k| \leq 1$, and are represented using signed 2's-complement with a word length of B bits, the components of $\mathbf{x}$ can be expressed as:

$$x_k = -x_{k,0} + \sum_{n=1}^{B-1} x_{k,n} 2^{-n}, \qquad k = 1, \ldots, 4 \tag{5}$$

where $x_{k,n}$ are consecutive bits: from the sign bit, $x_{k,0}$, to least significant bit (LSB), $x_{k,B-1}$.
Such expansion allows the components of the result $r = qx$ to be expressed in terms of bits of x. In particular,

$$\begin{aligned} r_1 &= q_1 x_1 - q_2 x_2 - q_3 x_3 - q_4 x_4 \\ &= q_1 \sum_{n=0}^{B-1} \sigma(n) x_{1,n} 2^{-n} - q_2 \sum_{n=0}^{B-1} \sigma(n) x_{2,n} 2^{-n} - q_3 \sum_{n=0}^{B-1} \sigma(n) x_{3,n} 2^{-n} - q_4 \sum_{n=0}^{B-1} \sigma(n) x_{4,n} 2^{-n} \end{aligned} \tag{6}$$

where

$$\sigma(n) = \begin{cases} -1, & \text{for } n = 0 \\ 1, & \text{otherwise} \end{cases} \tag{7}$$

By reordering additions and introducing

$$\tilde{r}_1 (\alpha, \beta_1, \beta_2, \beta_3, \beta_4) = \alpha (q_1 \beta_1 - q_2 \beta_2 - q_3 \beta_3 - q_4 \beta_4) \tag{8}$$

we can obtain the following:

$$r_1 = \sum_{n=0}^{B-1} \tilde{r}_1 (\sigma(n), x_{1,n}, x_{2,n}, x_{3,n}, x_{4,n}) 2^{-n} \tag{9}$$

The derivations for the remaining three inner products are similar so they can be omitted.
Because every bit $x_{k,n}$ can be either 0 or 1, $\tilde{r}_1 (\sigma(n), x_{1,n}, x_{2,n}, x_{3,n}, x_{4,n})$, $n = 0, \ldots, B-1$ can take only $2^4 = 16$ possible magnitudes, so its values for $\sigma(n) = 1$ can be precomputed and stored in a memory. Then, during computation of a quaternion product, $x_{1,n}, x_{2,n}, x_{3,n}$, and $x_{4,n}$ can be used as an address into the table, from where partial results are read. After accumulating B appropriately shifted partial results, the final result is obtained.
The corresponding scheme of a DA-based quaternion multiplier is shown in Fig. 1, where the "P/S" block is a parallel-to-serial converter. The sign flag $\sigma(n)$ is represented by an additional binary control signal, which switches the accumulators between the addition and subtraction modes, so that the first partial result is subtracted whereas the remaining ones are added. A less common but possible approach is to use this signal as a supplementary address into a memory extended so as to contain values for $\sigma(n) = -1$ (negated copies).
In FPGA devices, the ROM can be implemented easily and efficiently using look-up tables (LUTs), which are the basic building block, or using external memory banks. Thus different design trade-offs are possible so as to utilize hardware features optimally.

It is noteworthy that the DA-based multiplier is composed of four subblocks of the same structure, which differ only in ROM contents. The special structure of the multiplication matrix is not exploited.

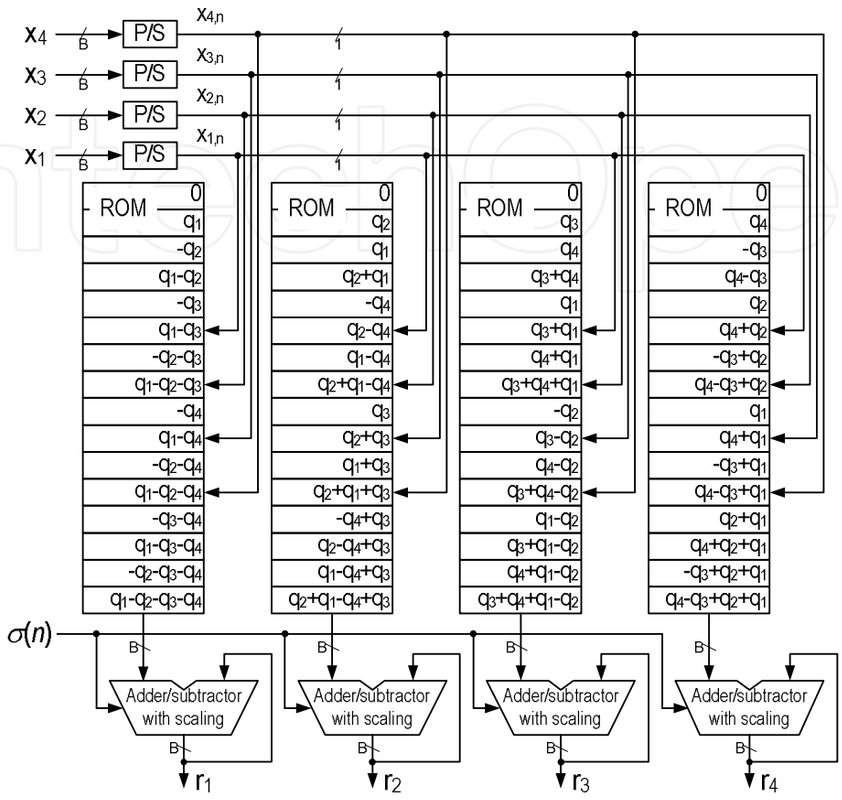

Fig. 1. General scheme of DA-based quaternion multiplier.

3.2.2 Reducing memory requirements

A disadvantage of DA is that LUTs can occupy a lot of memory. Even though the problem concerns mainly inner products of lengthy vectors, which is typical for FIR filters, it can be important in case of implementing algorithms that use a number of quaternion multiplications.

The necessary memory can be halved by using the offset-binary coding (OBC) (White, 1989). Its derivation begins by rewriting (5) as

$$x_k = \frac{1}{2}\left(x_k - (-x_k)\right) = \frac{1}{2}\left(-(x_{k,0} - \bar{x}_{k,0}) + \sum_{n=1}^{B-1}(x_{k,n} - \bar{x}_{k,n})2^{-n} - 2^{-(B-1)}\right) \tag{10}$$

where the overbars represent the bit complements. If we define

$$\rho(x_{k,n}) = x_{k,n} - \bar{x}_{k,n} \tag{11}$$

which can be -1 or $+1$, then (10) can be rewritten as

$$x_k = \frac{1}{2}\left(\sum_{n=0}^{B-1}\sigma(n)\rho(x_{k,n})2^{-n} - 2^{-(B-1)}\right) \tag{12}$$

Using (12), (6) can be rewritten as

$$\begin{aligned} r_1 = q_1 \sum_{n=0}^{B-1} \left(\frac{\sigma(n)}{2} \rho(x_{1,n}) 2^{-n} - 2^{-B} \right) - q_2 \sum_{n=0}^{B-1} \left(\frac{\sigma(n)}{2} \rho(x_{2,n}) 2^{-n} - 2^{-B} \right) \\ - q_3 \sum_{n=0}^{B-1} \left(\frac{\sigma(n)}{2} \rho(x_{3,n}) 2^{-n} - 2^{-B} \right) - q_4 \sum_{n=0}^{B-1} \left(\frac{\sigma(n)}{2} \rho(x_{4,n}) 2^{-n} - 2^{-B} \right) \end{aligned} \tag{13}$$

Then, reordering terms and substituting (8) give

$$r_1 = \sum_{n=0}^{B-1} \tilde{r}_1 \left(\frac{\sigma(n)}{2}, \rho(x_{1,n}), \rho(x_{2,n}), \rho(x_{3,n}), \rho(x_{4,n}) \right) 2^{-n} + \tilde{r}_1 \left(-2^B, 1, 1, 1, 1 \right) \tag{14}$$

Obviously, the DA principle can be realized by using $\tilde{r}_1(\frac{\sigma(n)}{2}, \rho(x_{1,n}), \rho(x_{2,n}), \rho(x_{3,n}), \rho(x_{4,n}))$ as precomputed partial results and using $\rho(x_{1,n}), \ldots, \rho(x_{4,n})$ to address look-up-tables. However,

$$\begin{aligned} \tilde{r}_1 \left(\frac{\sigma(n)}{2}, \rho(x_{1,n}), \rho(x_{2,n}), \rho(x_{3,n}), \rho(x_{4,n}) \right) \\ = -\tilde{r}_1 \left(\frac{\sigma(n)}{2}, -\rho(x_{1,n}), -\rho(x_{2,n}), -\rho(x_{3,n}), -\rho(x_{4,n}) \right) \end{aligned} \tag{15}$$

and thus only half of 16 possible values of $\tilde{r}_1(\frac{\sigma(n)}{2}, \rho(x_{1,n}), \rho(x_{2,n}), \rho(x_{3,n}), \rho(x_{4,n}))$ need to be be stored in ROM.

In exchange, LUT addressing must be quite tricky, accumulators need to be switched between the addition and subtraction modes, as in the scheme in Fig. 2. Namely, the partial result that corresponds to the n-th bit of input data is selected by using $x_{2,n}$, $x_{3,n}$, and $x_{4,n}$, whereas $x_{1,n}$ decides whether it is added to the accumulator or subtracted. It is also noteworthy that accumulators are initialized with data from outside LUTs: see the term on the right-hand side of (14). This is why the timing diagram in Fig. 2 shows that 17 cycles of the master clock (Clk) are necessary to multiply 16-bit values.

3.2.3 Increasing the speed of DA-based quaternion multiplier

The speed of the straightforward bit-serial implementation of DA may be insufficient for certain real-time applications. In such cases, faster circuits can be developed in which subwords of input data are processed in parallel using more memory and adders (White, 1989). Namely, L bits can be processed in a single clock period using L LUTs and a L-input accumulator that combines their outputs. Thus, the whole input word is processed in B/L clock cycles, assuming that L is an integer divisor of B, its word length.

The possibility of this becomes obvious after rewriting (14) in the following way:

$$\begin{aligned} r_1 = \sum_{m=0}^{B/L-1} \sum_{l=0}^{L-1} \tilde{r}_1 \left(\frac{\sigma(n)}{2} 2^{-l}, \rho(x_{1,mL+l}), \rho(x_{2,mL+l}), \rho(x_{3,mL+l}), \rho(x_{4,mL+1}) \right) 2^{-mL} \\ + \tilde{r}_1 \left(-2^{-B}, 1, 1, 1, 1 \right) \end{aligned} \tag{16}$$

The summation over n in (14) is parallelized by breaking it into two sums: the first over m, from 0 to $B/L-1$, and the second over l, from 0 to $L-1$. The former sum is computed iteratively, whereas the latter sum is realized as a single step using replicated hardware.
A circuit that processes in such a way L bits at a time is called the L-BAAT scheme. Figures 2 and 3 allow for comparing 1- and 2-BAAT schemes we have developed. In the latter case, it is necessary to use two adders instead of one, and to shift the accumulator contents by 2 bits instead of by 1 bit. Additionally, the word length of the second memory block is shorter by 1 bit.

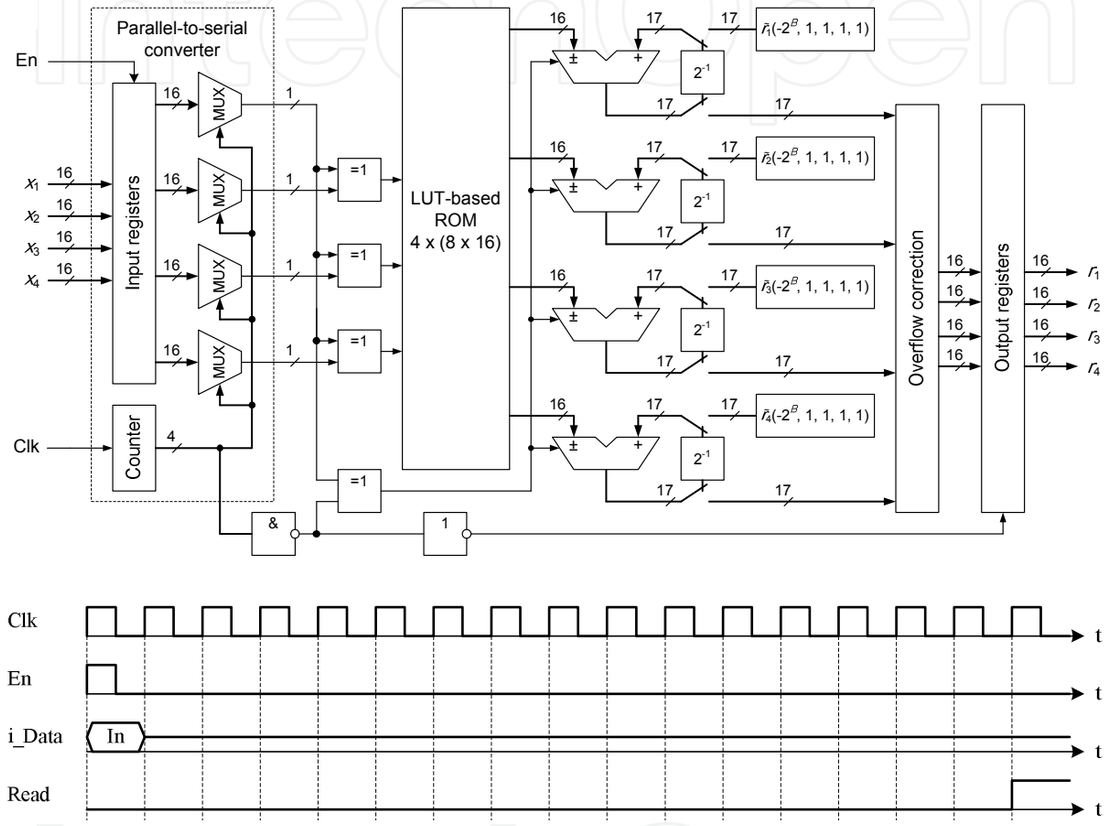

Fig. 2. 1-BAAT DA-based quaternion multiplier and its timing diagram.

3.3 CORDIC

3.3.1 Implementing plane rotations

CORDIC (COordinate Rotation DIgital Computer) is a hardware-oriented algorithm for computing plane (2D) rotations as well as related elementary functions and transforms (Meher et al., 2009). The algorithm consists in approximating a rotation matrix by a product of discrete elementary rotations, each of which is implemented using only binary shifts and additions.

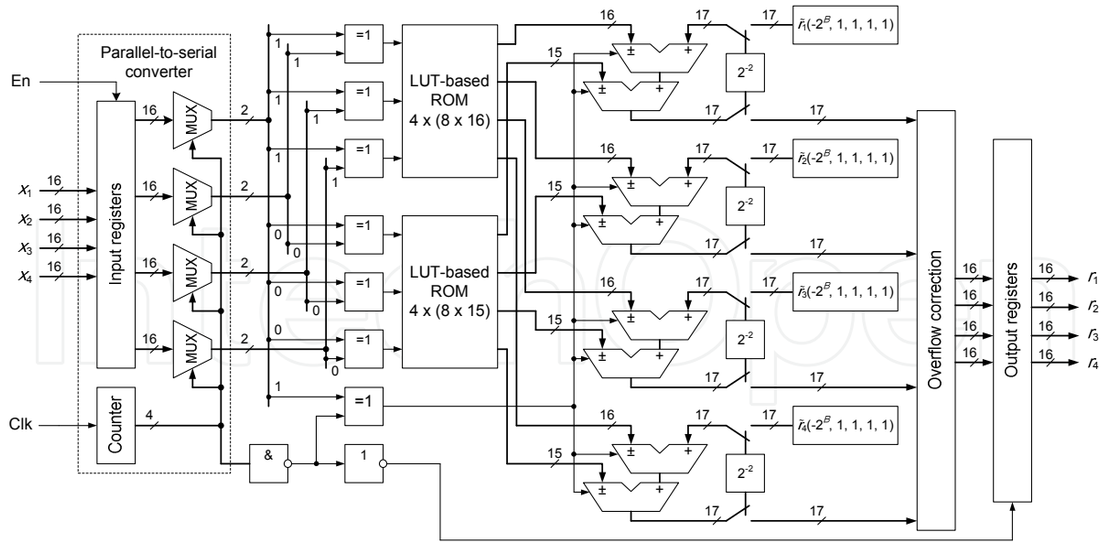

Fig. 3. 2-BAAT DA-based quaternion multiplier.

The rotation of a point (x_1, x_2) by an angle ϕ can be described as the multiplication of the corresponding coordinate vector $\mathbf{x}$ by the appropriate rotation matrix $\mathbf{R}(\phi)$

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}}_{\mathbf{R}(\phi)} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathbf{x}} \tag{17}$$

From another point of view, this equation describes the complex multiplication of $x_1 + jx_2$ by $e^{j\phi}$.

The CORDIC algorithm is based on factorizing the rotation matrix in the following way

$$\mathbf{R}(\phi) \approx S_{\text{tot}} \prod_{n=0}^{N-1} \begin{bmatrix} 1 & -\sigma(n)2^{-\tau(n)} \\ \sigma(n)2^{-\tau(n)} & 1 \end{bmatrix} \tag{18}$$

where N, $\sigma(n)$, and $\tau(n)$ are selected in such a way that the product approximates a scaled rotation by ϕ, and simultaneously the matrix on the right-hand side describes a transformation that can be implemented using two binary shifts and two additions.

The transformation is called a CORDIC iteration, elementary rotation, or microrotation. It not only rotates the point but also increases its norm by $\sqrt{1 + 2^{-2\tau(n)}}$. In order to counterbalance the latter effect, the output of a sequence of microrotations must be scaled by

$$S_{\text{tot}} = \prod_{n=0}^{N-1} \frac{1}{\sqrt{1 + 2^{-2\tau(n)}}} \tag{19}$$

which can also be done by using only binary shifts and additions, which form a series of scaling iterations

$$S_{\text{tot}} \approx \prod_{m=0}^{M-1} 1 - 2^{-s(m)} \tag{20}$$

where $s(m) \in \mathbb{N}\backslash\{0\}$ are selected so as the product approximates S_{tot}.
In the classical form of CORDIC,

$$\sigma(n) \in \{-1, 1\} \tag{21}$$

$$\tau(n) = n \tag{22}$$

and

$$N = B \tag{23}$$

This makes scaling independent of the angle, which is advantageous when angle determination is a part of a CORDIC-based algorithm. If the angle is known a-priori, as in case of a constant-coefficient complex multiplication, then by making $\tau(n) \in \mathbb{N}$ independent of n, the same accuracy can be obtained using less microrotations.

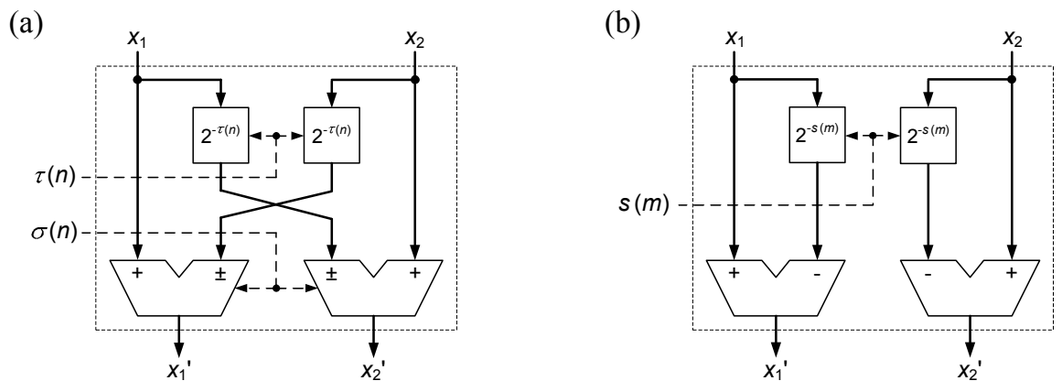

Fig. 4. Schemes for realizing (a) microrotation and (b) scaling iteration of the 2D-CORDIC.

Figure 4 shows the schemes that realize (a) microrotations and (b) scaling iterations of the 2D-CORDIC. If chip area must be saved, a single switched CORDIC unit can be developed, which is able to realize different microrotations. At the price of occupying more chip area, throughput can be increased by unfolding computations, i.e. by pipelining many units, each of which computes a different microrotation. We are interested mainly in the unfolded architecture, which can be optimized by hardwiring fixed binary shifts.
It is also noteworthy that (18) applies only to rotations (angles) for which $\cos(\phi) \geq 0$ and $|\cos(\phi)| \geq |\sin(\phi)|$. Fortunately, the rotation by an unsuitable angle ϕ can always be realized by simply pre- and post-processing the rotation by a tightly connected suitable angle $\tilde{\phi}$:

$$\mathbf{R}(\phi) = \mathbf{P}_{\text{post}}\mathbf{R}(\tilde{\phi})\mathbf{P}_{\text{pre}} \tag{24}$$

In particular,

$$\mathbf{R}(\phi) = \mathbf{\Gamma}_2\mathbf{R}(-\phi)\mathbf{\Gamma}_2 \tag{25}$$

$$\mathbf{R}(\phi) = \mathbf{J}_2\mathbf{R}(\tfrac{\pi}{2} - \phi)\mathbf{\Gamma}_2 \tag{26}$$

$$\mathbf{R}(\phi) = -\mathbf{J}_2\mathbf{R}(\pi - \phi)\mathbf{J}_2 \tag{27}$$

so it is sufficient to swap values and/or to change their signs.
It is also important for us that arbitrary $S_{\text{tot}} < 1$ can be approximated using (20).

3.3.2 4D CORDIC

In (Delosme & Hsiao, 1990), it has been shown that the CORDIC algorithm can be extended to more dimensions and used to implement quaternion multiplications, which consists in the following factorization

$$\mathbf{M}^{\pm}(q) \approx S_{\text{tot}} \prod_{n=0}^{N-1} \begin{bmatrix} 1 & \sigma_1(n)2^{-\tau(n)} & \sigma_2(n)2^{-\tau(n)} & \sigma_3(n)2^{-\tau(n)} \\ -\sigma_1(n)2^{-\tau(n)} & 1 & \pm\sigma_3(n)2^{-\tau(n)} & \mp\sigma_2(n)2^{-\tau(n)} \\ -\sigma_2(n)2^{-\tau(n)} & \mp\sigma_3(n)2^{-\tau(n)} & 1 & \pm\sigma_1(n)2^{-\tau(n)} \\ -\sigma_3(n)2^{-\tau(n)} & \pm\sigma_2(n)2^{-\tau(n)} & \mp\sigma_1(n)2^{-\tau(n)} & 1 \end{bmatrix} \tag{28}$$

Setting the parameters in the conventional way, in accordance with (21) – (23), makes scaling independent of q, as

$$S_{\text{tot}} = \prod_{n=0}^{N-1} \frac{1}{\sqrt{1+3\cdot 2^{-2\tau(n)}}} \tag{29}$$

Four-dimensional CORDIC iterations need to be implemented using four-argument adders, as shown in Fig. 5.

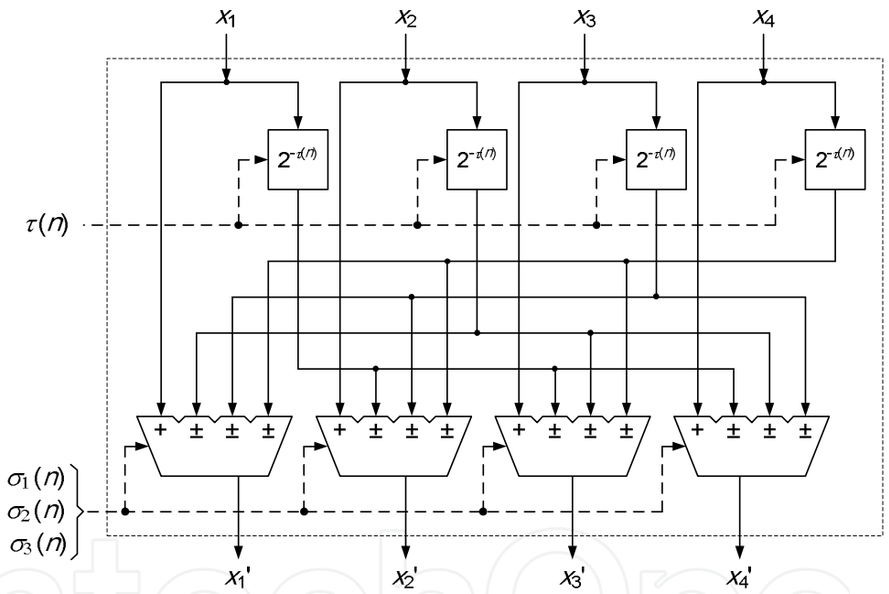

Fig. 5. Microrotation implementation for 4D-CORDIC-based quaternion multipliers.

3.4 CORDIC-Inside-Lifting

3.4.1 Lifting-based quaternion multiplier

Due to word length limitations, results of binary shifts and additions must be truncated. This causes both DA- and CORDIC-based quaternion multipliers to realize data transformations that cannot be reversed in fixed-point arithmetics. Namely, the multiplication by the inverse of the coefficient, or simply by its conjugate in case of a unit-norm coefficient, gives only an approximation of the input value. This is unacceptable for applications like lossless image coding.

The issue can be solved by using lifting schemes, which allow reversible integer-to-integer mappings to be realized using finite-precision arithmetic (Calderbank et al., 1998). Similarly

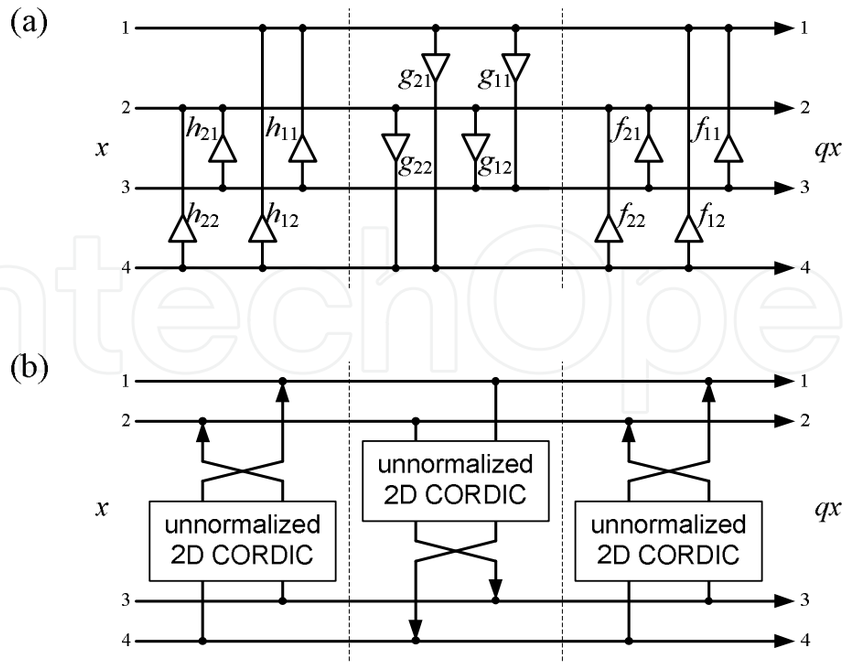

Fig. 6. Lifting-based quaternion multiplier: (a) direct scheme and (b) lifting steps implemented using 2D CORDIC.

to CORDIC, the idea has been introduced in the context of 2D transformations, but we have extended it to quaternions in (Parfieniuk & Petrovsky, 2010b). Our idea is based on the structural similarity between $\mathbf{R}(\phi)$ in (18) and $\mathbf{M}^{+}(q)$ in (3a), which becomes evident after rewriting the latter matrix in the following way

$$\mathbf{M}^{+}(q) = \begin{bmatrix} \mathbf{C}(q) & -\mathbf{S}(q) \\ \mathbf{S}(q) & \mathbf{C}(q) \end{bmatrix} \tag{30}$$

where

$$\mathbf{C}(q) = \begin{bmatrix} q_1 & -q_2 \\ q_2 & q_1 \end{bmatrix} \tag{31}$$

and

$$\mathbf{S}(q) = \begin{bmatrix} q_3 & q_4 \\ q_4 & -q_3 \end{bmatrix} \tag{32}$$

This allows for assuming the following factorization

$$\mathbf{M}^{+}(q) = \underbrace{\begin{bmatrix} \mathbf{I}_2 & \mathbf{F}(q) \\ \mathbf{0} & \mathbf{I}_2 \end{bmatrix}}_{\mathbf{U}(q)} \underbrace{\begin{bmatrix} \mathbf{I}_2 & \mathbf{0} \\ \mathbf{G}(q) & \mathbf{I}_2 \end{bmatrix}}_{\mathbf{L}(q)} \underbrace{\begin{bmatrix} \mathbf{I}_2 & \mathbf{H}(q) \\ \mathbf{0} & \mathbf{I}_2 \end{bmatrix}}_{\mathbf{V}(q)} \tag{33}$$

which is inspired by the known three-shear factorization of the 2D rotation matrix (Daubechies & Sweldens, 1998). The corresponding scheme is shown in Fig. 6(a).

For a given hypercomplex coefficient q, (33) defines a set of matrix equations, which can be solved uniquely for $\mathbf{F}(q)$, $\mathbf{G}(q)$, and $\mathbf{H}(q)$, provided that $\mathbf{S}(q)$ is non-singular, or more specifically, non-zero. Namely, the three matrices are given by

$$\mathbf{F}(q) = (\mathbf{C}(q) - \mathbf{I}_2)\,\mathbf{S}(q)^{-1} \tag{34a}$$

$$\mathbf{G}(q) = \mathbf{S}(q) \tag{34b}$$

$$\mathbf{H}(q) = \mathbf{S}(q)^{-1}\,(\mathbf{C}(q) - \mathbf{I}_2) \tag{34c}$$

so that their elements represent real-valued lifting coefficients.
Inverting the triangular matrices in (33) requires only changing the signs of their out-of-diagonal elements (lifting coefficients). Thus, the multiplication by $1/q$, or equivalently by $\overline{q}$, is realized by reversing the order of the lifting steps and negating their coefficients.
Rounding of neither lifting coefficient nor the result of the related multiplication affects invertibility, even though it causes slight deviation of the stage output from that of the original. Fortunately, the issue is acceptable for most applications.
In addition to making reversible integer-to-integer mappings possible in fixed-point arithmetic, an inherent property of lifting schemes is that all computations can be performed in-place, i.e. without using auxiliary memory.
Another advantage of the proposed lifting-based quaternion multiplier is that 12 real multiplications and 12 real additions are necessary to compute the result. This is 14% less than required by the straightforward matrix-by-vector multiplication (3a).

3.4.2 Embedding CORDIC inside lifting

Alternatively, quaternions can be represented in polar form, using modulus $|q|$ and three angles: ϕ, ψ, and χ, which can be converted into the rectangular components as follows:

$$\begin{aligned} q_1 &= |q| \cos\phi \\ q_2 &= |q| \sin\phi \cos\psi \\ q_3 &= |q| \sin\phi \sin\psi \cos\chi \\ q_4 &= |q| \sin\phi \sin\psi \sin\chi \end{aligned} \tag{35}$$

where taking $-\pi \le \phi < \pi$, $-\pi/2 \le \psi \le \pi/2$, and $-\pi/2 \le \chi \le \pi/2$ is sufficient to describe all quaternions of a given norm.
By assuming $|q| = 1$ and substituting (35) into (34), we can derive the following simple closed-form equations for lifting coefficient values (Parfieniuk & Petrovsky, 2010b):

$$\begin{aligned} f_{11}(q) &= -f_{22}(q) = \frac{\cos\chi(\cos\phi - 1) - \sin\phi\cos\psi\sin\chi}{d} \\ f_{12}(q) &= f_{21}(q) = \frac{\sin\chi(\cos\phi - 1) + \sin\phi\cos\psi\cos\chi}{d} \\ g_{11}(q) &= -g_{22}(q) = d\cos\chi \\ g_{12}(q) &= g_{21}(q) = d\sin\chi \\ h_{11}(q) &= -h_{22}(q) = \frac{\cos\chi(\cos\phi - 1) + \sin\phi\cos\psi\sin\chi}{d} \\ h_{12}(q) &= h_{21}(q) = \frac{\sin\chi(\cos\phi - 1) - \sin\phi\cos\psi\cos\chi}{d} \end{aligned} \tag{36}$$

where $d = \sin\phi \sin\psi$.
It turns out that there are pairs of lifting coefficients with the same absolute value, which is not obvious from (34) and was not assumed when we constructed the factorization (33). Moreover, $\mathbf{F}(q)$, $\mathbf{G}(q)$, and $\mathbf{H}(q)$ have a structure closely related to that of the rotation matrix

$$\begin{bmatrix} s & c \\ c & -s \end{bmatrix} = \mathbf{J}_2 \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \tag{37}$$

and thus can be modeled using 2D-CORDIC. Only simple postprocessing is necessary, as explained in Fig. 6(b).
In (Parfieniuk & Petrovsky, 2010b), it is shown that other known lifting factorizations result in computational schemes that are less regular and less efficient. It is also proved that the dynamic range of the proposed scheme can always be limited so as all lifting coefficients have magnitudes not greater than 1. This is achieved by replacing the hypercomplex multiplication by q with the multiplication by $\tilde{q}$, a version of q with exchanged and/or negated parts. The latter operation needs only to be appropriately pre- and postprocessed:

$$\mathbf{M}^{\pm}(q) = \mathbf{P}_{\text{post}} \mathbf{M}^{\pm}(\tilde{q})\, \mathbf{P}_{\text{pre}} \tag{38}$$

4. Rapid prototyping of quaternion multipliers using Xilinx FPGA

4.1 General methodology

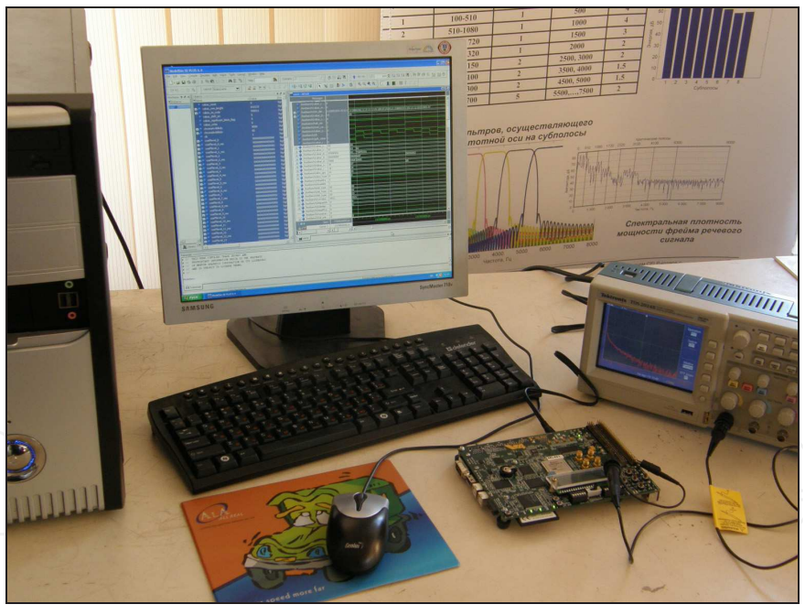

Fig. 7. Development environment based on the evaluation board Xilinx ML401 (Virtex-4 XC4VSX35 FPGA).

The methodology we used for rapid prototyping of quaternion multipliers can be described as the following sequence of steps:

1. Investigating the general idea of a multiplier architecture using rough MATLAB scripts: validating a decomposition of the quaternion product and related matrix factorizations, and then evaluating or determining properties of the corresponding computational scheme.

2. Refactoring and extending MATLAB code so as to prepare reusable functions and scripts, which can be used in different contexts: deeper investigations, low-level coefficient optimization, simulation-based functional verification of hardware designs etc.
3. Repeating Steps 1 and 2 for promising variants of the essential architecture so as to select variants that deserve a deeper investigation.
4. For each variant, determining circuit parameters and synthesizing low-level coefficients, partial results etc. that allow a given constant-coefficient hypercomplex multiplication to be computed with the necessary accuracy.
5. Rejecting parameter-coefficient sets, and possibly variants, which are definitely no match for the others in terms of circuit complexity.
6. Developing detailed MATLAB models of the acceptable hardware multipliers.
7. Initial hardware design using MATLAB code as a reference: VHDL coding and logic synthesis using an FPGA development environment, employing the most advanced of available predefined subcircuits, and generously selecting word lengths.
8. Functional verification of the preliminary designs using FPGA tools and MATLAB scripts. Refining the latter if necessary.
9. Fine-tuning word lengths and subcircuits so as to minimize chip area and maximize throughput of the multipliers.
10. A final comparison of the developed circuits so as to determine that which is best suited to a given application context.

As our goal was to develop optimized circuits and to investigate their properties, and code generators are known to produce highly suboptimal implementations (Haldar et al., 2001), VHDL code was hand-written. This slows down development but not much as quaternion multipliers are rather simple systems, compared to filters, transforms, etc. Additionally, manual transition from MATLAB to VHDL can be made easier by partitioning MATLAB code so as it reflects a circuit architecture.

Rather modest resources are necessary to implement our methodology, as shown in Fig. 7. Apart form MATLAB, the Leonardo Spectrum and Xilinx ISE software were used in our research. On the other hand, experiments were conducted on the Virtex-4 ML 401 Evaluation Platform, which is powered by the Xilinx XC4VLX25 FPGA device and equipped with a range of memories, industry-standard peripherals, interfaces, and connectors like USB or VGA. Thus the board allows for both studying quaternion multipliers and employing them in practical embedded systems like an image codec.

4.2 DA-based circuits

In DA-based multipliers, the output error is essentially determined by the word length and independent of implementation details. Thus we focus on how parallelization of the computations affects throughput, chip area and power consumption.

In order to evaluate the available design opportunities, five 16-bit DA-based quaternion multipliers have been prototyped with the parallelism ratio from 1 to 16. The average synthesis results for the Xilinx Virtex v400-4 FPGA are listed in Table 1.

Figures 8 and 9 show the power consumption of DA-based quaternion multipliers as a function of the parallelism ratio and required sampling time, respectively. The analysis of the results shows that the 1-BAAT scheme of the multiplier occupies the smallest chip area, whereas the 8-BAAT scheme consumes the smallest power for a required sampling time. The

explanation for this is that, for the same required sampling time, the 1-BAAT scheme has a master clock frequency greater than that of the 8-BAAT scheme.

Scheme	Multiplication time [μs]	Frequency [MHz]	No. of Function Generators [LUTs]	No. of CLB Slices	No. of Dffs or Latches
1-BAAT	0.292598967	58,1	219	110	205
2-BAAT	0.160714286	56,0	339	170	207
4-BAAT	0.123152709	40,6	561	281	212
8-BAAT	0.100671141	29,8	958	479	223
16-BAAT	0.107526882	18,6	1637	819	246

Table 1. Parameters of DA-based quaternion multipliers with different values of the parallelism ratio.

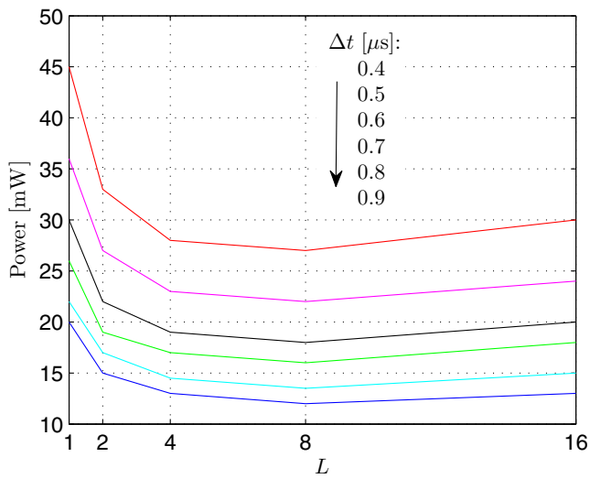

Fig. 8. Power consumption of DA-based quaternion multiplier as a function of the parallelism ratio.

4.3 4D-CORDIC-based circuit

As an example, a 4D-CORDIC-based quaternion multiplier by $q = \frac{1}{\sqrt{30}}(4 - i + 3j - 2k)$ has been developed using our methodology for rapid prototyping. In order to provide the desired accuracy, $\epsilon = 10^{-4}$, a word length of 16 bits is necessary, and the circuit needs to comprise $N = 16$ microrotations and $M = 7$ scaling iterations, which are realized as illustrated in Figs. 5 and 4(b), respectively.

The parameters of the stages are listed in Table 2. We have verified that it is advantageous to modify the microrotations in (28) so as to allow only one of $\sigma_k(n)$ to be nonzero. Without affecting the convergence, this simplifies the circuit as two-operand adders can be used in the scheme in Fig. 5 instead of four-operand ones. Additionally, the expression for the scaling factor changes from (29) to (19).

The FPGA implementation occupies 201 Slice Flip-Flops, 554 4-Input LUTs, and 351 Slices, which is about 1% of the resources provied by the Virtex chip. The maximum obtainable clock frequency is 120 MHz.

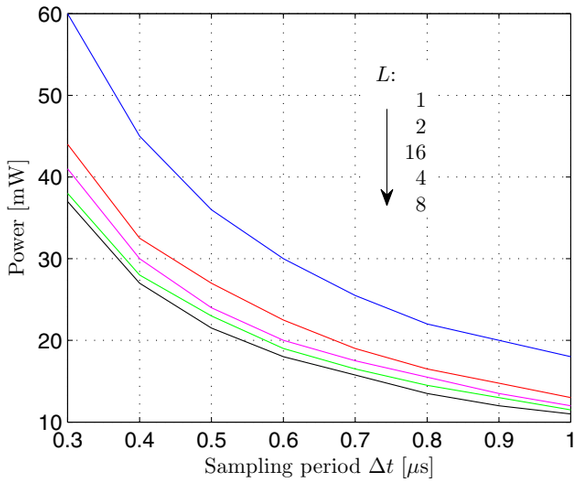

Fig. 9. Power consumption of DA-based quaternion multiplier as a function of the sampling period.

(a) Microrotations: nonzero $\sigma_k(n)$

Parameter	Value
$\sigma_2(0)$	1
$\sigma_1(1)$	-1
$\sigma_2(2)$	-1
$\sigma_2(3)$	1
$\sigma_1(4)$	1
$\sigma_1(5)$	-1
$\sigma_3(6)$	-1
$\sigma_2(7)$	-1
$\sigma_2(8)$	1
$\sigma_3(9)$	1
$\sigma_3(10)$	-1
$\sigma_2(11)$	-1
$\sigma_2(12)$	1
$\sigma_1(13)$	-1
$\sigma_1(14)$	1
$\sigma_3(15)$	1

(b) Microrotations: $\tau(n)$

Parameter	Value
$\tau(0)$	0
$\tau(1)$	1
$\tau(2)$	2
$\tau(3)$	3
$\tau(4)$	3
$\tau(5)$	4
$\tau(6)$	4
$\tau(7)$	4
$\tau(8)$	6
$\tau(9)$	6
$\tau(10)$	7
$\tau(11)$	7
$\tau(12)$	8
$\tau(13)$	9
$\tau(14)$	10
$\tau(15)$	10

(c) Scaling iterations $s(m)$

Parameter	Value
$s(0)$	2
$s(1)$	3
$s(2)$	4
$s(3)$	6
$s(4)$	7
$s(5)$	11
$s(6)$	12

Table 2. Parameters of 4D-CORDIC-based quaternion multiplier by $q = \frac{1}{\sqrt{30}}(4 - i + 3j - 2k)$; $N = 16$ and $M = 7$.

4.4 CORDIC-Inside-Lifting-based multiplier

The CORDIC-Inside-Lifting architecture is most complicated to design. Before prototyping a multiplier by q, all modifications of the hypercomplex number are reviewed that potentially determine computational schemes with all lifting coefficients in the range $-1, \ldots 1$. For every such a scheme, the number of 2D-CORDIC iterations is estimated that guarantees the desired accuracy of computations, and this allows for selecting candidates for fine-tuning parameters and then for FPGA synthesis.

Table 3 shows the accurate values of lifting coefficients related to 24 constructive[1] modifications of $q = \frac{1}{\sqrt{30}}(4 - i + 3j - 2k)$. For multiplications by only 7 of them, all lifting coefficients are in the desired range. And, the first of these 7 quaternions, which is shown the 2nd row of Table 3, $\tilde{q} = q_3 + q_2 i + q_1 j + q_4 k$, evidently can be realized using the lowest number of microrotations, $N = N_U + N_L + N_V = 8$ at accuracy of an individual CORDIC unit not worse than $\epsilon = 10^{-5}$.
In order to realize the multiplication by q, the multiplication by $\tilde{q}$ must be preprocessed with

$$\mathbf{P}_{\text{pre}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \tag{39}$$

No.	$\tilde{q}$	f_{11} $(-f_{22})$	f_{12} (f_{21})	g_{11} $(-g_{22})$	g_{12} (g_{21})	h_{11} $(-h_{22})$	h_{12} (h_{21})	N_U	N_L	N_V
1	$q_2 + q_1 i + q_3 j + q_4 k$	−2.110	0.073	0.548	0.365	−0.879	1.920			
2	$q_3 + q_2 i + q_1 j + q_4 k$	−0.395	0.448	0.730	0.365	−0.595	0.048	3	1	4
3	$q_4 + q_2 i + q_3 j + q_1 k$	−1.057	−1.076	0.548	−0.730	−0.737	−1.316			
4	$q_1 + q_3 i + q_2 j + q_4 k$	−0.905	1.191	−0.183	0.365	1.495	−0.009			
5	$q_1 + q_4 i + q_3 j + q_2 k$	−0.243	0.748	0.548	0.183	−0.643	−0.452	6	7	7
6	$q_4 + q_3 i + q_1 j + q_2 k$	−1.936	−0.266	0.730	0.183	−1.583	1.146			
7	$q_1 + q_2 i + q_3 j + q_4 k$	−0.495	−0.004	0.548	−0.365	−0.187	0.458	3	2	6
8	$q_2 + q_3 i + q_1 j + q_4 k$	−0.995	1.248	0.730	−0.365	−1.595	0.048			
9	$q_2 + q_4 i + q_3 j + q_1 k$	−0.457	−1.276	0.548	0.730	−1.097	−0.796			
10	$q_3 + q_1 i + q_2 j + q_4 k$	2.095	0.191	−0.183	−0.365	−1.105	1.791			
11	$q_4 + q_1 i + q_3 j + q_2 k$	−1.843	1.948	0.548	−0.183	−2.643	−0.452			
12	$q_3 + q_4 i + q_1 j + q_2 k$	−0.701	−0.325	0.730	−0.183	−0.465	0.616	4	1	6
13	$-(q_2 + q_1 i + q_3 j + q_4 k)$	0.418	−1.612	−0.548	−0.365	1.649	0.234			
14	$-(q_3 + q_2 i + q_1 j + q_4 k)$	1.795	−0.648	−0.730	−0.365	1.595	−1.048			
15	$-(q_4 + q_2 i + q_3 j + q_1 k)$	0.257	0.676	−0.548	0.730	0.577	0.436	7	5	6
16	$-(q_1 + q_3 i + q_2 j + q_4 k)$	−3.095	−3.191	0.183	−0.365	−0.695	−4.391			
17	$-(q_1 + q_4 i + q_3 j + q_2 k)$	3.043	−0.348	−0.548	−0.183	2.643	−1.548			
18	$-(q_4 + q_3 i + q_1 j + q_2 k)$	0.642	−0.910	−0.730	−0.183	0.995	0.501	8	1	5
19	$-(q_1 + q_2 i + q_3 j + q_4 k)$	2.033	−1.689	−0.548	0.365	2.341	−1.227			
20	$-(q_2 + q_3 i + q_1 j + q_4 k)$	1.195	0.152	−0.730	0.365	0.595	−1.048			
21	$-(q_2 + q_4 i + q_3 j + q_1 k)$	0.857	0.476	−0.548	−0.730	0.217	0.956	7	5	6
22	$-(q_3 + q_1 i + q_2 j + q_4 k)$	−0.095	−4.191	0.183	0.365	−3.295	−2.591			
23	$-(q_4 + q_1 i + q_3 j + q_2 k)$	1.443	0.852	−0.548	0.183	0.643	−1.548			
24	$-(q_3 + q_4 i + q_1 j + q_2 k)$	1.877	−0.969	−0.730	0.183	2.112	−0.028			

Table 3. Lifting coefficients that result from factorizing the multiplication matrices of modified versions of $q = \frac{1}{\sqrt{30}}(4 - i + 3j - 2k)$.

[1] In (Parfieniuk & Petrovsky, 2010b), it has been shown that other modifications are redundant.

and postprocessed with

$$\mathbf{P}_{\text{post}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{40}$$

Table 4 shows the parameters of the individual 2D-CORDIC modules of the scheme in Fig. 6(b). They have been obtained by careful optimization aimed at achieving the total accuracy of a multiplier not worse than $\epsilon = 10^{-4}$.
The results of FPGA design are worse than for the 4D-CORDIC approach, as about 50% more chip area is occupied. A single 2D-CORDIC circuit needs on average 50% less resources than a 4D-CORDIC unit, but three such units and extra adders are necessary to implement the scheme in Fig. 6(b). However, only this approach offers the invertibility of multiplication.

(a) **U**; $N_U = 3$ and $M_U = 7$

Parameter	Value
$\sigma(0)$	1
$\tau(0)$	0
$\sigma(1)$	-1
$\tau(1)$	4
$\sigma(2)$	1
$\tau(2)$	11
$s(0)$	1
$s(1)$	3
$s(2)$	5
$s(3)$	8
$s(4)$	10
$s(5)$	12
$s(6)$	13
$\mathbf{P}_{\text{pre}}$	$-\mathbf{\Gamma}_2$
$\mathbf{P}_{\text{post}}$	$\mathbf{J}_2$

(b) **L**; $N_L = 1$ and $M_L = 5$

Parameter	Value
$\sigma(0)$	-1
$\tau(0)$	1
$s(0)$	2
$s(1)$	6
$s(2)$	7
$s(3)$	9
$s(4)$	10
$\mathbf{P}_{\text{pre}}$	$\mathbf{I}_2$
$\mathbf{P}_{\text{post}}$	$\mathbf{I}_2$

(c)**V**; $N_V = 3$ and $M_V = 5$

Parameter	Value
$\sigma(0)$	1
$\tau(0)$	4
$\sigma(1)$	1
$\tau(1)$	6
$\sigma(2)$	1
$\tau(2)$	9
$s(0)$	2
$s(1)$	3
$s(2)$	4
$s(3)$	6
$s(4)$	6
$\mathbf{P}_{\text{pre}}$	$-\mathbf{I}_2$
$\mathbf{P}_{\text{post}}$	$\mathbf{I}_2$

Table 4. Parameters of CORDIC-Inside-Lifting-based quaternion multiplier.

5. Conclusions

There are several approaches to implementing constant-coefficient quaternion multipliers using only binary shifts and additions. Design experiments based on MATLAB, Virtex FPGA device, and our methodology for rapid prototyping, clearly show that each of the presented architectures has both advantages and disadvantages, and thus it is impossible to definitely recommend one of them as best suited for all development situations. From the point of view of rapid prototyping they require similar development effort and time. Distributed arithmetic offers a smooth trade-off between throughput and occupied chip area, whereas in CORDIC, we can only select between iterative or unfolded computations. However, the latter approach allows for maximizing throughput at modest resource utilization. Finally, the CORDIC-Inside-Lifting approach is the only option if lossless processing has to be realized, even though this architecture is neither economical in terms of chip area nor fast.

6. Acknowledgment

This work was supported by Bialystok University of Technology under the grant S/WI/4/08.

7. References

Alexiadis, D. & Sergiadis, G. (2009). Estimation of motions in color image sequences using hypercomplex Fourier transforms, *IEEE Trans. Image Process.* 18(1): 168–187, ISSN 1057-7149.

Calderbank, A. R., Daubechies, I., Sweldens, W. & Yeo, B.-L. (1998). Wavelet transforms that map integers to integers, *Appl. Comput. Harmon. Anal.* 5(3): 332–369, ISSN 1063-5203.

Chan, W., Choi, H. & Baraniuk, R. (2008). Coherent multiscale image processing using dual-tree quaternion wavelets, *IEEE Trans. Image Process.* 17(7): 1069–1082, ISSN 1057-7149.

Daubechies, I. & Sweldens, W. (1998). Factoring wavelet transforms into lifting steps, *J. Fourier Anal. Appl.* 4(3): 245–267, ISSN 1069-5869.

Delosme, J.-M. & Hsiao, S.-F. (1990). CORDIC algorithms in four dimensions, *Advanced Algorithms and Architectures for Signal Processing IV (Proc. SPIE 1348)*, San Diego, CA, pp. 349–360, ISBN-10 0819404098, ISBN-13 978-0819404091.

Denis, P., Carre, P. & Fernandez-Maloigne, C. (2007). Spatial and spectral quaternionic approaches for colour images, *Computer Vision and Image Understanding* 107: 74–87, ISSN 1077-3142.

Ell, T. & Sangwine, S. (2007). Hypercomplex Fourier transforms of color images, *IEEE Trans. Image Process.* 16(1): 22–35, ISSN 1057-7149.

Haldar, M., Nayak, A., Choudhary, A. & Banerjee, P. (2001). A system for synthesizing optimized FPGA hardware from MATLAB, *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, San Jose, CA, pp. 314–319, ISBN: 0-7803-7247-6.

Howell, T. D. & Lafon, J. C. (1975). The complexity of the quaternion product, *Technical Report TR 75-245*, Cornell University.
URL: *http://citeseer.ist.psu.edu/howell75complexity.html*

Hsiao, S. F. & Delosme, J. M. (1996). Parallel singular value decomposition of complex matrices using multidimensional CORDIC algorithms, *IEEE Trans. Signal Process.* 44(3): 685–697, ISSN 1053-587X.

Hsiao, S.-F., Lau, C.-Y. & Delosme, J.-M. (2000). Redundant constant-factor implementation of multi-dimensional CORDIC and its application to complex SVD, *J. VLSI Signal Process.* 25(2): 155–166, ISSN 0922-5773.

Karney, C. (2007). Quaternions in molecular modeling, *J. Molecular Graphics and Modelling* 25: 595–604, ISSN 1093-3263.

Leiterman, J. (2003). *Vector game math processors*, Wordware, Plano, TX, ISBN 1-55622-921-6.

Marion, A., Girard, P. & Vray, D. (2010). Quaternionic spatiotemporal filtering for dense motion field estimation in ultrasound imaging, *EURASIP J. Adv. Signal Process.* 2010: 11. Article ID 693218, ISSN: 1687-6172.

Meher, P., Valls, J., Juang, T.-B., Sridharan, K. & Maharatna, K. (2009). 50 years of CORDIC: Algorithms, architectures, and applications, *IEEE Trans. Circuits Syst. I* 56(9): 1893–1907, ISSN 1549-8328.

Parfieniuk, M. & Petrovsky, A. (2010a). Inherently lossless structures for eight- and six-channel linear-phase paraunitary filter banks based on quaternion multipliers, *Signal Process.* 90: 1755–1767, ISSN 0165-1684.

Parfieniuk, M. & Petrovsky, A. (2010b). Quaternion multiplier inspired by the lifting implementation of plane rotations, *IEEE Trans. Circuits Syst. I* 57(10): 2708–2717, ISSN 1549-8328.

Petrovsky, A., Parfieniuk, M. & Omieljanowicz, M. (2001). Computationally efficient hypercomplex filters based on matrix-by-vector multiplier, *Proc. IEEE Scientific Workshop Signal Process.*, Poznan, Poland, pp. 7–12.

Seberry, J., Finlayson, K., Spence Adams, S., Wysocki, T., Xia, T. & Wysocki, B. (2008). The theory of quaternion orthogonal designs, *IEEE Trans. Signal Process.* 56(1): 256–265, ISSN 1053-587X.

Took, C. & Mandic, D. (2010). Quaternion-valued stochastic gradient-based adaptive IIR filtering, *IEEE Trans. Signal Process.* 58(7): 3895–3901, ISSN 1053-587X.

Tsui, T., Zhang, X.-P. & Androutsos, D. (2008). Color image watermarking using multidimensional Fourier transforms, *IEEE Trans. Inf. Forensics Security* 3(1): 16–28, ISSN 1556-6013.

Verenik, A., Parfieniuk, M. & Petrovsky, A. (2007). An FPGA implementation of the distributed arithmetic based quaternionic multipliers for paraunitary filter banks, *Proc. 14th Int. Conf. "Mixed Design of Integrated Circuits and Systems" (MIXDES 2007)*, Ciechocinek, Poland, pp. 605–610, ISBN 83-922632-4-3.

White, S. A. (1989). Applications of distributed arithmetic to digital signal processing: A tutorial review, *IEEE ASSP Mag.* 6(3): 4–19, ISSN 0740-7467.

Zhou, J., Xu, Y. & Yang, X. (2007). Quaternion wavelet phase based stereo matching for uncalibrated images, *Pattern Recognit. Lett.* 28: 1509–1522, ISSN 0167-8655.

Rapid Prototyping Technology - Principles and Functional Requirements

Edited by Dr. M. Hoque

ISBN 978-953-307-970-7
Hard cover, 392 pages
Publisher InTech
Published online 26, September, 2011
Published in print edition September, 2011

Modern engineering often deals with customized design that requires easy, low-cost and rapid fabrication. Rapid prototyping (RP) is a popular technology that enables quick and easy fabrication of customized forms/objects directly from computer aided design (CAD) model. The needs for quick product development, decreased time to market, and highly customized and low quantity parts are driving the demand for RP technology. Today, RP technology also known as solid freeform fabrication (SFF) or desktop manufacturing (DM) or layer manufacturing (LM) is regarded as an efficient tool to bring the product concept into the product realization rapidly. Though all the RP technologies are additive they are still different from each other in the way of building layers and/or nature of building materials. This book delivers up-to-date information about RP technology focusing on the overview of the principles, functional requirements, design constraints etc. of specific technology.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Marek Parfieniuk, Nikolai A. Petrovsky and Alexander A. Petrovsky (2011). Rapid Prototyping of Quaternion Multiplier: From Matrix Notation to FPGA-Based Circuits, Rapid Prototyping Technology - Principles and Functional Requirements, Dr. M. Hoque (Ed.), ISBN: 978-953-307-970-7, InTech, Available from: http://www.intechopen.com/books/rapid-prototyping-technology-principles-and-functional-requirements/rapid-prototyping-of-quaternion-multiplier-from-matrix-notation-to-fpga-based-circuits

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen