MATRIX BASED SIGNAL PROCESSING ON A RECONFIGURABLE HARDWARE ACCELERATOR

Marius Otte, Jürgen Götze

University of Dortmund, Information Processing Lab
Otto-Hahn-Str. 4, 44221 Dortmund, Germany
E-Mail: {marius.otte, juergen.goetze}@udo.edu
Phone: +49 231 755 3036, FAX: +49 231 755 3251

Martin Bücker

Nokia Research Center
Bochum, Germany
E-Mail: martin.bucker@nokia.com

ABSTRACT

In this paper we describe a novel architecture for high speed signal processing applications. The respective hardware accelerator is designed to support a certain class of algorithms. The chosen class is based on a special kind of matrix transformation that could be efficiently implemented in terms of CORDIC modules. A consistent notation is presented to ease the process of mapping signal processing algorithms to the considered hardware. It is also shown how to implement complex valued arithmetic with less growth of complexity compared to standard techniques. Some algorithms are examined to illustrate the potential of our approach.

1. INTRODUCTION

A lot of modern signal processing applications require such a high computational power that only ASICs can fulfil the technical demands. Unfortunately, ASICs are inflexible, costly (development and debugging) and only economical for mass-products. As a consequence, system designers are striving to replace specialized hardware solutions with software based solutions as, e. g., developments in the field of software radio demonstrate.

Due to the fact that even the most commonly used programmable devices, i. e. DSPs, often lack of enough processing power, one tries to develop a solution that lays somewhere in between the two extrema *programmable signal processing* and *dedicated hardware*. The efforts in this area are summarized under the term *reconfigurable computing*.

Some already available reconfigurable solutions tend to be as flexible as microprocessors but with the focus on streaming signal processing. This general purpose-like orientation often yields to high power consumption and poor area efficiency.

Therefore, here the idea is to identify a class of algorithms and to develop a reconfigurable hardware accelerator structure that is optimized to support this very subset of algorithms used in digital signal processing applications.

Most of the digital signal processing algorithms stem from the same mathematical roots, namely linear algebra. Therefore, a promising candidate for such a class of algorithms comes from the area of matrix based algorithms. We will restrict ourselves to such a class and will present a framework of how to implement the algorithms on a specialized reconfigurable hardware structure.

The paper is organized as follows. First, we will introduce a definition of what we understand by *classes of algorithms*. Then we will declare a notation that makes it easier for us to write down the considered algorithms in a structured manner. In section 4 the actual hardware implementation of a processor element that serves as a basic algorithmic unit will be described. Section 5 will give an overview of the concept of the hardware accelerator that builds a frame the processor elements are embedded into.

This project is partly funded from the German Ministry of Education and Research (BMBF) in the ‘Mobile, Re-configurable Multi-standard Terminal for UMTS and WLAN (MoReTeX)’project, which is a part of the umbrella ‘Software Defined Radio Based Architecture Studies for Reconfigurable Mobile Communication Systems (RMS)’.

2. CLASSES OF ALGORITHMS

A class of algorithms as it is understood in this paper is the set of algorithms that fundamentally are based on a finite set of non-elementary basic arithmetic and/or logic operations. If we choose e. g. the *dot product* of two vectors as the basic arithmetic operation of a certain algorithmic class, the following algorithms would be members of this class: cross-correlation calculation, autocorrelation calculation, FIR filtering, Matched filter receiver.

Now, if an efficient hardware implementation for the basic operations is found, the processing of the whole class of algorithms can be speed up compared to architectures that can cope with arbitrary algorithms. One can say that we are trying to introduce as much reconfigurability as needed but as less as it is possible.

In modern digital signal processing, most of the used algorithms or techniques can be viewed as applied linear algebra. Further on, most of these algorithms can be described in terms of matrix computations. Thus, we identify a class of algorithms that is based on the basic arithmetic operation *2×2 orthogonal/linear matrix transformation*.

3. NOTATION

For a simplified and structured notation of the used algorithms we introduce a matrix manipulation operator $\mathcal{M}_{i,j}$. Application of operator $\mathcal{M}_{i,j}$ to a $m \times n$ matrix $\mathbf{X}$ with elements $x_{j,i}$ at row j and column i is actually a matrix multiplication: $\mathcal{M}_{i,j}\{\mathbf{X}\} = \mathbf{M}\mathbf{X}$, where $\mathbf{M}$ is of suitable size. Four different operator modes are defined: (1) linear evaluation, i. e. evaluate and apply a Gaussian transformation such that $x_{j,i}$ becomes zero, (2) orthogonal evaluation, i. e. evaluate and apply a Givens transformation such that $x_{j,i}$ becomes zero, (3) linear application and (4) orthogonal application. Note that modes (3) and (4) require an additional parameter

θ that is denoted by a superscript of $\mathcal{M}$, i. e. $\mathcal{M}^{\theta}$. In each mode the transformation matrix $\mathbf{M}$ results from embedding the matrix

$$\mathbf{M}_{\square} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{1}$$

into a $m \times m$ identity matrix, such that a becomes $m_{i,i}$, b becomes $m_{i,j}$, c becomes $m_{j,i}$ and d becomes $m_{j,j}$, where $m_{k,l}$ is the element of $\mathbf{M}$ at row k and column l. Table 1 shows the values for a, b, c and d according to different modes.

Mode	a	b	c	d
(1), (3)	1	0	θ	1
(2), (4)	$\cos\theta$	$\sin\theta$	$-\sin\theta$	$\cos\theta$

Table 1. Elements of matrices $\mathbf{M}$ and $\mathbf{M}_{\square}$, respectively, in different modes.

Another operator that is used is the element-by-element product $\odot$ of two matrices of same size. Product $\mathbf{X} = \mathbf{A} \odot \mathbf{B}$ means $x_{i,k} = a_{i,k} b_{i,k}$ for all possible i, k. Finally, the interchange operator $\mathcal{X}_{i,j}^{k,l}\{\mathbf{X}\}$ changes position of elements $x_{i,j}$ and $x_{k,l}$ in matrix $\mathbf{X}$ while leaving all other elements untouched.

4. HARDWARE REALIZATION OF $\mathcal{M}$

Using the matrix manipulation operator proves convenient when it comes to hardware realization. The basic hardware module is a processing element (PE) with three input and three output ports and one configuration port. Table 2 lists the correspondence between the modules and operator $\mathcal{M}$ for each possible configuration and mode, respectively.

Processor Element

Basically, the PE a is reconfigurable CORDIC (*COordinate Rotation on a DIgital Computer*) cell. The CORDIC algorithm makes it possible to carry out vector rotations (and hence to calculate trigonometric functions) only by using shifters and adders, which is very attractive from a hardware point of view[1][2]. The general CORDIC iteration is given by

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & m d_k 2^{-\sigma_k} \\ -d_k 2^{-\sigma_k} & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} \tag{2}$$

$$\theta_{k+1} = \theta_k + d_k \alpha_k \tag{3}$$

Dependent on the choice of m and σ_k we can distinguish between different types of CORDICs. Up to now, we support linear and orthogonal mode which corresponds to $m \in \{0, 1\}$ and $\sigma_k = k$.

To perform a complete transformation, i. e. an application of $\mathcal{M}$, we have to loop through the iterations in equation (2) and (3). The number of iterations depends on the wordlength of x and y components and/or on the desired resolution of the results.

In the following we will sketch the main properties of our CORDIC implementation that is depicted in Fig 1. The three input ports of the PE are divided into the two vector components of the vector that should be transformed (x, y) and into the transformation parameter θ. Each of the three datapathes is assumed to be implemented with a wordlength of w bits. Additionally a configuration port exists that is used for selecting the defined modes and setting the factor λ.

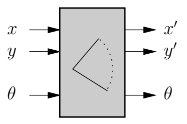
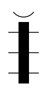
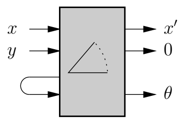
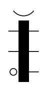
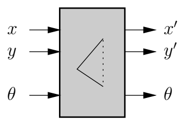
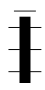
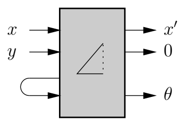
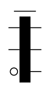
$\overset{\smile}{\mathcal{M}}{}^{\theta}_{1,2}\left\{\begin{bmatrix} x \\ y \end{bmatrix}\right\} = \begin{bmatrix} x' \\ y' \end{bmatrix}$	$\equiv$		$\equiv$	
$\overset{\smile}{\mathcal{M}}_{1,2}\left\{\begin{bmatrix} x \\ y \end{bmatrix}\right\} = \begin{bmatrix} x' \\ 0 \end{bmatrix}$	$\equiv$		$\equiv$	
$\overline{\mathcal{M}}{}^{\theta}_{1,2}\left\{\begin{bmatrix} x \\ y \end{bmatrix}\right\} = \begin{bmatrix} x' \\ y' \end{bmatrix}$	$\equiv$		$\equiv$	
$\overline{\mathcal{M}}_{1,2}\left\{\begin{bmatrix} x \\ y \end{bmatrix}\right\} = \begin{bmatrix} x' \\ 0 \end{bmatrix}$	$\equiv$		$\equiv$	

Table 2. Processor elements (PEs) equivalent to the four modes of $\mathcal{M}$. In orthogonal modes it is also possible to configure the PEs to perform a rotation with angle $-\theta$ by setting a reverse angle bit in the configuration word. In such case we use a $\frown$ instead of a $\smile$ for the icons.

The configuration port in the current implementation has a wordlength of eight bit. Fundamentally, a CORDIC element consists of a series of shift-add stages as it can be seen in Fig. 1. In case of the linear modes the chain consists of 16 stages; in case of the orthogonal modes the first two stages are substituted or bypassed by a pre-rotation stage that transforms the input vector into a $\pm 45^\circ$ segment in the x-y-plane.

Each stage carries out a so-called microrotation. Dependent on the CORDIC mode, the steered adders that are part of each stage are switched according to the value of θ-data (mode 1,2) or by examination of the sign of y.

The vector $[x\ y]$ that results at the end of the chain is stretched by a factor

$$1/K = \prod_{k=1}^{w-1} \sqrt{1 + d_k^2 2^{-2k}} \tag{4}$$

and must therefore scaled appropriate to get the correct result. This correction is implemented by two scalar multipliers. Another scalar multiplication with λ is required in the x-branch to implement the above mentioned matrix operator $\odot$. Here, we use only a fixed set of four possible factors mainly to support exponentional weighting as it is required in updating QR-algorithms [3](see below for details).

A suitable interconnection of the basic modules corresponds to an application of $\mathcal{M}$ to a $m \times n$ matrix. Fig. 2 shows the realization of $\mathcal{M}$ when it is applied to a four column matrix $\mathbf{X}$. Because only two rows of the matrix $\mathbf{X}$ are affected by the operation, only these two rows (i and j) are processed by the module row. The series connection of multiple matrix manipulators that transform different rows of a matrix therefore yields to a cascading of multiple module rows.

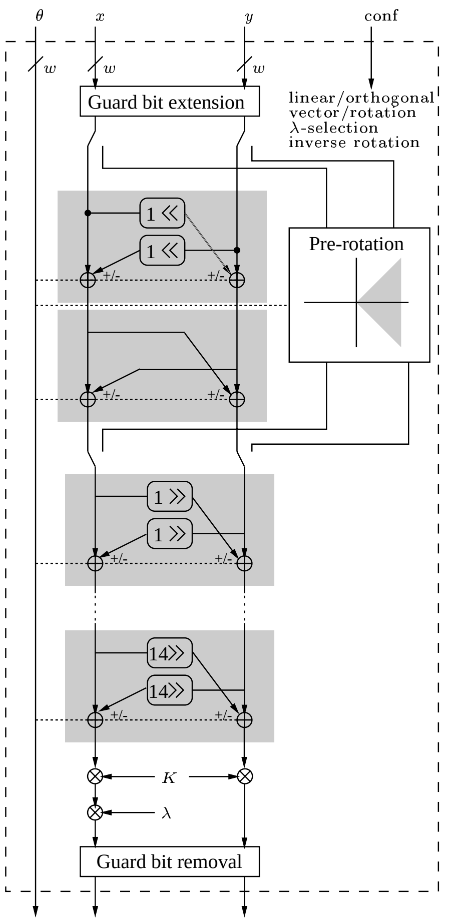


Fig. 1. Internal structure of the CORDIC PE. The pre-rotation is only necessary in orthogonal modes. In linear modes holds $K = 1$.

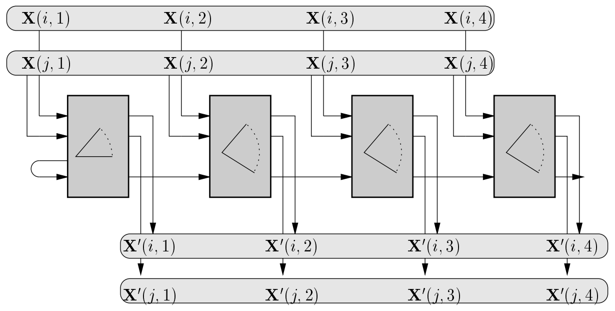


Fig. 2. Interconnection of the PEs to realize $\breve{\mathcal{M}}_{i,j}(\mathbf{X}) \to \mathbf{X}'$. The four PEs form a *module row*.

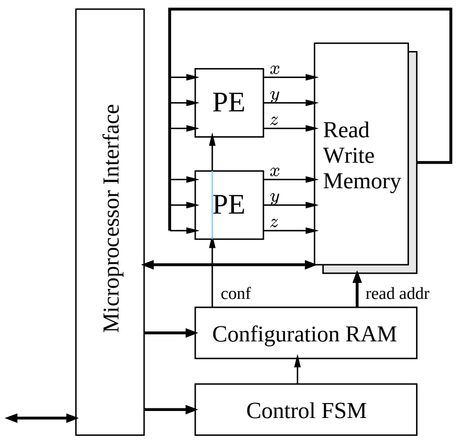


Fig. 3. Hardware accelerator structure.

5. CONFIGURABLE HARDWARE ACCELERATOR

The PEs serve as arithmetic units of a special hardware accelerator as it is shown in Fig. 3. Its architecture is described in [4]. The current implementation basically consists of two PEs that are connected to a fast memory interface. In each processing step the PEs read data from one bank of the *Read-Write-Memory* and store the output data in the other bank. The read addresses are generated by the *Configuration RAM* while the write addresses are serially incremented. After a number of activations of the PEs the microprocessor interface takes over the control and is responsible for data exchange with the host system.

The design of the accelerator could be easily extended by more PEs working in parallel. Moreover, the PEs have a fixed interface description that allows to exchange CORDIC PEs by other PE implementations that support some other class of algorithms. See [5] and [6] for the description of a MAC PE and a Reed Solomon PE, respectively.

To illustrate the functionality of the accelerator in more detail, we will study the design flow starting with a given algorithm. The first design step in programming the hardware accelerator is the development of a virtual array of interconnected processor elements. If we have formulated an algorithm in terms of the operators we have defined in section 3, it is also possible to assemble a processor array.

The netlist of the virtual processor array is written to the configuration RAM of the accelerator. The netlist not only contains information about the interconnection of ports but also configuration information that is necessary to set the operation mode of the PEs.

The memory interface structure requires that exactly one register has to be placed between interconnected ports of the PEs. Therefore one has to watch the timing carefully to ensure a correct synchronization. On the other hand, the accelerator enables a high degree of flexibility because it is allowed to change interconnections and modes of processor elements during runtime. This is possible due to the fact that the configuration RAM can hold multiple configurations in parallel. We call this concept *time variant netlists*. Mapping tools have been developed to automate the process of synthesizing the appropriate description files.

6. SELECTED ALGORITHMS

Here we will present some algorithms that belong to the above defined class of algorithms. We will focus on algorithms that are completely or partially based on an update QR decomposition. This choice is motivated by the high usefulness of these algorithms is the area of digital signal processing.

6.1. QR-Decomposition for Solving LS-Systems

The QR-decomposition of a $m \times n$ $(m > n)$ matrix $\mathbf{X}$ into a product $\mathbf{Q}[\mathbf{R}^T \quad \mathbf{0}]^T$, where $\mathbf{Q}$ is a $m \times m$ orthogonal matrix and $\mathbf{R}$ is a $n \times n$ upper triangular matrix constitutes a fundamental technique for the efficient and numeric stable solution of overdetermined linear systems. Based on the equations

$$\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \quad \Leftrightarrow \quad \min_{\mathbf{w}} ||[\mathbf{R}^T \quad \mathbf{0}]^T\mathbf{w} - \underbrace{\mathbf{Q}^T\mathbf{y}}_{\mathbf{y}_q}||_2^2 \qquad (5)$$

the solution $\mathbf{w}$ can be easily calculated by a backsubstitution process. Note, that the transformation $\mathbf{Q}^T\mathbf{X}$ that is used to convert $\mathbf{X}$ into an upper triangular matrix has also be applied to vector $\mathbf{y}$.

Since the calculation and application of $\mathbf{Q}^T$ can be viewed as a sequence of elementary 2×2 matrix transformations, the process can be described in terms of the matrix manipulation operator:

```
Calculate $\mathbf{R}$ and $\mathbf{y}_q$, such that
$\mathbf{X} = \mathbf{Q}[\mathbf{R}^T \quad \mathbf{0}]^T$ and $\mathbf{y}_q = \mathbf{Q}^T\mathbf{y}$

for $i = 1 : n$
    for $j = i + 1 : m$
        $[\mathbf{X}|\mathbf{y}] \leftarrow \breve{\mathcal{M}}_{i,j}\{[\mathbf{X}|\mathbf{y}]\}$
    end
end
$\mathbf{R} \leftarrow \mathbf{X}$
$\mathbf{y}_q \leftarrow \mathbf{y}$
```

The mapping of this algorithm to a virtual processor array now is straightforward. Basically, it is done by a vertical concatenation of n module rows, each of them with a structure as depicted in Fig. 2. However, since each y-output of the leftmost PE generates a zero, the number of cells is reduced by one from row to row. This results in a triangular processor array as it is shown for $n = 3$ in Fig. 4. Note that there exist internal feedback connections from x-out to x-in. These are necessary to hold intermediate results. After n activations of the complete array the feedback registers contain the elements of matrix $\mathbf{R}$ and vector $\mathbf{y}_q$.

The number of rows m of matrix $\mathbf{X}$ is arbitrary using this algorithm, because with each processing step a new row of the matrix is processed by the array. Hence, this method is also called *Updating QR-Algorithm*.

For solving the system, we still have to perform the backsubstitution. A backsubstitution array that is fed with the content of the feedback registers as proposed e. g. in [7] would be one possibility to handle the problem.

On the other hand, it is also possible to exploit the on-line reconfiguration capabilies of the virtual array. For this purpose we have to reconfigure the PEs from orthogonal mode to linear mode after n processing steps. If at the same time the input data is adjusted properly, the array output yields a sequence containing the elements of vector $\mathbf{w}$. The method is based on the Schur complement [8], that states

$$\overline{\mathcal{M}}_{n,2n} \cdots \overline{\mathcal{M}}_{1,n+1} \left\{ \begin{bmatrix} \mathbf{R} & \mathbf{y}_q \\ \mathbf{I}_{n,n} & \mathbf{0}_{n,1} \end{bmatrix} \right\} = \begin{bmatrix} \mathbf{R}' & \mathbf{y}_q' \\ \mathbf{0}_{n,n} & \mathbf{R}^{-1}\mathbf{y}_q \end{bmatrix} \qquad (6)$$

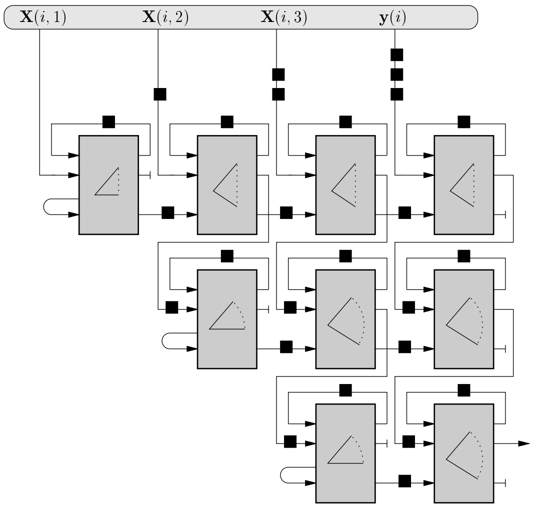
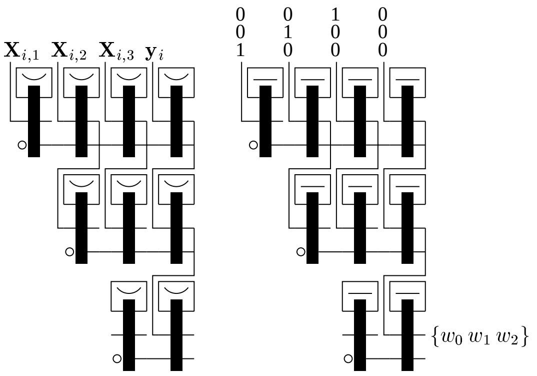


Fig. 4. Basic QR-decompositon array.



$\mathbf{X}_{i,1}$ $\mathbf{X}_{i,2}$ $\mathbf{X}_{i,3}$ $\mathbf{y}_i$
0 0 1 0
0 1 0 0
1 0 0 0
$\{w_0\, w_1\, w_2\}$

Fig. 5. Two modes of the QR-decomposition array. Note that in this simplified notation each interconnection between ports corresponds to a unit delay. After n activations of the left array the $\mathbf{R}$ matrix and vector $\mathbf{y}_q$ is stored into the internal registers. To perfom the backsubstitution, the PEs are switched to linear mode (right array).

It becomes obvious, that the solution $\mathbf{w} = \mathbf{R}^{-1}\mathbf{y}_q$ is contained in the transformed matrix in the lower right part. Regarding the virtual array, this means that with every activation one element w_i is produced by the y-output port of the lowermost PE if the vectors [1 0 0 0], [0 1 0 0] and [0 0 1 0] are fed into the array one after another. The swithing process and the two virtual arrays, respectively, are shown in Fig. 5.

6.2. Complex valued signal processing

Complex number arithmetic plays an important role in various signal processing algorithms. However, typical arithmetical architectures only support calculations in the real number system. This is not a real problem due to the fact that complex arithmetic can be expressed in terms of real arithmetic. A complex multiplication, e. g., can be substituted by 4 real multiplication and 2 real additions or by 3 real multiplications and 5 real additions [9]

If the basic algorithmic operations are not summations and

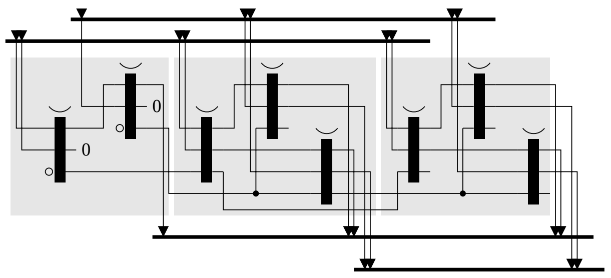


Fig. 6. A module row that corresponds to the one shown in Fig. 2 but for complex valued input and only three input columns.

multiplications but, like in our case, real valued matrix transformations the transition from complex to real becomes slightly more difficult. As described in [10] a unitary or linear 2×2 matrix transformation can be expressed by a sequence of 4 real valued 2×2 matrix transformation. However, it is possible to implement the complex transformation by means of only 2 or 3 real PEs in typical applications like the here mentioned QR-decompositon based algorithms. The resulting complex module row is shown in Fig. 6.

Note that in case of using feedback branches as they are necessary in complex QR-updating the feedback path now goes from x-out to y-in. This means that if the hierarchical complex cell is switched from unitary to linear transformation mode, these connections have to be rewired to a x-out $\rightarrow$ x-in feedback connection in order not to destroy the register entries. Here the on-line reconfiguration capabilies of the interconnection structure becomes evident.

6.3. Example 1: Least squares solution, calculation of $e(m)$

A couple of applications require the solution of an overdetermined system of linear equations like [7]

$$\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \qquad \text{with} \quad e(m) \quad = \quad \mathbf{X}(m,:)\mathbf{w}_{\text{opt}} - \mathbf{y}(m), \qquad (7)$$

where $\mathbf{X}$ is a $m \times n$ matrix, without explicitly calculating the solution vector $\mathbf{w}_{\text{opt}}$; sufficient is a calculation of the last component of the error vector $\mathbf{e} = \mathbf{X}\mathbf{w}_{\text{opt}} - \mathbf{y}$.

The above defined notation allows a very compact formulation of an algorithm that solves the problem:

$$\mathbf{B} = \overline{\mathcal{M}}_{n+1,n+2}\{\mathcal{X}^{n+1,n+1}_{n+2,n+1}\{\breve{\mathcal{M}}_{n,n+1} \cdots \breve{\mathcal{M}}_{1,n+1}\{\mathbf{A}\}\}\} \quad (8)$$

Here, $\mathbf{A}$ and $\mathbf{B}$ are $(n{+}2) \times (n{+}2)$ matrices, where $\mathbf{A}$ contains the input data and $b_{n+2,n+2}$ yields the negated error signal $-e(m)$. This formulation becomes more understandable with a view on Fig. 7 where the matrix manipulation steps are illustrated.

The resulting virtual array structurely looks exactly like the QR-decomposition array shown in Fig. 4 but is composed of the complex modul row shown in Fig. 6.

In adaptive filtering applications the input data stream in usually of infinite length. In such case, the matrix $\mathbf{X}$ has also an infinite number of rows; in each time step a new row is appended to $\mathbf{X}$. In order to take into account only data symbols from the near past an exponentional weighting factor λ is introduced. Thereby each data sample is scaled by a factor $\lambda^{\text{(sample age)}}$. Referring to the processor array this means that the internal feedback connections have to be multiplied with λ after each processing step.

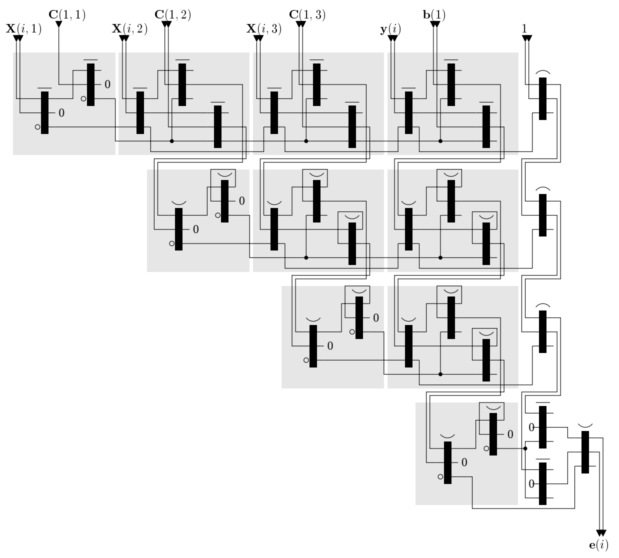


Fig. 8. Virtual processor array for the solution of a LS problem with one constraint.

6.4. Example 2: Least squares solution with constraints, calculation of $e(m)$

The problem mentionend in section 6.3 can be extended by introducing additional linear constraints. The problem formulation then would be

$$\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \qquad \text{subj. to} \quad \mathbf{C}\mathbf{w} = \mathbf{b}$$
$$\text{with} \quad e(m) \quad = \quad \mathbf{X}(m,:)\mathbf{w}_{\text{opt}} - \mathbf{y}(m), \qquad (9)$$

where matrix $\mathbf{C}$ and vector $\mathbf{b}$ contain constants. A number of applications are based on this set of equations, e. g. adaptive beamforming [10], constrained MMSE detection [11][12] or pre-equalization in OFDM systems [13]. All of these applications can be implemented on the same virtual array; the only difference lays in the assembly of the input data (matrix $\mathbf{X}$ and vector $\mathbf{y}$) and the number of constrained equations.

The incorporation of k linear constrains is achieved by substituting the first n rows modules of the array operating in orthogonal mode by row modules operating in linear mode. Fig. 8 shows an example for an array that solves equation (9) for a 3 column matrix $\mathbf{X}$ and one constrained equation.

7. CONCLUSION

We have proposed a strategy for the efficent and flexible hardware implementation of a class of algorithms that we call matrix based signal processing algorithms. A common notation for these algorithms was presented. A processing element that had been developed for the embedding in a special hardware accelerator was presented. Algorithms that are based on a QR-updating process have been used as an example to underline the capabilites of the approach especially in complex valued signal processing. However, other applications as e. g. orthogonal filters, orthogonal signal transformations or eigenvalue decompositions can also be incorporated into the presented framework.

Initialization: $\mathbf{R}(0) = \mathbf{0}_{n\times n}, \mathbf{y}_q(0) = \mathbf{0}$

Calculate $e(m)$ *if* $\mathbf{R}(m-1)$ *and* $\mathbf{y}_q(m-1)$ *are known:*

$$\begin{bmatrix} \mathbf{R}(m) & \mathbf{y}_q(m) & | \\ \mathbf{0}_{1\times n} & e_q(m) & \gamma \\ \mathbf{0}_{1\times n} & 1 & 0 \end{bmatrix} \leftarrow \begin{bmatrix} \lambda\mathbf{I}_{n\times n} & \mathbf{I}_{n\times 2} \\ \mathbf{I}_{2\times n} & \mathbf{I}_{2\times 2} \end{bmatrix} \odot \breve{\mathcal{M}}_{n,n+1} \cdots \breve{\mathcal{M}}_{1,n+1} \left\{ \begin{bmatrix} \mathbf{R}(m-1) & \mathbf{y}_q(m-1) & \mathbf{0}_{n\times 1} \\ \mathbf{X}(m,:) & y(m) & 1 \\ \mathbf{0}_{1\times n} & 1 & 0 \end{bmatrix} \right\}$$

with $\gamma = \prod_{i=1}^{n} \cos\theta_i$ and $|$: dont't care

$$\begin{bmatrix} \mathbf{R}(m) & \mathbf{y}_q(m) & | \\ \mathbf{0}_{1\times n} & 1 & \gamma \\ \mathbf{0}_{1\times n} & e_q(m) & 0 \end{bmatrix} \leftarrow \mathcal{X}_{n+2,n+1}^{n+1,n+1} \left\{ \begin{bmatrix} \mathbf{R}(m) & \mathbf{y}_q(m) & | \\ \mathbf{0}_{1\times n} & e_q(m) & \gamma \\ \mathbf{0}_{1\times n} & 1 & 0 \end{bmatrix} \right\}$$

$$\begin{bmatrix} \mathbf{R}(m) & \mathbf{y}_q(m) & | \\ \mathbf{0}_{1\times n} & 1 & \gamma \\ \mathbf{0}_{1\times n} & 0 & -\gamma e_q(m) \end{bmatrix} \leftarrow \overline{\mathcal{M}}_{n+1,n+2} \left\{ \begin{bmatrix} \mathbf{R}(m) & \mathbf{y}_q(m) & | \\ \mathbf{0}_{1\times n} & 1 & \gamma \\ \mathbf{0}_{1\times n} & e_q(m) & 0 \end{bmatrix} \right\} \quad \text{with } e(m) = \gamma e_q(m)$$

Fig. 7. Direct calculation of the error signal $e(m)$. The last transformation to calculate $\gamma e_q(m)$ is again an application of the Schur complement.

8. REFERENCES

[1] J.E. Volder, "The cordic trigonometric computing technique," *IRE Trans. Electronic Computers*, vol. EC-8, pp. 330–334, 1959.

[2] J.S. Walther, "An unified algorithm for elementary functions," in *Proc. Spring Joint Computer Conference*. 1971, vol. 38, p. 397, AFIPS press.

[3] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1991.

[4] B. Oelkrug, M. Bücker, D. Uffmann, A. Dröge, J. Brakensiek, and M. Darianian, "Programmable hardware accelerator for universal telecommunication applications," in *2nd Workshop on Software Radios*, Karlsruhe, Germany, 2002.

[5] H. Lange, O. Franzen, H. Schröder, M. Bücker, and B. Oelkrug, "Reconfigurable Multiply-Accumulate-based Processing Element," in *IEEE Workshop on Heterogeneous Reconfigurable Systems on Chip*, Hamburg, Germany, 2002.

[6] S. Roy, M. Bücker, W. Wilhelm, and B.S. Panwar, "Reconfigurable Hardware Accelerator for a Universal Reed Solomon Codec," in *1st IEEE Int. Conference on Circuits and Systems for Communication*, St. Petersburg, June 2002.

[7] S. Haykin, *Adaptive Filter Theory*, Prentice Hall (3rd edition), 1995.

[8] I. Schur, "Über Potenzreihen, die im Inneren des Einheitskreises beschränkt sind. I.," *J. für reine und angewandte Mathematik*, vol. 147, pp. 205–232, 1917.

[9] Donald E. Knuth, *The Art of Computer Programming, 3rd ed.*, Addison-Wesley, 1998.

[10] M. Otte, M. Bücker, and J. Götze, "Complex CORDIC-like Algorithms for Linearly Constrained MVDR Beamforming," in *Proc. of IEEE 2000 International Zurich Seminar on Broadband Communication*, 2000.

[11] M. Otte and J. Götze, "Parallel Implementation of Modified MMSE Interference Cancellation for CDMA Systems," in *Int. Conf. on Telecommunications ICT'2000*, 2000.

[12] S. R. Kim, Y. G. Jeong, I. K. Choi, and S. Lee, "An adaptive pilot symbol-aided mmse receiver in fading channels," in *Proc. IEEE, ICT'99*, 1999.

[13] H. Schmidt and K. D. Kammeyer, "Impulse truncation for wireless OFDM systems," in *International OFDM-Workshop 2000, Hamburg*, 2000.