日本電気株式会社　システム IP コア研究所
SYSTEM IP CORE LABORATORY,
NEC CORPORATION

研修報告書
INTERNSHIP REPORT.

Interference Mitigation for WCDMA Using QR Decomposition and a CORDIC-based Reconfigurable Systolic Array

Robin SCHEIBLER†, James OKELLO††, Katsutoshi SEKI††, Tomoyoshi KOBORI††,
and Masao IKEKAWA††

† Swiss Federal Institute of Technology, Lausanne, Switzerland
†† System IP Core Laboratory, NEC Corporation

Abstract This paper presents implementation and performance of QR Decomposition based Recursive Least--Squares (QRD-RLS) for interference mitigation in Wideband CDMA (WCDMA). The implementation is carried on CORSAEngine which is a new Software-Defined Radio (SDR) processor developed by NEC Corporation and highly optimized for MIMO-OFDM systems. It is shown how QRD-RLS can be mapped on its rectangular CORDIC-based reconfigurable systolic array, hence demonstrating its capability to process WCDMA. In addition, the performance of CORSAEngine is compared to that of other architectures and it is found to achieve at least 91% of the performance of dedicated hardware in terms of computational density.

Key words Software-Defined Radio, QR Decomposition, Wideband CDMA, Interference Mitigation

1. Introduction

In 1991, Mitola [15] introduced the concept of Software-Defined Radio (SDR) that allows operations of different modes of communications systems on a single hardware, dramatically decreasing equipment costs and development time of new technologies. While programmability is attractive to mobile communication equipments manufacturers and operators, it also brings one of the biggest challenges of SDR. The need to maintain high performance while retaining enough flexibility to process as many different standards as possible. This constraint becomes even more difficult to fulfill as modern communication standards require more complex signal processing technology.

In the field of cellular communications, such modern standards are usually referred to as Beyond 3G (B3G) technologies. It has been recognized that B3G systems, already exemplified by WiMAX and 3GPP LTE among others, will heavily rely on Orthogonal Frequency Division Multiplexing (OFDM) and Multiple-Input Multiple-Output (MIMO) technologies [18]. But at the same time, it is important for an SDR to support non-OFDM-based standards like IS-95, CDMA2000 and WCDMA. Firstly, those systems enjoy a very deep market penetration and are likely to remain used for many years. Secondly, in the case of WCDMA, it has the potential to be used in conjunction with an OFDM scheme such as in Multi-Carrier Code Division Multiple Access (MC-CDMA).

Recently, NEC Corporation developed CORSAEngine, a new SDR processor highly optimized for MIMO-OFDM systems [17]. Its rectangular COordinate Rotation DIgital Computer (CORDIC) based reconfigurable systolic array makes it highly suitable to process the computationally intensive baseband algorithms required by those systems, among others QR Decomposition (QRD), Singular Value Decomposition (SVD), least-squares fit or fast Fourier transform. However, performance of efficient interference mitigation algorithms for WCDMA had not been investigated on this processor.

This paper presents the implementation and the performance of QR Decomposition based Recursive Least-Squares (QRD-RLS) for interference mitigation of WCDMA on CORSAEngine. QRD-RLS has been shown to effectively mitigate both Intersymbol Interference (ISI) and Multiple Access Interference (MAI), outperforming the conventional Rake while maintaining reasonable complexity when implemented as a systolic array [14]. In this paper, it is shown how this arbitrarily large systolic array can be split into parts that fit on the reduced size array of CORSAEngine. Through reconfigurability, it is furthermore possible to run successively those different parts on the same hardware structure.

The remainder of this paper is organized as follows. Section 2 gives a brief revision of conventional and QRD-RLS based interference for WCDMA along with simulation results and computational load comparison of those two methods. In Section 3, the architecture of CORSAEngine is described. The mapping of QRD-RLS onto the systolic array

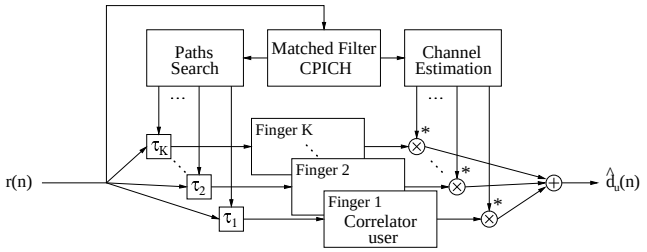


Fig. 1 Block diagram of the conventional RAKE receiver.

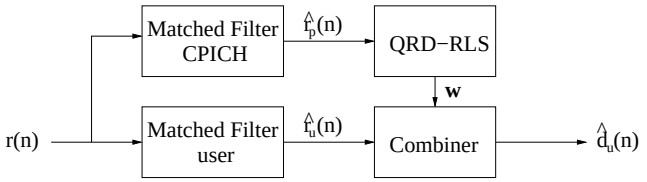


Fig. 2 Block diagram of WCDMA receiver based on QRD-RLS.

is described in Section 4. Finally in Section 5, the performance of the implementation of WCDMA on CORSAEngine is assessed and a benchmark against other devices is done. Section 6 concludes this paper.

2. Interference Mitigation in WCDMA

2.1 Conventional Interference Mitigation

The conventional interference mitigation for WCDMA is characterized by the RAKE receiver shown in Fig. 1. It uses short-time averaging (typically two slots) of the received pilot symbols to estimate the channel characteristics. Then, long-time averaging (about one frame) is used to get a good power delay profile of the channel. The Path Search uses a threshold-based algorithm to select the paths with a sufficiently large Signal-to-Noise Ratio (SNR). Those paths are despread using a bank of correlators and combined according to the Maximum Ratio Combining (MRC) principle with respect to the channel coefficients. For different algorithm for channel estimation and path search, refer to [8], [9]. For more details about the principles of the RAKE receiver, refer to [16].

2.2 QRD-RLS Interference Mitigation

This section describes QRD-RLS based interference mitigation applied to WCDMA. A block diagram of the receiver considered is shown in Fig. 2. First the Common Pilot CHannel (CPICH) and the signal of the user of interest are despread using Matched Filters (MF) corresponding to their respective spreading codes. The despread pilot signal $\hat{r}_p(n)$ is then sent to the QRD-RLS weight calculation unit which produces the optimal weight vector $\boldsymbol{w}$. It is then sent to the combiner and used to combine the despread user signal $\hat{r}_u(n)$.

In the WCDMA system, the despread signal can be written as in [6] :

$$\hat{r}(n) = \sigma_l d(i) + I(n) + \xi(n), \tag{1}$$

where the time index $n = iF + l$ with $i \in \mathbb{N}$ and $l \in \{0, \ldots, F-1\}$, F is the spreading factor, σ_l is a multiplicative coefficient introduced by the channel impulse response and the spreading code autocorrelation function and $d(i)$ is the i^{th}symbol sent. $I(n)$ is an interference term created by the ISI and the MAI. $\xi(n)$ is the filtered noise. Let's define $\boldsymbol{u}(i) = [\hat{r}(iF), \ldots, \hat{r}(iF+M-1)]^T$, a vector containing the M first chips of the despread signal corresponding to the i^{th} symbol sent. The goal is then to find the optimal weight vector $\boldsymbol{w}(m) = [w_0(m), \ldots, w_{M-1}(m)]^T$ to combine the elements of $\boldsymbol{u}_u(i) = [\hat{r}_u(iF_u), \ldots, \hat{r}_u(iF_u+M-1)]^T$ in order to enhance the symbol $d_u(i)$ and reduce the interference signal $I(n)$, m being the number of symbols received so far. Subscript p and u are used to distinguish between pilot and user signals.

QRD-RLS is a technique borrowed from the adaptive filtering theory [11]. To adaptively calculate $\boldsymbol{w}(m)$, it attempts to minimize the following error function :

$$E(m) = \|\boldsymbol{\Lambda}(m)(\boldsymbol{A}(m)\boldsymbol{w}(m) - \boldsymbol{d}(m))\| \tag{2}$$

where $\boldsymbol{A}(m) = [\boldsymbol{u}_p(0), \ldots, \boldsymbol{u}_p(m)]^H$ contains the received pilot signal, $\boldsymbol{\Lambda}(m) = diag(\lambda^{m/2}, \ldots, \lambda^{1/2}, 1)$ the exponentially decreasing forgetting factor and $\boldsymbol{d}(m) = [d_p(0), \ldots, d_p(m)]^H$ the original pilot symbols.

Minimizing Eq. (2) can be done by multiplying $\boldsymbol{\Lambda}(m)[\boldsymbol{A}(m)\,\boldsymbol{d}(m)]$ by a unitary matrix $\boldsymbol{Q}(m)$:

$$\boldsymbol{Q}(m)\boldsymbol{\Lambda}(m)[\boldsymbol{A}(m)\,\boldsymbol{d}(m)] = \begin{bmatrix} \boldsymbol{R}(m) & \boldsymbol{p}(m) \\ \boldsymbol{0} & \boldsymbol{v}(m) \end{bmatrix}, \tag{3}$$

where $\boldsymbol{R}(m)$ is an $M \times M$ upper triangular matrix, $\boldsymbol{p}(m)$ is a vector of length M, $\boldsymbol{0}$ is an $(m-M) \times M$ null matrix and $\boldsymbol{v}(m)$ is a vector of length $m-M$. The least-squares estimation of $\boldsymbol{w}$(m) is then given by :

$$\boldsymbol{w}(m) = \boldsymbol{R}^{-1}(m)\boldsymbol{p}(m). \tag{4}$$

Once $\boldsymbol{w}(m)$ has been calculated, it is used to combine the signal of the user :

$$\hat{d}_u(i) = \boldsymbol{w}^H(m)\boldsymbol{u}_u(i). \tag{5}$$

Using the Extended QRD-RLS algorithm described in [14], the recursion can be done by applying QRD to the following extended $(M+2) \times (2M+2)$ matrix :

$$\begin{bmatrix} \tilde{\boldsymbol{R}}(m+1) & \tilde{\boldsymbol{R}}^{-H}(m+1) \\ \boldsymbol{0} & \boldsymbol{v}'(m+1) \end{bmatrix} = \boldsymbol{Q}'(m+1) \begin{bmatrix} \boldsymbol{\Lambda}'\tilde{\boldsymbol{R}}(m) & (\boldsymbol{\Lambda}')^{-1}\tilde{\boldsymbol{R}}^{-H}(m) \\ \tilde{\boldsymbol{u}}^H(m+1) & \boldsymbol{0} \end{bmatrix}, \tag{6}$$

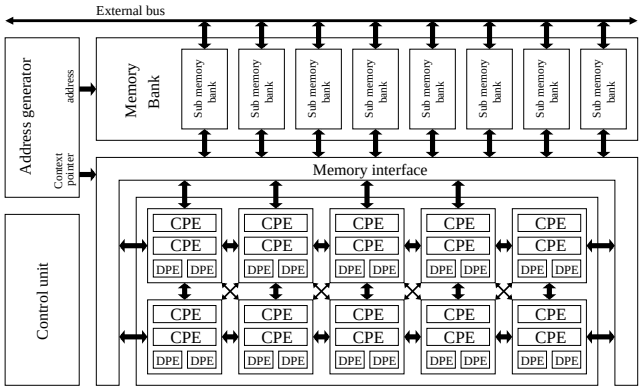


Fig. 3 CORSAEngine architecture.

where $\mathbf{\Lambda}' = diag(\lambda, \ldots, \lambda, 1)$, $\tilde{\boldsymbol{u}}^H(m+1) = [\boldsymbol{u}_p^H(m+1)\, d_p^*(m+1)]$, $\boldsymbol{v}'(m+1)$ is an auxiliary vector and

$$\tilde{\boldsymbol{R}}(m) = \begin{bmatrix} \boldsymbol{R}(m) & \boldsymbol{p}(m) \\ \boldsymbol{0} & \alpha(m) \end{bmatrix}, \tag{7}$$

where $\alpha(m+1)$ is a scalar. After the matrix $\boldsymbol{Q}'$ has zeroed $\tilde{\boldsymbol{u}}^H(m+1)$, a scaled version of the new weight vector appears in $\tilde{\boldsymbol{R}}^{-H}(m+1)$:

$$\tilde{\boldsymbol{R}}^{-H}(m+1) = \begin{bmatrix} \boldsymbol{R}^{-H}(m+1) & \boldsymbol{0} \\ -\frac{\boldsymbol{w}^H(m+1)}{\alpha(m+1)} & \frac{1}{\alpha(m+1)} \end{bmatrix}. \tag{8}$$

This method has the advantage of avoiding back-substitution which can be very time-consuming if it has to be performed frequently.

3. CORSAEngine Architecture

CORSAEngine's architecture is composed of a 2-dimensional array of processing nodes (PN), a control unit, a memory bank and an address generator which controls the algorithms running on the array. The work presented here was realized on a scaled-down architecture represented in Fig. 3.

This scaled-down version of CORSAEngine has a 2-by-5 array of PNs. Each PN is composed of two CORDIC Processing Elements (CPE) and two Delay Processing Elements (DPE). The CPEs implement the unfolded CORDIC algorithm which allows pipelining. The pipeline is used to implement interleaved threads. Different data sets or even completely unrelated algorithm can be executed in the different threads. The data types supported by the processor are real and complex numbers and rotation angles, which are a subset of real numbers. A complex number is the concatenation of two real numbers. A 20-bit floating point format, consisting of a 16-bit mantissa and a 4-bit exponent is used for real numbers.

The control of the operations on the array is done by a context pointer which is attached to the data by the memory interface when it is sent from the memory to the array. Then, every CPE and DPE possesses an instruction table linking a context pointer to the operation to be done with the incoming data and the destination of the result. The result can be sent to any neighboring PN. PNs have horizontal and diagonal connections. A horizontal connection can hold one complex or two real numbers while a diagonal connection is limited to one real number. As a result, complex data flows can be created in the array, giving an efficient and flexible way to easily implement systolic algorithm.

4. Implementation of QRD-RLS

In this section, the implementation of the Extended QRD-RLS algorithm on the array of CORSAEngine is described. An example of the Extended QRD-RLS systolic array for a 3-tap weight vector is given in Fig. 4. Each non-zero complex-valued coefficient of the matrix $\tilde{\boldsymbol{R}}_{ext} = \left[\tilde{\boldsymbol{R}}(m)\ \tilde{\boldsymbol{R}}^{-H}(m)\right]$ is represented by one cell. This cell holds the coefficient value in its register. Note that the coefficient in the right-bottom corner of the matrix is not needed and hence doesn't require a cell.

4.1 Cell operations

Two main types of cell can be seen. Border cells are placed on the left diagonal and produce the required Givens rotation to nullify the input. Inner cells apply this rotation to their own input and register value. One more distinction can be made between cells holding the coefficients of $\boldsymbol{R}$, $\boldsymbol{p}$ or $\boldsymbol{R^{-H}}$ and the last row containing the scaling factor α and the scaled weights $-\boldsymbol{w}/\alpha$. The former must multiply the coefficient they hold with the forgetting factor between every two input, while the latter don't.

Fig. 5 describes how the operations of the cells composing the array can be implemented using CORDIC units in vectoring (VEC), rotation (ROT) and multiplication mode. The two stages of the complex givens rotation are referred to as θ-VEC/ROT and ϕ-VEC/ROT. In the cells of normal rows, the forgetting factor λ must be applied to the register value after every input. However, as the input of the ϕ-VEC/ROT depends on the output of the CORDIC unit applying the forgetting factor, those two operations cannot be pipelined. As a result, the ϕ-VEC/ROT can only operate every two cycle. If the same CORDIC is used for both the ϕ-VEC/ROT and the multiplication by λ, it is fully utilized. But on the other hand the CORDIC used for the θ-VEC/ROT will only be used every two cycles thus wasting half of this resource. As a solution, the same CORDIC can be time-shared by two adjacent cells for their θ-VEC/ROT as illustrated in Fig. 6. The left cell first receives its input and the angle θ, apply the latter to the former and send the result down to its ϕ-VEC/ROT unit. However, the angle θ is is not sent further but stored in a register of the CORDIC unit. In the next

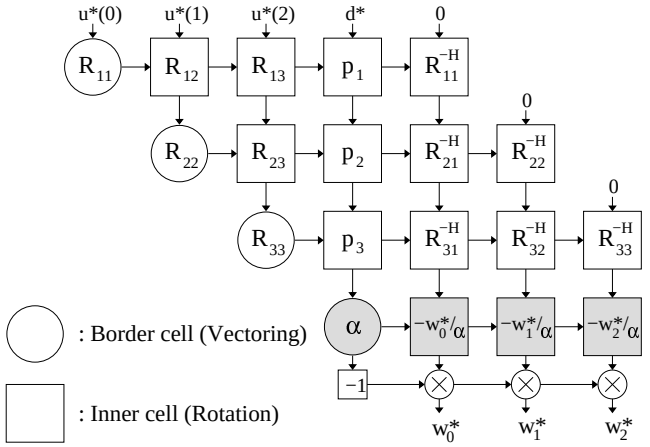


Fig. 4 A systolic array for the production of a 3-tap weight vector using Extended QRD-RLS. Border cells doing complex vectoring and inner cells doing complex rotation are represented respectively as round and squared cells. A distinction is made between cells that must apply the forgetting factor, in white, and the ones that don't, in gray.

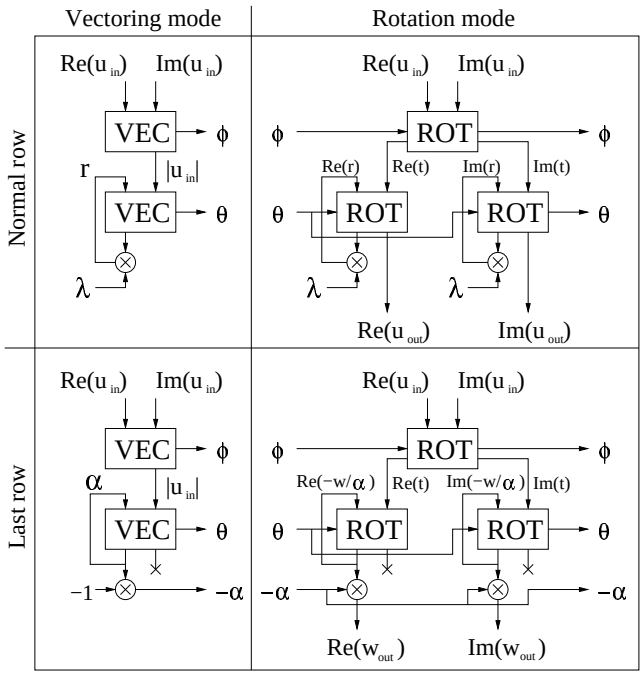


Fig. 5 The CORDIC implementation of the different cells composing a systolic array for Extended QRD-RLS. The multiplication present are also implemented with CORDIC units using a multiplication opcode. the values r and λ are contained in the registers of the CORDIC units.

cycle, the same CORDIC unit receives only the input of the right cell. It will then reuse the angle stored to rotate the input before sending the result down to the ϕ-ROT unit of the right cell. This time, θ is not stored but sent to the next cell on the right.

As the cells from the last row don't apply the forgetting factor, it allows the two CORDIC operations to be fully pipelined. Therefore, successive cells can be connected to each other in a straightforward manner and no time-sharing of CORDIC units is required. And, as the registers of the cells contain a scaled version of the desired weights and the scaling factor α, it is possible, by adding one multiplication to each cell, to scale the weights before they are output. The structure of those cells is also illustrated in Fig. 5.

4. 2 Partitioning

Now that the cell operations have been mapped to CORDIC units, it is possible to use them to construct a full size array for the production of an M-tap weight vector. Such an array has $M^2 + 3M + 1$ cells, each using from 3 to 5 CORDIC units depending on its type. Consequently it has to be divided into smaller partitions that will be successively run on the PN array of CORSAEngine. Because of the strong vertical dependency in the Extended QRD-RLS array, it is first divided into rows, each row having $M + 2$ cells except the last one with $M + 1$ cells. To make it fit on the PN array, these rows still have to be subdivided into segments of a few cells as shown in Fig. 7. Each of these segments contains 7 cells for a normal row and 3 cells for the last row. Considering a single row there are two types of segments: one with a border cell at the beginning and one containing only inner cells that will be respectively referred to as border and inner segments. In conclusion we have 4 partition types, **T**, **X**, **Y** and **Z**, with respectively **T** and **X** referring to border and inner segments of a normal row and **Y** and **Z** to border and inner segments of the last row. Each partition type is implemented on the array as a specific context pointer.

To run the complete algorithm, it is first assumed that the matrix $\tilde{\boldsymbol{R}}_{ext}$, as well as the N new pilots received along with their local copies in the form of the matrix :

$$\boldsymbol{U} = \begin{bmatrix} \tilde{\boldsymbol{u}}^H(m+1) \\ \vdots \\ \tilde{\boldsymbol{u}}^H(m+N) \end{bmatrix}, \tag{9}$$

are stored in the memory bank. A flowchart of the algorithm is represented in Fig. 8. The 7 first coefficients of the first row of $\tilde{\boldsymbol{R}}_{ext}$ are loaded into the registers of the appropriate CORDIC units. Then the 7 first columns of $\boldsymbol{U}$ are processed through the array configured as partition **T**. The processed columns, the modified coefficients of $\tilde{\boldsymbol{R}}_{ext}$ and the angles produced are stored back into memory. The next 7 coefficients of $\tilde{\boldsymbol{R}}_{ext}$ are now loaded into the appropriate CORDIC units registers and the next 7 columns of $\boldsymbol{U}$ are processed, this time using a partition type **X** configuration and the angles produced by the partition **T**. Processed columns and register values are sent back to memory at the end of the execution. This step is repeated until all columns of $\boldsymbol{U}$ have been pro-

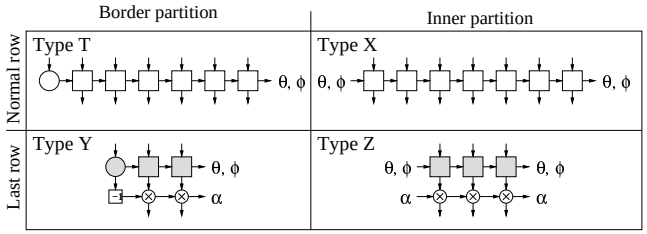


Fig. 7 The four partition types created. With those four types, it is possible to process different array size on the same rectangular systolic array.

Method	MF and Comb.	PS/QRD-RLS	Total
RAKE	200 MFLOPS	0.2 MFLOPS	200 MFLOPS
QRD-RLS	250 MFLOPS	290 MFLOPS	540 MFLOPS

Table 1 Comparison of the computational load of QRD-RLS based interference mitigation with the conventional RAKE receiver.

cessed. After this, a new matrix $\boldsymbol{U}'$ has actually replaced $\boldsymbol{U}$ in the memory. Now for the second row of the array the whole process is repeated using $\boldsymbol{U}'$ and the second row of $\tilde{\boldsymbol{R}}_{ext}$. Eventually, all the rows of the array are processed in the same way, only for the last row, types **Y** and **Z** replace types **T** and **X** and the number of columns processed at a time is only 3. The outputs of the last rows are the N weight vectors corresponding to the N rows of the input matrix $\boldsymbol{U}$.

5. Performance Evaluation

5.1 Simulation Results

Simulation of a WCDMA downlink were carried out to determine the necessary length M of the weight vector $\boldsymbol{w}$. The simulated transceiver used is compliant to current 3GPP standards [1], [2]. Perfect pulse shaping, perfect synchronization and no power control were assumed. The channel model used is the Typical Urban channel from [3]. Simulations were run with coherence time of 1 frame and then 5 slots, corresponding respectively speeds of 13 km/h and 40 km/h. As shown in Fig. 9, a length $M = 16$ is found to be sufficient to outperform the RAKE by as much as an order of magnitude at high Signal-to-Noise Ratio.

5.2 Computational Load

To highlight the cost of the performance gain brought by QRD-RLS based interference mitigation, its computational load is compared to the one of the conventional RAKE receiver. QRD-RLS uses the 10 and 8 pilots per slot, present respectively in CPICH and the user data channel [1], to compute a 16-tap weight vector. A 10 fingers RAKE receiver is considered for comparison. It uses the method described in [8] to obtain a channel estimate with a resolution of 16 paths. It is assumed that synchronization has already been performed at this stage. Complexity of QRD of an $m \times n$ matrix where $m > n$ is given by $3n^2(m - n/3)$ [10]. As shown in Table 1, the complexity of the QRD-RLS based method is more than 2.5 times the one of the RAKE . The main difference comes from the QRD-RLS algorithm which is computationally intensive compared to the insignificant amount of computation required by the path search in the RAKE . However, it is shown in the following sections that using the implementation introduced in Section 4., this complexity can be easily handled by CORSAEngine.

MF Pilot	MF Data	QRD-RLS	Combining	**Total**
4%	6.25%	1.15%	0.35%	**11.75%**

Table 2 The detail of the resource consumption of the different steps of QRD-RLS based interference mitigation on CORSAEngine.

5.3 Resource Usage

In this section, the resource usage of QRD-RLS will be calculated. As shown in Section 5.1, a 16-tap weight vector is sufficient to efficiently mitigate interference. Using the implementation as described in Section 4.2, it is possible to construct an array for the calculation of a 19-tap weight vector which is therefore sufficient to efficiently mitigate the interference. Taking into account the matched filtering of pilot and data channel as well as the combining, QRD-RLS interference mitigation for WCDMA consumes 11.75% of the resources of the scaled-down version of the CORSAEngine. The resource consumptions of the different blocks of the interference mitigation are detailed in Table 2.

5.4 Benchmark

The performance of the implementation of QRD-RLS on CORSAEngine will now be compared to other implementations on different architectures. The architectures considered for comparison are : two dedicated hardwares for QRD-RLS, based on designs conducted on respectively Altera Stratix [5] and Xilinx Virtex-4 [7] FPGAs, and an Application Specific Instruction set Processor (ASIP) for matrix computations (QRD, SVD) using an array of modified CORDIC units [13].

The performance metric used to compare those architectures is the computational density defined as :

$$\rho_{m\times n} = \frac{1}{t_{m\times n} \times \sum_i u_i \times A_i}, \tag{10}$$

where $t_{m\times n}$ is the processing time for a complex matrix of size $m \times n$ in seconds [s], A_i and u_i are respectively the chip area in [Kgates] and a resources utilization factor. The index i accounts for architectures with totally independent parts. To make a fair comparison, the CORSAEngine implementation is adapted to the matrix sizes that were used for evaluating performance of the referred architectures [5], [7], [13]. The performance of the CORSAEngine is furthermore used to normalize the results.

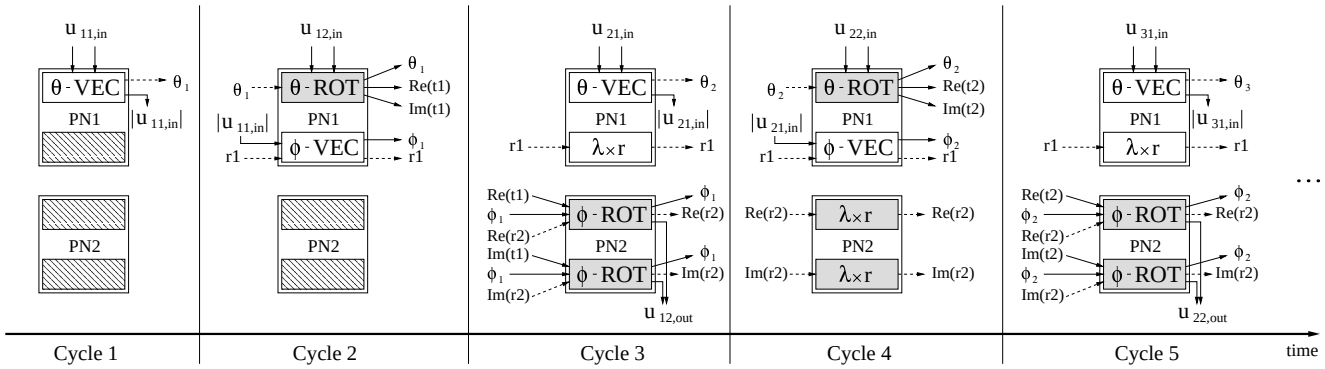


Fig. 6 An example of a border (vectoring) and an inner (rotation) cell on a normal row sharing a CORDIC respectively for the vectoring and rotation of their inputs. The CORDIC is used in vectoring mode during the odd cycles and in rotation mode during the even cycles. Dashed lines represent values that are kept in a register.

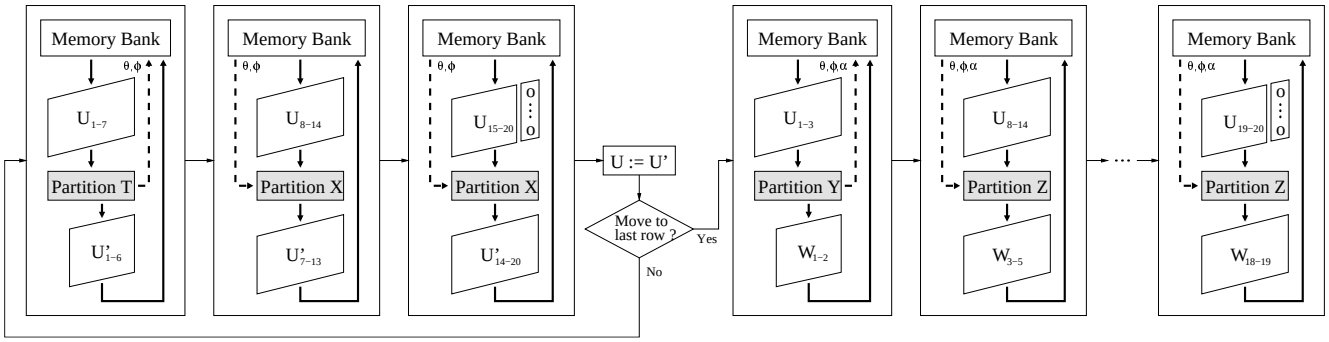
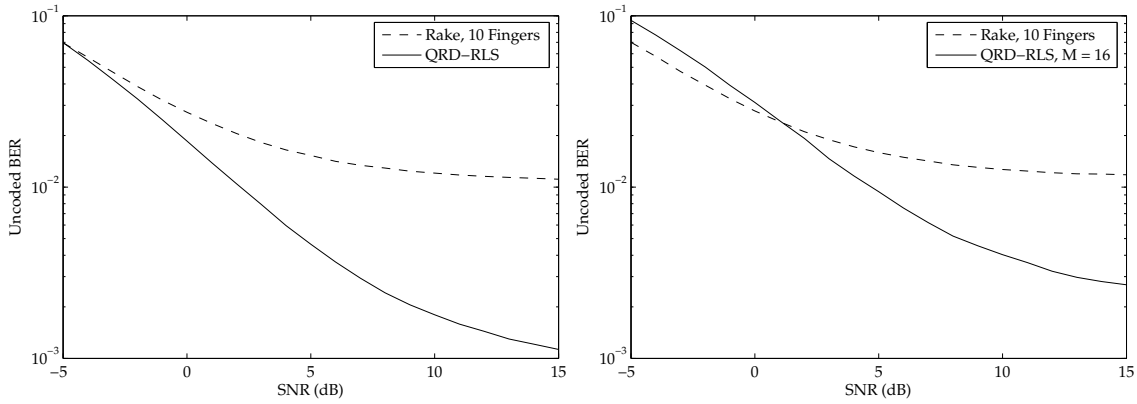


Fig. 8 A run of the algorithm for a 19-tap weight vector. The gray rectangle represents the array of CORSAEngine, the dashed line is for angles and scaling factor that return to memory. $U_{i\text{-}j}$ is the matrix composed of the $i^{\text{th}}$to the j^{th}columns of U. The matrix W output by the last row contains all the weight vectors produced by the processing of the matrix U through the Extended QRD-RLS systolic array.



(a) 13 km/h (b) 40 km/h

Fig. 9 Performance of QRD-RLS in quarter system load (4 users) with a spreading factor of 16.

Table 3 shows the results of the benchmark. The area estimation of the dedicated hardwares was based on the number of lookup tables used in the FPGA design. The corresponding number of gates was estimated according to the available literature [12], [19]. The Altera Stratix design uses two CORDIC blocks for the QRD and the Embedded Nios Soft processor for the back-substitution. The performance of the latest version of the Nios (II) were used [4]. In the case of the ASIP, as it only handles real-valued QRD-RLS, the fact that a 128×20 real-valued matrix can be used to represent a 64×10 complex-valued matrix is used. For CORSAEngine, a utilization factor is introduced as an input matrix with 10 columns such as the ones used in the benchmark only use 80.2% of the resource available.

The result of the benchmark shows that CORSAEngine achieves respectively 50% and 80% more computational density than the dedicated hardware II (based on Xilinx design) and the ASIP processor. The dedicated hardware I (based on

	Ded. Hardware I [5]	Ded. Hardware II [7]	ASIP [13]	CORSAEngine	CORSAEngine
Clock frequency [MHz]	170	250	300	300	300
Matrix size	64×10	10×10	64×10	64×10	10×10
$\boldsymbol{t_{m \times n}}$ [μs]	268.67	56.76	7.04	10.63	2.89
$\boldsymbol{A}$ [gates]	33480	95310	7M	1150K	1150K
Utilization factor	100%	100%	100%	80.2%	80.2%
$\boldsymbol{\rho}$ [update/s/Kgates]	111.22	184.85	20.29	102	375.17
Normalized to CORSA	**109.04%**	**49.27%**	**19.8%**	**100%**	**100%**

Table 3 Performance of the different architectures in terms of the computational density ρ. The final result is normalized in terms of the performance of CORSAEngine to give a fair comparison when the matrix sizes used are different.

Altera design), on the other hand, achieves 9% more computational density. However, it should be noted that the dedicated hardware I (as well as dedicated hardware I and the ASIP) implements the weight extraction as back substitution. It was assumed in this benchmark that the weight are only extracted once after QRD has been done. However, the CORSAEngine implementation, as it uses the Extended QRD-RLS as described in Section 2. 2, output one weight vector after every input row in any case. It will therefore achieve better performance in term of interference mitigation when the coherence time of the channel is very short.

6. Conclusion

In this paper a new implementation of QRD-RLS interference mitigation for WCDMA on CORSAEngine has been presented. First the necessary complex Givens operations were mapped to the available CORDIC units in a way that maximize the utilization of resources. Then the Extended QRD-RLS systolic array was split into manageable sizes that fit on the PN array of CORSAEngine. Simulations were furthermore used to determine the necessary size of the weight vector to be about 19 taps. Finally, the performance of this implementation was compared to other available architectures for QRD-RLS and it was shown to achieve at least 91% of the dedicated hardware performance in terms of computational density. In conclusion, CORSAEngine was shown to be able to handle computationally intensive but efficient interference mitigation algorithm for WCDMA using only 11.75% of its resources.

Acknowledgments This work was realized between March 2007 and January 2008 while the first author was an internship student at the System IP Core Laboratory, NEC Corporation.

References

[1] 3GPP TS 25.211 V7.1.0, "Physical channels and mapping of transport channels onto physical channels (fdd)," 2007.

[2] 3GPP TS 25.213 V7.1.0, "Spreading and modulation (fdd)," 2007.

[3] 3GPP TS 25.943 V6.0.0, "Deployment aspects," 2004.

[4] Altera, "Nios II performance benchmark." Altera Data Sheet, 2007.

[5] D. Boppana, K. Dhanoa, and J. Kempa, "FPGA based embedded processing architecture for the QRD-RLS algorithm," Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on, pp.330–331, 20-23 April 2004.

[6] G. Bottomley, T. Ottosson, and Y.P. Wang, "A generalized rake receiver for interference suppression," Selected Areas in Communications, IEEE Journal on, vol.18, no.8, pp.1536–1545, Aug 2000.

[7] C. Dick, F. Harris, M. Pajic, and D. Vuletic, "Real-Time QRD-Based Beamforming on an FPGA Platform," Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on, pp.1200–1204, Oct.-Nov. 2006.

[8] S. Fukumoto, K. Okawa, K. Higuchi, M. Sawahashi, and F. Adachi, "Path search performance and its parameter optimization of pilot symbol-assisted coherent RAKE receiver for W-CDMA mobile radio," IEICE Trans. Fundamentals, vol.E83-A, no.11, pp.2110–2119, November 2000.

[9] S. Fukumoto, M. Sawahashi, and F. Adachi, "Matched filter-based RAKE combiner for wideband DS-CDMA mobile radio," IEICE Trans. Commun., vol.E81-B, no.7, pp.1384–1391, July 1998.

[10] G.H. Golub and C.F. Van Loan, Matrix Computations, 3 ed., Johns Hopkins, 1996.

[11] S. Haykin, Adaptive Filter Theory, 4 ed., Prentice Hall, 2002.

[12] H. Krupnova and G. Saucier, "FPGA technology snapshot: current devices and design tools," Rapid System Prototyping, 2000. RSP 2000. Proceedings. 11th International Workshop on, pp.200–205, 2000.

[13] Z. Liu, K. Dickson, and J. McCanny, "Application-specific instruction set processor for SoC implementation of modern signal processing algorithms," Circuits and Systems I: Regular Papers, IEEE Transactions on, vol.52, no.4, pp.755–765, April 2005.

[14] T.Z. Mingqian, A.S. Madhukumar, and F. Chin, "QRD-RLS Adaptive Equalizer and its CORDIC-based Implementation for CDMA Systems," International Journal on Wireless & Optical Communications, vol.1, no.1, pp.25–39, 2003.

[15] J. Mitola III, "Software radios-survey, critical evaluation and future directions," Telesystems Conference, 1992. NTC-92., National, pp.13/15–13/23, 19-20 May 1992.

[16] A.F. Molisch, Wireless Communications, IEEE Press, 2005.

[17] K. Seki, T. Kobori, J. Okello, and M. Ikekawa, "A CORDIC-Based Reconfigrable Systolic Array Processor for MIMO-OFDM Wireless Communications," Signal Processing Systems, 2007 IEEE Workshop on, pp.639–644, 17-19 Oct. 2007.

[18] M. Steer, "Beyond 3G," Microwave Magazine, IEEE, vol.8, no.1, pp.76–82, Feb. 2007.

[19] Xilinx, "An alternate capacity metric for LUT-based FPGAs." Xilinx Application Brief, 1997.