Efficient Fixed-Point Implementation of Linear Equalization for Cooperative MIMO Systems

Matthias Mehlhose and Stefan Schiffermüller
Fraunhofer Institute for Telecommunications
Heinrich-Hertz-Institut (HHI)
Einsteinufer 37, 10587 Berlin, Germany
{Matthias.Mehlhose, Stefan.Schiffermueller}@hhi.fraunhofer.de

***Abstract*—We describe an efficient fixed-point calculation of the weight matrices for a linear MMSE equalizer in a MIMO-OFDM system intended for the large numbers of antennas typical for cooperative base stations. Our real-time implementation is numerically stable also in critical channels with reduced rank. Implementation loss due to the 16 bit fixed-point arithmetic is noticeable only in high SNR range above 30 dB. A speed optimization technique is proposed splitting the algorithm in several dot products which can be highly optimized. Using 3 parallel C64x+ DSP cores running at 1 GHz each, equalization of a 12×12 MIMO-OFDM system with 1200 subcarriers can be realized in less than 1 millisecond. Compared to a floating-point implementation, at the same performance power consumption can be reduced by a factor of 6.**

I. Introduction

Modern radio standards like 3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE) require complex and complicated signal processing. Efficient but flexible solutions are therefore increasingly important. On the other hand, low power consumption is desired both at the base station and at the mobile terminal. Fixed-point DSPs meet both these requirements. A further adaptation of the algorithms is needed however to cope with the reduced dynamic range of 16 or 32 bit fixed-point numbers. This is in particular critical for lower layer processing functions as the calculation of MIMO equalizer weights.

Future extensions of 3GPP LTE are related to increasing the number of antenna signals which can be processed jointly, combating the crucial inter-cell interference within one cell [1] [2] or by using cooperative signal processing of multiple base stations [3] [4]. Both lead to larger size of the MIMO channel matrices and thus to higher computational complexity. To show the technical feasibility of such LTE extensions real-time solutions with high performance are of substantial interest.

The calculation of the MMSE equalizer matrix investigated in this paper is the most complex part of the signal processing chain in a MIMO-OFDM system, in particular if the numbers of antennas get larger. This matrix may be calculated e.g. by means of a matrix inversion and by QR or Cholesky decompositions. Common criteria for algorithm selection are complexity, numerical stability and dynamic range of the intermediate results in the calculation. The latter point is specifically important for fixed-point implementations. In [5] a block wise analytic matrix inversion is used. This algorithm becomes prohibitively complex for large channel matrices. The Gauss Jordan inversion in [6] is the better choice in this case and it requires $2n^3$ complex multiplications [7]. However, the inversion of the channel covariance matrix used in both methods leads to numerical stability problem due to the high dynamic range typical for wireless fading channels.

The square-root-algorithm chosen for FPGA implementation in [8] is more stable and it has a slightly lower complexity of $\frac{11}{6}n^3$ as well [7]. For the QR decomposition the authors have chosen a modified Gram-Schmidt method showing stability comparable to the Givens method [9].

An algorithm with even lower complexity of $\frac{5}{3}n^3$ based on Cholesky decomposition is suggested in [7]. It has been implemented in real-time on the IBM Cell processor in [10] where the calculation of all equalizer matrices for the 12×12 MIMO-OFDM system with 1200 subcarriers (LTE) takes less than 1 ms. With the Cell processor we can get a high floating-point performance for little money. But the power efficiency can be improved using fixed-point arithmetic.

In this paper, we report on a fixed point implementation of this algorithm achieving comparable speed performance on 3 C64x+ DSP cores running at 1 GHz each. We have limited the implementation losses due to the fixed-point arithmetic and applied a speed optimization technique that meets both efficiency and flexibility. In addition to the algorithms reported in the literature, we have taken different noise powers at each receive antenna into account to cope with independent automatic gain control (AGC) circuits. This allows the use of distributed radio frontends having autonomous AGC. The paper is organized as follows. In Section II, our system model is explained. Section III describes in detail the implemented algorithm. The implementation on the DSP is reviewed in section IV. The efficiency of our implementation is quantified in section V.

II. System model

The MIMO-OFDM system described in the 3GPP LTE standard has 1200 subcarriers in 20 MHz bandwidth. We consider N_T transmit and N_R receive antennas. For each subcarrier i the transmission is described by

$$y(i) = H(i) \cdot x(i) + n(i), \tag{1}$$

where $H(i)$, $x(i)$, $y(i)$, $n(i)$ are the $N_R \times N_T$ channel matrix, the $N_T \times 1$ transmit symbol, the $N_R \times 1$ receive symbol, the

$N_R \times 1$ noise vector at subcarrier i, respectively. Ignoring the index i the linear MMSE equalisation is performed by

$$\hat{x} = H^H(HH^H + N)^{-1} \cdot y, \tag{2}$$

where N is the noise covariance matrix, $\hat{x}$ is the estimated transmit vector. If N is diagonal with equal diagonal entries σ^2 the following formula is equivalent:

$$\hat{x} = (H^H H + \sigma^2 I)^{-1} H^H \cdot y, \tag{3}$$

where I is the identity matrix. The advantage of the second approach is the reduced dimension of the matrix to be inverted if $N_R > N_T$ and thus a lower computational complexity. To include independent AGC settings in each receiver we assume a different noise power at each antenna. Therefore we have to use (2), but for $N_R > N_T$ the noise must not be set to zero to ensure that the matrix is invertible. For each subcarrier, accordingly, the following MMSE equalizer coefficient matrix is calculated.

$$G = H^H(HH^H + N)^{-1} \tag{4}$$

III. Algorithm

A. Principle

After calculating the positive definite matrix A the Cholesky decomposition is applied:

$$HH^H + N = A = LL^H \tag{5}$$

where L, L^H are the lower and its symmetric upper triangular matrix, respectively. From (4) we get

$$G = H^H(LL^H)^{-1} \tag{6}$$

$$GLL^H = H^H \tag{7}$$

$$LL^H G^H = H \tag{8}$$

The following 2 systems of equations must be solved:

$$LS = H \tag{9}$$

$$L^H G^H = S \tag{10}$$

Since L and L^H are triangular, this is easily achieved by forward substitution to get S in the first equation and subsequent back substitution in the second equation to get G.

B. Fixed-Point Extensions

Before calculating A, the rows of H and the noise matrix N are scaled by the elements in the diagonal matrix B containing the reciprocal square roots of the diagonal elements in A. This operation results in a scaled version of A denoted as $\tilde{A}$ having all elements less or equal to 1, as required in our fixed-point implementation. The corresponding equations read

$$\tilde{H} = BH \tag{11}$$

$$\tilde{N} = B\left(N + \sigma_a^2 I\right)B \tag{12}$$

The additional term $\sigma_a^2 I$ ensures the positive definiteness in (5) and stabilizes the algorithm at high SNR. The above algorithm is applied using $\tilde{H}$ and $\tilde{N}$ resulting in a column-scaled matrix $\tilde{G}$ yielding the desired result

$$G = \tilde{G}B \tag{13}$$

C. Fixed-Point Arithmetic

After scaling the absolute values of all matrix elements in $\tilde{H}$,$\tilde{A}$,$\tilde{L}$,S are less or equal than unity. With a fixed word length of 16 bit we are able to use the most commonly used fraction format Q0.15 [11] to store the real and imaginary part. This format contains 0 and 15 bits before and after the decimal point and an additional sign bit. An advantage of the Q0.15 format is that the format can be retained unchanged after a multiply operation by a simple shift operation.

$$Q0.15 \cdot Q0.15 \Rightarrow Q0.30 =^{15\ bit}_{shift} \Rightarrow Q0.15$$

To avoid overflows in the backward substitution algorithm we change the fractional number format of the matrix S from Q0.15 into Q5.10 with a 5 bit shift after the forward substitution. The scaled coefficient matrix $\tilde{G}$ then has the same fractional number format Q5.10 as well.

$$Q5.10 \cdot Q0.15 = Q5.25 \Rightarrow^{15\ bit}_{shift} \Rightarrow Q5.10$$

The necessary reciprocal square root values for the scaling matrix B and inside the Cholesky decomposition will be stored in the Q16.15 format with a word length of 32 bit. The multiplication with these factors results again in the Q0.15 format:

$$Q16.15 \cdot Q0.15 \Rightarrow Q16.30 =^{15\ bit}_{shift} \Rightarrow Q16.15 =^{16\ bit}_{LSB} \Rightarrow Q0.15$$

D. Cholesky Decomposition

We adapted the cholesky algorithm such that a dot-product is calculated in the inner loop. Due to the symmetry only the lower triangular matrix of A is used. The reciprocal values in forward and backward substitution can be saved, because they correspond to the factors $scale_i$.

Algorithm 1 Pseudo-code of used Cholesky decomposition

$L \leftarrow \begin{cases} A_{ij}, & \text{for } i \geq j \\ 0, & \text{for } i < j \end{cases}$
for $i = 1$ to N_R **do**
  for $j = i$ to N_R **do**
    $sum_j \leftarrow 0$
    for $k = 1$ to $i - 1$ **do**
      $sum_j \leftarrow sum_j + L_{jk} \cdot L^*_{ik}$
  for $j = i$ to N_R **do**
    $L_{ji} \leftarrow L_{ji} - sum_j$
    $scale_i \leftarrow 1/\sqrt{L_{ii}}$
  for $j = i$ to N_R **do**
    $L_{ji} \leftarrow L_{ji} \cdot scale_i$

IV. Optimization for a C64x+ DSP core

A. The C64x+ DSP core

The C64x+ core [12] by Texas Instruments has 4×2 processing units. They are designed for 4 different numeric skills, i.e. memory access, multiplication, accumulation and logic. Thus one very long instruction word (VLIW) can contain 8 instructions to be performed in parallel once per

CPU cycle. Although the execution can take several cycles every cycle a new VLIW can be scheduled due to pipelining. There are 64 bit memory instructions as well as instructions for multiplication and addition of complex numbers. The real and imaginary parts are placed in the most and least significant half words in the 32 bit register.

B. Optimization method

Our initial C-implementation has been far away from exploiting the limit of having 2 complex multiplications per cycle feasible with the C64x+. A highly-optimized assembly implementation of the entire algorithm would be very expensive, in particular if it shall support variable matrix dimensions typical for adaptive MIMO applications. Therefore we have developed a technique splitting all our partial algorithm tasks like matrix multiplications, Cholesky decomposition, forward and backward substitution into highly optimized dot products called from the main program. For a small number of multiply and add operations the overhead is relatively large caused by the function call as well as the prologue and epilogue of the program loop. This overhead can be reduced if the assembly function is able to calculate many dot products within one function call. The function should remain flexible with respect to the number of dot products, the number of vector elements and its memory distance. Let's assume that a and b are 3-dimensional arrays while the result r has 2-dimensions. The common algorithm is extended to perform $n_3 \times n_2$ dot products instead of only one:

Algorithm 2 dot product with an optional parameter $[^*]$ for complex conjugating b

for $i = 1$ to n_3 **do**
  for $j = 1$ to n_2 **do**
    $r_{ij} \leftarrow 0$
    for $k = 1$ to n_1 **do**
      $r_{ij} \leftarrow r_{ij} + a_{ijk} \cdot b_{ijk}[^*]$

We use the following 14 parameters:

parameter	function
a, b, r	Arrays
n_1, n_2, n_3	Dimension of the Arrays
d_{1a}, d_{1b}	Memory distance of a_{ijk} and a_{ijk+1}. Analogously b
d_{2a}, d_{2b}, d_{2r}	Memory distance of a_{ijk} and a_{ij+1k}. Analogously b, r
d_{3a}, d_{3b}, d_{3r}	Memory distance of a_{ijk} and a_{i+1jk}. Analogously b, r
$conjugate$	true if b complex conjugated

Notice that in principal, the MIMO-OFDM algorithm allows parallel processing of any group of subcarriers which makes our optimization method very efficient.

C. Implementation

Flexibility with respect to the parameter n_2 was not possible without compromise concerning speed. We have optimized the function for $n_2 = 12$ being the resource block (RB) size defined in the LTE standard. I.e. groups of 12 subcarriers of a RB are treated in the same function call. Our assembly program performs n_2 dot products in parallel by only one program loop using n_2 accumulators accodingly. The parameter n_3 exploits the inherent parallelism in the matrix multiplication, Cholesky decomposition, forward and backward substitution. In case of Cholesky decomposition we set this parameter to $n_3 = N_R$ and applied it to the outer loop (row 3) of the dot product (row 5) in Algorithm 1. Operating at full load, the following set of instructions is performed in each cycle:

instruction	function
2 x LDDW	2 x 64 bit load, 2 pairs (2 subcarriers) of operands a,b
2 x SWAP2	Exchange real and imaginary part of a if conjugate = true
2 x CMPYR1	Complex multiplication of 2 pairs a,b then 15 bit right shift
2 x SADD2	Accumulation of the complex results with saturation on overflow

To make use of the SWAP2 instruction we could replace the complex conjugate variant of Algorithm 2 by

Algorithm 3 dot product with SWAP2 for complex conjugating b

$r \leftarrow 0$
for $k = 1$ to n **do**
  $r \leftarrow r + \text{SWAP2}(a_k) \cdot b_k$
$r \leftarrow \text{SWAP2}(r)$

The inner loop is performed n_1 times. Afterwards, the $n_2 = 12$ results are stored using 6 STDW instructions. If the variable conjugate is true, a SWAP2 instruction is applied before. Then the next group of dot products is calculated (outer loop). The entire function executes $n_3 \cdot n_2 \cdot n_1$ complex multiplications. If we consider additional cycles for saving results and add the constant function call overhead of 15 then we get the achieved performance of our optimized assembly routine $n_3 \cdot \frac{n_2}{2} \cdot \left(n_1 + \frac{1}{2}\right) + 15$ cycles.

The transfers from and to data cache memory take place in the background using an internal direct memory access (IDMA) controller. Thus additional cycles caused by missing data in the cache do not occur.

D. Reciprocal square roots

The calculation of one equalizer coefficient matrix requires the calculation of $2N_R$ real-valued reciprocal square roots $(1/\sqrt{\ })$, where N_R of which are calculated in the scaling part and another N_R roots occur in the Cholesky decomposition reused by the back and forward substitution. To calculate $1/\sqrt{b}$ we execute one Newton-Raphson iteration which doubles the precision of the initial estimate x:

$$x = x\left(1.5 - \frac{bx^2}{2}\right) \tag{14}$$

For the initial reciprocal square root estimate x we use a constant approximation lookup table of length 256×16 bit acording to [13] Table II. We call $12N_R$ times the $(1/\sqrt{\ })$-function in a loop. The compiled code becomes very efficient in this way.

E. Saving multiplications

After scaling H the diagonal elements in A are set to unity. In the Cholesky decomposition this means $scale_1 = 1$. It

follows that the multiplication of $scale_1$ by the first column in L and the first row in S and G can be saved. The conjugate complex transpose operation and the back scaling of G is performed in one step.

V. Results

A. Complexity and power consumption

Table I gives an overview of the complexity needed on a single RB. The number of complex multiplications, the CPU cycles and the time used for 12 MMSE equalizer matrices (1 chunk in LTE) with various matrix dimensions are given. For matrix dimension 12×12, Table II lists the values for all partial algorithms separately. The fixed-point computation includes the scaling procedures for the matrices H, S and G. Interestingly the scaling of H saves multiplications in the other algorithms, see columns 2, 3.

TABLE I: cycles for twelve equalizer matrices

matrix dimension	complex mult. float	complex mult. fixed	total C64x+ (1 GHz) cpu cycles	total C64x+ (1 GHz) time
4×4	1632	2076	2831	3 us
8×8	11712	13740	11601	12 us
12×12	37920	42684	29916	30 us

TABLE II: detailed view to twelve 12×12 equalizer matrices

part of algoritm	complex mult. float	complex mult. fixed	total C64x+ (1 GHz) cpu cycles	total C64x+ (1 GHz) time
scale H	0	3456	2356	2.4 us
HH^H	11232	9504	5281	5.3 us
Cholesky	4224	4092	4548	4.6 us
forward	11232	11088	7951	8.0 us
scale S	0	1728	878	0.9 us
backward	11232	11088	7972	8.0 us
scale G	0	1728	884	0.9 us
total	37920	42684	29916	30.0 us

TABLE III: complexity

part of algoritm	complex multiplications floating-point	complex multiplications fixed-point
scale H	0	$(2N_R) \cdot N_T$
HH^H	$\frac{1}{2}\left(N_R^2 + N_R\right) \cdot N_T$	$\frac{1}{2}\left(N_R^2 - N_R\right) \cdot N_T$
Cholesky	$\frac{1}{6}\left(N_R^3 + 3N_R^2 - 4N_R\right)$	$\frac{1}{6}\left(N_R^3 + 3N_R^2 - 10N_R + 6\right)$
forward	$\frac{1}{2}\left(N_R^2 + N_R\right) \cdot N_T$	$\frac{1}{2}\left(N_R^2 + N_R - 2\right) \cdot N_T$
scale S	0	$(N_R) \cdot N_T$
backward	$\frac{1}{2}\left(N_R^2 + N_R\right) \cdot N_T$	$\frac{1}{2}\left(N_R^2 + N_R - 2\right) \cdot N_T$
scale G	0	$(N_R) \cdot N_T$

Table III lists the complexity formulas to calculate one MMSE equalizer matrix. The asymptotic complexity of the scaling procedures is only quadratic. So its relative cost decreases with increasing the matrix dimensions.

Multiplying all results by 100, we get results for all 1200 subcarriers in 20 MHz bandwidth. According to these numbers, 3 cores of the C64x+ DSP running at 1 GHz could compute all 1200 MMSE equalizer matrices in less then a millisecond, which is the requirement for a real-time implementation. These values are comparable to the results in [10]. But we have to consider that the authors state that only 4 of 8 Cell cores are used operating at 3.2 GHz and that the calculation of the equalizer matrices is only a part of their computation taking 80% of the entire time. Taking the power consumption of 6 W [14] [15] for a 3 core DSP and 92 W [16] for the Cell into account our implementation is about 6 times more efficient related to power consumption. Note that multi-core variants of the C64x+ are already available [17] which make the system integration easier.

B. Numerical stability

The following subsection is based on the above algorithm as well. It has been implemented in a first step in MATLAB® using the Fixed-Point Toolbox™ in order to evaluate the numerical stability prior to the implementation on the DSP. We assume a Rayleigh fading channel, thus the elements in H are modelled as independently and identically distributed random Gaussian numbers with zero mean. The normalization in (11) assures a constant dynamic range of channel data. Consequently different path losses or AGCs for each receive antenna do not affect numerical stability. Therefore we do not consider this case in our simulation and set the noise covariance matrix to:

$$N = \sigma^2 I \tag{15}$$

where σ^2 is the variance of additive white Gaussian noise (AWGN) in (1). We get σ^2 from a given SNR by means of signal power p_s:

$$\sigma^2 = \frac{p_s}{SNR} \tag{16}$$

$$p_s = \frac{1}{N_R} \cdot trace(H \cdot H^H) \tag{17}$$

For the expectation $E\{.\}$ of the transmit power we assume $E\left\{x^H x\right\} = 1$. Fixed-point arithmetic is used in this analysis only for the core algorithm computing the matrix G. The computed matrix G is multiplied with the received signal vector y (2) yielding the estimated transmit vector $\hat{x}$. The mean square error (MSE) is as follows:

$$MSE = \frac{1}{N_T} \cdot E\left\{\|\hat{x} - x\|_2^2\right\} \tag{18}$$

In all figures 16 QAM modulation is used. We assume a limited 12 bit precision for the channel estimates H both for the real and imaginary parts. Using Q0.15 format the maximum rather minimum number of each row in H is ± 0.0625. In Figure 1 it is demonstrated that there is no significant deviation $(0, 5\%)$ between the fixed-point and perfect computation below an SNR of 30 dB. At larger SNR the MSE has been stabilized by adding the constant $\sigma_a^2 = -57\,dB$ to the noise covariance matrix N. This assures numerical stability also at ill-conditioned channel matrices H appearing some times in our Rayleigh fading model but more often in a realistic scenario. Setting this parameter too small ($\sigma_a^2 = -60\,dB$) the algorithm becomes unstable at high SNR. We see at high SNR that the limited 12 bit precision of channel data increases the MSE of the perfect calculation. But it is sufficient for the fixed point calculation because the MSE is mainly affected by the calculation itself. Figure 2 shows the BER after applying the equalizer matrix to the received signal. No channel coding is

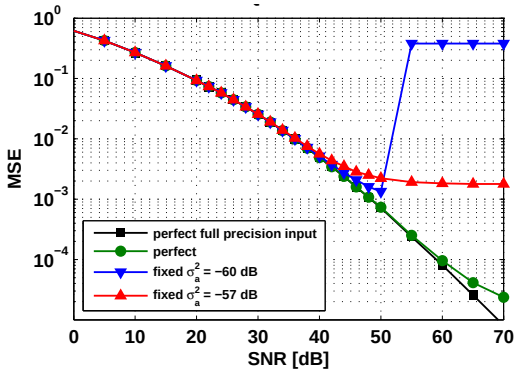

FIG. 1: MSE 12×12, 16 QAM

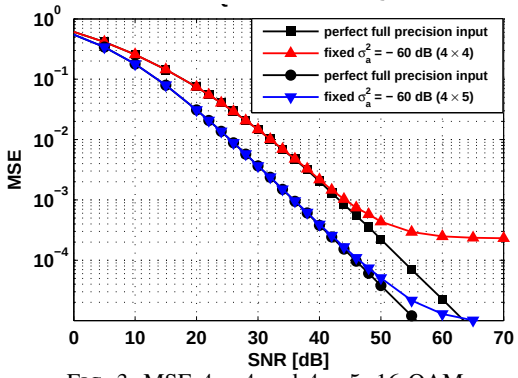

FIG. 3: MSE 4×4 and 4×5, 16 QAM

assumed here. The stabilization has a similar effect in terms of BER performance at high SNR as for the MSE. In Figure 3 an additional receive antenna ($4N_T \times 5N_R$) has been used, we observe that the deviation to the perfect curve is at even higher SNR in this way. The error floor is reduced, accordingly.
All fixed-point results have been calculated by the DSP using the Embedded IDE Link™ CC for TI's Code Composer Studio™ IDE.

VI. CONCLUSION

We have successfully implemented on a state-of-the art fixed-point DSP the channel equalization function for a future cellular system applying cooperative transmission and reception, which is critical both in terms of numerical stability and complexity. Implementation losses are observed only at very high SNR > 30 dB. Note that an error floor is in principle unavoidable because of the finite machine precision and the quantisation of the input data. By splitting the algorithm into several dot products, optimizing this function in assembly language and calling it for multiple subcarriers at once, a similar computing performance can be achieved as in a recent floating-point implementation on the IBM Cell processor using a similar number of processor cores and at a much lower clock speed. Using fixed-point arithmetic, the power consumption has been reduced by a factor of 6, approximately. The results in this paper may be regarded as a first milestone towards proving that cooperative MIMO is mature for next generation cellular systems.

ACKNOWLEDGMENT

The authors wish to thank the German Federal Ministry of Research and Education for financial support in the national research project MxMobile. Furthermore we wish to thank T. Haustein, J. Eichinger and E. Schulz (Nokia Siemens Networks) and V. Jungnickel (HHI) for their continuous interest in this work and Texas Instruments for supplying us with early working samples of a multi-core C64x+ DSP. Moreover, we wish to thank D. Kuehling and A. Ibing for detailed information on the IBM Cell processor implementation.

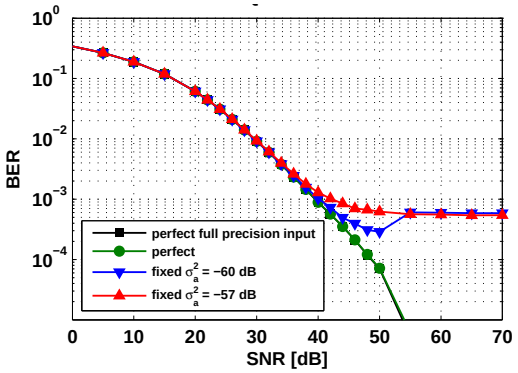

FIG. 2: BER 12×12, 16 QAM

REFERENCES

[1] C. A. Baird and C. L. Zahm, "PERFORMANCE CRITERIA FOR NARROWBAND ARRAY PROCESSING," in *Proc. IEEE CDC 1971*, vol. 10, pp. 564–565.
[2] L. Thiele, V. Jungnickel, M. Schellmann, and W. Zirwas, "Capacity Scaling of Multi-User MIMO with Limited Feedback in a Multi-Cell Environment," in *Conference Record of the Forty-First ACSSC '07*, pp. 93–100.
[3] T. Weber, I. Maniatis, A. Sklavos, E. Costa, H. Haas, and E. Schulz, "Joint transmission and detection integrated network (JOINT), a generic proposal for beyond 3G systems," in *Proc. 9th ICT '02*, pp. 479–483.
[4] H. Huang and S. Venkatesan, "Asymptotic Downlink Capacity of Coordinated Cellular Networks," in *Conference Record of the Thirty-Eighth ACSSC '04*, vol. 1, pp. 850–855 Vol.1.
[5] J. Eilert, D. Wu, and D. Liu, "IMPLEMENTATION OF A PROGRAMMABLE LINEAR MMSE DETECTOR FOR MIMO-OFDM," in *Proc. IEEE ICASSP '08*, pp. 5396–5399.
[6] M. Mckeown, I. Lindsay, D. Cruickshank, J. Thompson, S. Farson, and Y. Hu, "RAPID PROTOTYPING - Re-scalable V-BLAST MIMO system for FPGA," *IEE Proc. VISP '06*, vol. 153, no. 6, pp. 747–753.
[7] D. Kühling, S. Schiffermüller, and A. Ibing, "On Efficient Computation of MMSE Equalizer Matrix for MIMO-OFDM Systems," in *1st Workshop on CMCS '07*.
[8] H. S. Kim, W. Zhu, J. Bhatia, K. Mohammed, A. Shah, , and B. Daneshrad, "AN EFFICIENT FPGA BASED MIMO-MMSE DETECTOR," in *15th EUSIPCO '07*.
[9] A. Björck, "Numerics of Gram-Schmidt Orthogonalization," *Linear Algebra Appl.*, vol. 197–198, pp. 297–316, 1994.
[10] D. Kuhling, A. Ibing, and V. Jungnickel, "12x12 MIMO-OFDM Real-time Implementation for 3GPP LTE+ on a Cell Processor," in *Proc. 14th EW '08*, pp. 1–5.
[11] *SPRUEB8B: TMS320C64x+ DSP Little-Endian DSP Library Programmer's Reference*, Texas Instruments, Mar 2008.
[12] *SPRU732G: TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide*, Texas Instruments, Feb 2008.
[13] N. Nenadio and S. Mladenovic, "Fast Division on Fixed-Point DSP Processors Using Newton-Raphson Method," in *Proc. EUROCON '05*, vol. 1, pp. 705–708.
[14] "Texas Instruments newest high performance multicore digital signal processor integrates three 1 GHz cores on a single chip for DSP farm applications." [Online]. Available: http://focus.ti.com/docs/pr/pressrelease.jhtml?prelId=sc08121
[15] *SPRAAX3: TMS320C6474 Power Consumption Summary*, Texas Instruments, Oct 2008.
[16] *PowerXCell 8i processor product brief*, IBM, May 2008.
[17] *SPRS532: TMS320C6474 Multicore Digital Signal Processor*, Texas Instruments, Oct 2008.