# VLSI Implementation of a Lattice-Reduction Algorithm for Multi-Antenna Broadcast Precoding

*A. Burg*[†], *D. Seethaler*[‡], *and G. Matz*[‡]

[†] Integrated Systems Laboratory, ETH Zurich
8092 Zurich, Switzerland, email: apburg@iis.ee.ethz.ch

[‡] Institute of Communications and Radio-Frequency Engineering, Vienna University of Technology
1040 Wien, Austria, email: {dominik.seethaler, gerald.matz}@tuwien.ac.at

***Abstract*— This paper describes the first VLSI implementation of lattice reduction (LR) aided multi-antenna broadcast precoding with vector perturbation. The considered LR scheme is based on Brun's algorithm for finding integer relations. We analyze its high-level architectural issues, we devise a corresponding low-complexity implementation, and, finally, we develop a suitable VLSI architecture. The resulting circuit provides reference for the true silicon complexity of LR for broadcast precoding with vector perturbation.**

## I. Introduction

Multi-antenna multi-user communication systems employ multiple antennas at the transmitter to concurrently serve multiple non-cooperating single-antenna users [1], [2], [3]. Here, precoding is performed so that each user receives only its intended data stream. With simple linear preequalization the transmit power is increased significantly, which, however, can be partially mitigated by lattice reduction (LR) aided vector perturbation [4], [5], [6]. Unfortunately, many LR algorithms have a considerable computational complexity, rendering the economic VLSI implementation of real-time multi-user communication systems a challenging task.

*Contribution:* This paper shows how LR-aided vector perturbation can be implemented efficiently as an ASIC. To this end, we consider the LR algorithm proposed in [6], that is based on Brun's algorithm for finding integer relations. The algorithm is first streamlined for VLSI implementation and a suitable hardware architecture is presented. The reported implementation results provide reference for the true silicon complexity of LR-aided vector perturbation, which – to the best of our knowledge – has not been described so far in the open literature.

*Outline:* The remainder of this section describes the system model and the LR-aided vector perturbation algorithm under consideration. Section II focuses on the high-level architecture and describes algorithm transformations that reduce the hardware complexity. The details of the proposed register transfer level (RTL) architecture are then described in Section III. Finally, implementation results are presented in Section IV.

### A. System Model

We consider the downlink of a multi-user communication system in which the base station is equipped with $M_T$ antennas and transmits to $K$ single-antenna users (see e.g., [1], [3]). The input output relation of this system is given by

$$\mathbf{y} = \gamma \mathbf{H}\mathbf{x} + \mathbf{n}, \tag{1}$$

where the $K \times M_T$ matrix $\mathbf{H}$ describes the channel and the $M_T$-dimensional vector $\mathbf{x}$ subsumes the complex-valued baseband samples transmitted from the $M_T$ transmit antennas. The normalization factor $\gamma = 1/\|\mathbf{x}\|$ ensures unit transmit power level. Finally, the $K$-dimensional vector $\mathbf{n}$ denotes white complex Gaussian noise.

This work was supported by the STREP project MASCOT (IST-026905) within the Sixth Framework of the European Commision.

### B. LR-Aided Zero-Forcing Precoding

With *zero-forcing (ZF) preequalization*, the vector $\mathbf{x}$ is constructed from the data vector $\mathbf{d}$ according to (e.g., [1])

$$\mathbf{x} = \mathbf{P}\mathbf{d} \quad \text{with} \quad \mathbf{P} = \mathbf{H}^H\left(\mathbf{H}\mathbf{H}^H\right)^{-1}, \tag{2}$$

where the main drawback is an enhancement of the transmit power level, which, after the compensation by the normalization factor $\gamma$, can lead to a significant performance degradation.

*LR-aided vector perturbation* [4], [5], [6] is one approach to alleviate the problem of transmit power enhancement. The basic idea is to find an unimodular matrix $\mathbf{B}$ that is suitable to transform $\mathbf{P}$ into a *better* basis $\tilde{\mathbf{P}} = \mathbf{P}\mathbf{B}$. The vector $\mathbf{x}$ is then chosen according to

$$\mathbf{x} = \mathbf{P}\left(\mathbf{d} + \tau \mathbf{z}\right) \quad \text{with} \quad \mathbf{z} = \mathbf{B}\left\lfloor \frac{1}{\tau}\mathbf{B}^{-1}\mathbf{d} \right\rceil, \tag{3}$$

where $\mathbf{d}$ denotes the original modulated data, $\tau$ is a fixed constant, and $\lfloor\cdot\rceil$ denotes rounding to nearest integer.

### C. Low Complexity LR Algorithm

The challenge of LR-aided ZF precoding lies in the efficient implementation of LR. Here, the LLL algorithm [7] is most widely used in the literature. Unfortunately, its complexity is often prohibitive for real-time implementations. An alternative, low-complexity LR scheme based on Brun's algorithm has been described in [6]. It can be summarized as follows:

1) Initialize $\mathbf{u}^{(0)}$ as an arbitrary column of $\left(\mathbf{H}\mathbf{H}^H\right)^{-1}$ (which is a sideproduct of many matrix-inversion algorithms such as the one used in [8]), $\mathbf{P}^{(0)} = \mathbf{P}$, $\mathbf{B}^{(0)} = \mathbf{I}$, and $\mathbf{C}^{(0)} = \mathbf{I}$.
2) Choose the indices $s$ and $t$ according to

$$s = \underset{k\in\{1,\dots,K\}}{\arg\max} \left|u_k^{(i)}\right|, \quad t = \underset{k\in\{1,\dots,K\}/\{s\}}{\arg\max} \left|u_k^{(i)}\right|. \tag{4}$$

3) Compute

$$\beta^{(i)} = \left\lfloor \frac{u_s^{(i)}}{u_t^{(i)}} \right\rceil = \left\lfloor \frac{u_s^{(i)} u_t^{(i)*}}{|u_t^{(i)}|^2} \right\rceil \tag{5}$$

and update

$$\mathbf{u}^{(i+1)} = \left[u_1^{(i)}, \dots, u_{s-1}^{(i)}, u_s', u_{s+1}^{(i)}, \dots, u_K^{(i)}\right]^T, \tag{6}$$

$$\mathbf{P}^{(i+1)} = \left[\mathbf{p}_1^{(i)}, \dots, \mathbf{p}_{s-1}^{(i)}, \mathbf{p}_s', \mathbf{p}_{s+1}^{(i)}, \dots, \mathbf{p}_K^{(i)}\right], \tag{7}$$

$$\mathbf{B}^{(i+1)} = \left[\mathbf{b}_1^{(i)}, \dots, \mathbf{b}_{s-1}^{(i)}, \mathbf{b}_s', \mathbf{b}_{s+1}^{(i)}, \dots, \mathbf{b}_K^{(i)}\right], \tag{8}$$

$$\mathbf{C}^{(i+1)} = \left[\mathbf{c}_1^{(i)}, \dots, \mathbf{c}_{t-1}^{(i)}, \mathbf{c}_t', \mathbf{c}_{t+1}^{(i)}, \dots, \mathbf{c}_K^{(i)}\right]^T, \tag{9}$$

where $u'_s = u_s^{(i)} - \beta^{(i)} u_t^{(i)}$, $\mathbf{p}'_s = \mathbf{p}_s^{(i)} - \beta^{(i)*}\mathbf{p}_t^{(i)}$, $\mathbf{b}'_s = \mathbf{b}_s^{(i)} - \beta^{(i)*}\mathbf{b}_t^{(i)}$, and $\mathbf{c}'_t = \mathbf{c}_t^{(i)} + \beta^{(i)*}\mathbf{c}_s^{(i)}$.

4) Check the termination condition

$$\|\mathbf{p}'_s\| > \|\mathbf{p}_s\|. \tag{10}$$

If (10) is false, increment $i$ and go to Step 2; otherwise terminate and return $\mathbf{u}^{(J-1)}$, $\tilde{\mathbf{P}} = \mathbf{P}^{(J-1)}$, $\mathbf{B} = \mathbf{B}^{(J-1)}$, and $\mathbf{B}^{-1} = \mathbf{C}^{(J-1)}$ with $J = i + 1$.

## II. System-Level Architecture Considerations

For VLSI implementations it is practical to distinguish between channel-rate preprocessing (all operations, which are performed only when the channel changes) and symbol-rate processing (all operations that are carried out for each transmitted vector-symbol). Since the update-rate of the channel is usually much lower than the symbol rate, it is often argued that iterative decomposition may be used to schedule the computations on only few processing resources to reduce silicon area. However, in MIMO-OFDM systems, where preprocessing has to be performed on a large number of channel matrices, such area-efficient architectural transformations also result in a considerable preprocessing latency, which often cannot be tolerated in packet-based systems [8]. Hence, also the allegedly low-rate preprocessing must often be implemented using fast hardware architectures.

To obtain such an efficient, high throughput architecture we shall start from a straightforward high-level block diagram to restructure the original algorithm [6] to reduce its hardware complexity.

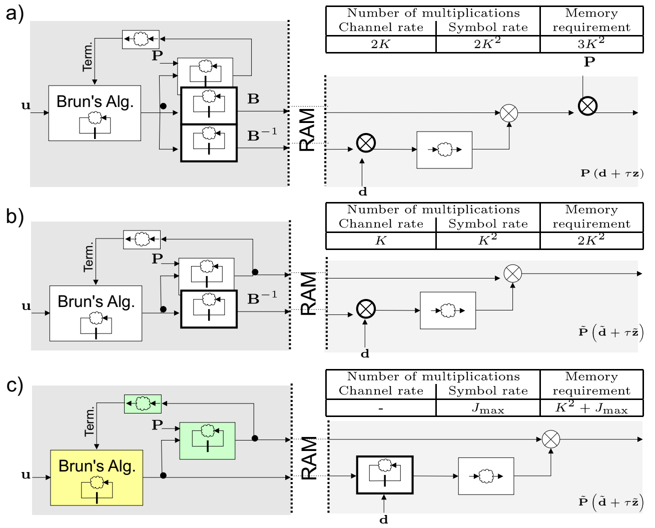
### A. Initial VLSI Architecture

Fig. 1a shows the data dependency graph (DDG) and the complexity of the LR algorithm implemented according to [6]. The channel-rate processing is shown on the left-hand-side of the block-diagram. It is comprised of Brun's Algorithm, the update of the matrices $\mathbf{B}^{(i)}$, $\mathbf{C}^{(i)}$, and $\mathbf{P}^{(i)}$, and of the evaluation of the termination condition (10). Due to the nonlinear recursive structure of the LR algorithm, the highest degree of parallel processing corresponds to the parallel execution of one iteration. The corresponding symbol-rate processing (on the right-hand-side of Fig. 1a) uses $\mathbf{B} = \mathbf{B}^{(J-1)}$ and $\mathbf{B}^{-1} = \mathbf{C}^{(J-1)}$ to compute $(\mathbf{d} + \tau\mathbf{z})$ according to (3) and $\mathbf{P} = \mathbf{P}^{(0)}$ to obtain the transmitted signal vector. Note that here $\mathbf{P}^{(J-1)}$ is only required to evaluate the termination condition.

### B. Restructuring for VLSI Implementation

At first, we streamline the original algorithm to avoid obvious redundant computations. To this end, one can exploit that $\mathbf{P}^{(0)}(\mathbf{d} + \tau\mathbf{z}) = \mathbf{P}^{(J-1)}(\tilde{\mathbf{d}} + \tau\tilde{\mathbf{z}})$ with $\tilde{\mathbf{d}} = \mathbf{B}^{-1}\mathbf{d}$ and $\tilde{\mathbf{z}} = \lfloor \tilde{\mathbf{d}}/\tau \rceil$ and that $\mathbf{P}^{(J-1)}$ must be computed anyway for the evaluation of the termination condition. The DDG in Fig. 1b shows the result of this optimization. Compared to the original implementation strategy, the computational complexity of both channel-rate and symbol-rate processing has been reduced and less memory is required since only two matrices ($\mathbf{C}^{(J-1)}$ and $\mathbf{P}^{(J-1)}$) must be stored while the original unoptimized architecture required storage for three matrices ($\mathbf{B}^{(J-1)}$, $\mathbf{C}^{(J-1)}$, and $\mathbf{P}^{(0)}$).

The second optimization step primarily aims at eliminating redundant multiplications in the update of $\mathbf{C}^{(i)}$ during preprocessing and in the application of $\mathbf{C}^{(J-1)}$ to $\mathbf{d}$ in the symbol-rate processing. The main reason for this redundancy in the preprocessing is that $\mathbf{C}^{(i)}$ is sparse during the first few updates according to (9). However, in a parallel implementation that carries out one update step in each cycle the number of multipliers is determined by the worst case where all $K$ entries of $\mathbf{c}_s^{(i)}$ are unequal to zero. Hence, it is not immediately possible


a)
Term.
P
u
Brun's Alg.
B
$\mathbf{B}^{-1}$
RAM
P
d
$\mathbf{P}(\mathbf{d}+\tau\mathbf{z})$
b)
Term.
P
u
Brun's Alg.
$\mathbf{B}^{-1}$
RAM
d
$\tilde{\mathbf{P}}\left(\tilde{\mathbf{d}}+\tau\tilde{\mathbf{z}}\right)$
c)
Term.
P
u
Brun's Alg.
RAM
d
$\tilde{\mathbf{P}}\left(\tilde{\mathbf{d}}+\tau\tilde{\mathbf{z}}\right)$

| Number of multiplications | | Memory |
|---|---|---|
| Channel rate | Symbol rate | requirement |
| $2K$ | $2K^2$ | $3K^2$ |

| Number of multiplications | | Memory |
|---|---|---|
| Channel rate | Symbol rate | requirement |
| $K$ | $K^2$ | $2K^2$ |

| Number of multiplications | | Memory |
|---|---|---|
| Channel rate | Symbol rate | requirement |
| - | $J_{\max}$ | $K^2 + J_{\max}$ |

Fig. 1. DDGs of various LR architectures with the complexities of the highlighted blocks, affected by the streamlining procedure.

to exploit the sparseness of $\mathbf{C}^{(i)}$ for hardware complexity reduction in the channel-rate processing. A similar argument applies to the application of $\mathbf{C}^{(J-1)}$ to $\mathbf{d}$ in the symbol-rate processing since for a small number of iterations even $\mathbf{C}^{(J-1)}$ is still sparse. However, runtime allocation of processing resources to the nonzero entries of $\mathbf{C}^{(i)}$ breaks the regularity of a matrix-vector multiplication and adds considerable hardware overhead for control. The basic idea for avoiding multiplications with ones and zeros is to apply the transformations in (9) directly to the transmitted vectors $\mathbf{d}$ as shown in the block diagram in Fig. 1c. The corresponding update equation that yields $\mathbf{d}^{(J-1)}$ from $\mathbf{d}^{(0)} = \mathbf{d}$ is given by

$$\mathbf{d}^{(i+1)} = \left[d_1^{(i)}, \ldots, d_{t-1}^{(i)}, d'_t, d_{t+1}^{(i)}, \ldots, d_K^{(i)}\right]^T, \tag{11}$$

where $d'_t = d_t^{(i)} + \beta^{(i)*} d_s^{(i)}$. The drawback of this approach is that the number of multiplications for the symbol-rate processing now depends on the number of iterations $J$. Compared to the implementation in Fig. 1b, the average computational complexity is only reduced if $\mathcal{E}\{J\} < K^2$, where $\mathcal{E}\{\cdot\}$ denotes the expectation and where $K^2$ is the number of multiplications required for the evaluation of $\mathbf{C}^{(J-1)}\mathbf{x}$. Numerical simulations show that $\mathcal{E}\{J\} \approx 2$ for $M_T = K = 4$ which is far below the constant complexity of a straightforward matrix-vector-multiplication based implementation. Unfortunately, the storage between the channel-rate and the symbol-rate processing and the symbol-rate processing unit itself must be designed to deal with the worst-case rather than for the average case. Hence, an artificial upper bound $J_{\max}$ must be enforced on the number of iterations (which is necessary anyway to constrain the preprocessing latency). This constraint should be chosen such that premature termination is a rare event. In a system with $M_T = K = 4$ with i.i.d. Rayleigh fading channel setting $J_{\max} = 8$ is sufficient for 99.9% of all channel realizations, which at the same time still guarantees a 50% complexity and memory reduction when comparing the direct application of $\beta^{(i)}$ to $\mathbf{d}$ to the matrix-vector-multiplication based approach in (3).

## III. RTL Implementation

The RTL implementation, described in the following is based on the high-level architecture depicted in Fig. 1c. Since the implementation of the symbol-rate processing is straightforward, we now focus on the channel-rate processing. The corresponding circuit is partitioned into two main entities:

- The *Brun's algorithm unit* (BAU) receives the vector $\mathbf{u}^{(0)}$ (for example from the circuit described in [8]), carries out *Brun's algorithm* as described by (4)–(6), and forwards $\beta^{(i)}$ to the second entity.
- The *Update* **P** *unit* (UPU) applies $\beta^{(i)}$ to update $\mathbf{P}^{(i)}$, checks the termination condition in (10), and sends the result of this check back to the BAU.

## A. RTL Architecture of BAU

The RTL block diagram of the BAU is shown in Fig. 2. The circuit is partitioned into two macro-pipeline stages:

*1) Initialization Stage:* The first stage (marked in dark-gray in Fig. 2) initializes the iteration. It computes the $\ell^2$-norms of the complex-valued entries of $\mathbf{u}^{(0)}$ and determines the index $s = s^{(0)}$ of the entry of $\mathbf{u}^{(0)}$ with the largest norm.

*2) Iteration Stage:* The second stage handles the iterative processing. At the start of the first cycle ($i = 0$), $\mathbf{u}$ and the results of the first macro-pipeline stage ($|u_i^{(0)}|^2$ and $s^{(0)}$) are loaded into the iteration registers. Since $s^{(0)}$ is already known, the implementation of (4) reduces to identifying $t^{(0)}$, while skipping the entry of $\mathbf{u}^{(0)}$ indexed by $s^{(0)}$. The entries $u_s^{(0)}$ and $u_t^{(0)}$ are then selected and a division unit computes $\beta$ according to (5), whereby the divisor $|u_t^{(0)}|^2$ is immediately available from the iteration register. The result of the division is used to compute $u'_s$ and $|u'_s|^2$ and is also forwarded to the termination unit. When the termination flag is not set, the next cycle starts by loading the iteration register with the updated vector $\mathbf{u}^{(1)}$ and the corresponding norms of its entries. At the same time, the iteration register that contains $s$ can be updated without explicitly searching for the entry of $\mathbf{u}^{(1)}$ with the largest norm, since Brun's algorithm ensures that $s^{(i+1)} = t^{(i)}$. When the termination flag is set, the iteration is reinitialized ($i = 0$) from the first macro-pipeline stage.

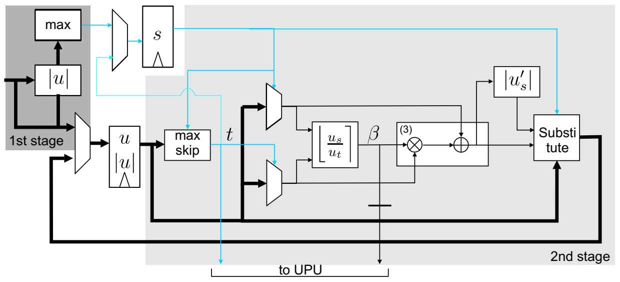


Fig. 2. Detailed block diagram of Brun's algorithm unit (BAU).

## B. RTL Architecture of UPU

The RTL block diagram of the UPU is shown in Fig. 3. Here, the problem is that an isomorphic architecture that implements (7) and (10) requires concurrent instantaneous read-access to two columns of $\mathbf{P}^{(i)}$ and write access to the same memory. However, the memory that contains $\mathbf{P}^{(i)}$ can usually not provide random read-access to more than one column of $\mathbf{P}^{(i)}$ at a time [8] and must also be assumed to have an access latency of one cycle[1].

The resulting memory access bottleneck can be alleviated considerably by exploiting that $\mathbf{p}_s^{(i)} = \mathbf{p}_t^{(i-1)}$ if $i > 0$ so that one can simply keep $\mathbf{p}_t^{(i)}$ in a local cache register and load only $\mathbf{p}_t^{(i+1)}$ from the memory. Unfortunately, the first iteration

[1]Memories with multiple random access ports have a low density and are thus undesirable.

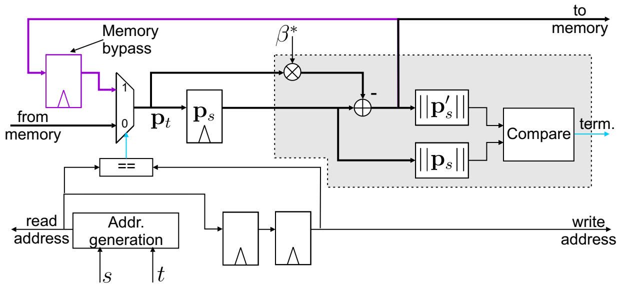


Fig. 3. Detailed block diagram of update **P** unit (UPU).

still requires two access cycles to the memory to load $\mathbf{p}_s^{(0)}$ and $\mathbf{p}_t^{(0)}$. To allow for this additional cycle, the update of $\mathbf{P}^{(i)}$ and the check of the termination condition must be delayed by one cycle, relative to the computation of $\beta^{(i)}$ as illustrated by the timing diagram shown in Fig. 4. In the first cycle of a new

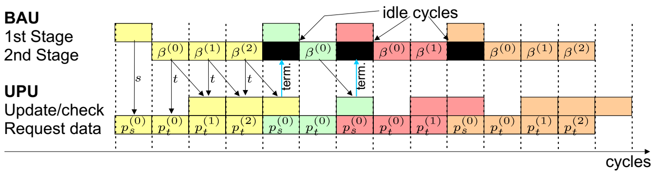


Fig. 4. Timing diagram, illustrating the operation and interaction of the BAU and UPU.

iteration, the first macro-pipeline stage of the BAU decides on the index $s^{(0)}$ which is used immediately by the UPU to issue a request to the memory for the corresponding column of $\mathbf{P}^{(0)}$. The answer to that request arrives one cycle later (Fig. 4 only shows the request to the one-cycle-latency memory[2]). In that second cycle, the index $t^{(0)}$ is available from the BAU and the UPU can immediately request $\mathbf{p}_t^{(0)}$. This vector $\mathbf{p}_t^{(0)}$ arrives in the third cycle together with $\beta^{(0)}$ which has been computed in the second cycle, but was delayed by one cycle by a pipeline register at the output of the BAU (shown in Fig. 2). Note that this register not only serves to align $\beta^{(i)}$ with the corresponding $\mathbf{p}_t^{(i)}$ in the UPU, but also decouples the timing of the two units and prevents a long combinatorial path through both units. The UPU can now update $\mathbf{p}_s^{(0)}$ and check the termination condition. If (10) is *false*, the UPU continues, by storing $\mathbf{p}_t^{(0)}$ in its internal cache register and by requesting $\mathbf{p}_t^{(1)}$ whose index $t$ is available from the BAU. If (10) is *true*, the termination flag is set, causing the BAU to proceed immediately to the next precoding matrix which is already waiting in the first macro-pipeline stage so that the UPU can again start by requesting $\mathbf{p}_s^{(0)}$.

Finally, an additional memory bypass is required in the UPU to deal with the case where $t^{(i)} = s^{(i-1)}$. In that case, the UPU requests data from the memory that is just about to be written back and can not be read at the same time. Hence, the corresponding data must be fed back through a local register as shown in the block diagram in Fig. 3.

## C. Fixed-Point Considerations

We now determine the accuracy requirements, where we consider a compromise between block floating-point and fixed-point arithmetic to work around the problem that the inputs of our circuit have an infinite dynamic range. To this end, the entries of $\mathbf{u}^{(0)}$ and $\mathbf{P}^{(0)}$ are normalized by powers of two in such

[2]Using on-chip memories with one cycle access latency ensures register-like behavior with short combinatorial propagation delays which would otherwise degrade the cycle time.

a way that the magnitudes of their largest real- and imaginary parts are as close as possible to, but smaller than one. The same scaling factors are thereby applied to all entries of the same matrix or vector. As opposed to true block floating-point arithmetic, this normalization is only performed once, before the algorithm starts. In the following, we write $[I, F]$ for a number with $I$ integer and $F$ fractional bits in its real- and imaginary part, where $I$ determines the dynamic range and $F$ the numerical accuracy.

*1) Numerical Requirements for* $\mathbf{u}$*:* Evidently, $\mathbf{u}^{(0)}$ requires zero integer bits and Brun's algorithm ensures that the magnitude of the entries of $\mathbf{u}^{(i)}$ can only decrease [6]. Thus, $I_{\mathbf{u}} = 0$. Furthermore, simulations indicate that $F_{\mathbf{u}} = 7$ maintains close-to floating-point performance up to a symbol error rate (SER) of $10^{-4}$.

*2) Numerical Requirements for* $\beta$*:* Since $\beta$ is an integer by definition, $F_\beta = 0$. Furthermore, the number of integer bits required to capture the result of the division in (5) is given by $I_\beta = 1+I_{\mathbf{u}}+F_{\mathbf{u}}$, provided that the exception $u_t^{(i)} = 0$ is handled separately. Hence, $[I_{\mathbf{u}}, F_{\mathbf{u}}] = [0, 7]$ implies $[I_\beta, F_\beta] = [8, 0]$.

*3) Numerical Requirements for* $\mathbf{P}$*:* The number of integer bits for $\mathbf{P}^{(0)}$ is zero because of the described input normalization. The termination condition in (10) ensures that the length of the column vectors of $\mathbf{P}^{(i)}$ can not increase. However, it may still happen, that a column in $\mathbf{P}^{(i)}$ is altered in such a way that the corresponding vector lies in a single real-valued dimension. In that case, the dynamic range of the corresponding quadrature component is bound by $\pm\sqrt{2K}$ and thus requires $I_{\mathbf{P}} = \lceil(1 + \log_2 K)/2\rceil$ integer bits (i.e., $I_{\mathbf{P}} = 2$ for $K = 4$).

The number of fractional bits must again be found through simulations. Fig. 5 shows the corresponding results, which already include the performance loss due to the fixed-point representation of $\mathbf{u}^{(i)}$. It shows that $F_{\mathbf{P}} \geq 13$ is necessary to operate at an SER of $10^{-4}$ with an SNR penalty of 1.3 dB. We shall set $[I_{\mathbf{P}}, F_{\mathbf{P}}] = [2, 13]$ for the subsequent implementation.

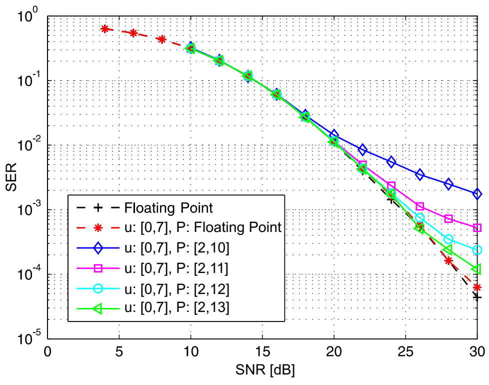


Fig. 5. Impact of quantization on SER performance.

## IV. Implementation Results

For a more detailed analysis of the true silicon complexity the proposed architecture must be implemented in hardware. The corresponding results and parameters used in the reference design for a system with $M_T = K = 4$ are summarized in Tbl. I. A direct comparison of the proposed hardware architecture to other multi-user precoders is not possible, since – to the best of our knowledge – no other VLSI implementations of similar algorithms have been reported in the open literature.

Hence, we compare the complexity overhead associated with adding the proposed LR-aided vector perturbation unit to the matrix-inversion unit described in [8], which has a silicon area of 89K GEs and requires 610ns for a $4 \times 4$ matrix. Obviously,

TABLE I
VLSI implementation results in a 0.25$\mu$m process

| | |
|---|---|
| Area | 1.27 mm$^2$ / 53K GE |
| Crit. path / Clock freq. | 22.4 ns / 44.6 MHz |
| Avrg. cycles/matrix | 3.1 ($\mathcal{E}\{J\} = 2.1$) |
| Avrg. time/matrix | 69.5 ns |
| Fixed-point parameters | $\mathbf{u} : [0, 7], \beta : [8, 0], \mathbf{P} : [2, 13]$ |

the throughput-optimized LR unit requires only a fraction of the time needed for matrix inversion. This indicates that a more area-optimized design for LR may be a more economic choice, since spending more time on LR would only have a minor impact on the total preprocessing latency. However, a high-throughput architecture for LR does become relevant in MIMO-OFDM systems, where the LR unit can be shared among multiple matrix-inversion circuits operating in parallel. Corresponding results are shown in Fig. 6, where a single pipelined LR unit is used together with one, four, and eight matrix processors, required to handle the preprocessing of 1, 64, and 128 matrices within 10$\mu$s.

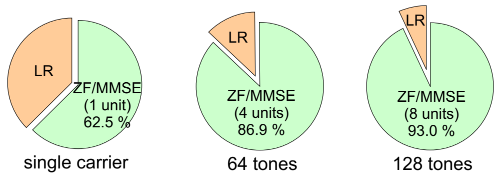


Fig. 6. Share of the total silicon area occupied by the LR unit when 1, 64, and 128 matrices need to be processed within 10$\mu$s.

## V. Conclusions

Reduced complexity algorithms based on integer relations enable the efficient VLSI implementation of lattice-reduction aided vector perturbation for multi-antenna multi-user communication systems. The presented high-throughput VLSI architecture is the first implementation of such a system and provides reference for the true silicon complexity of the required algorithms. Our implementation also illustrates how streamlining of algorithms, motivated by architectural considerations can lead to considerable complexity reduction.

## References

[1] C. B. Peel, B. M. Hochwald, and A. L. Swindlehurst, "A vector-perturbation technique for near-capacity multiantenna multiuser communication - Part I: Channel inversion and regularization," *IEEE Trans. Comm.*, vol. 53, no. 1, pp. 195–202, Jan. 2005.

[2] B. M. Hochwald, C. B. Peel, and A. L. Swindlehurst, "A vector-perturbation technique for near-capacity multiantenna multiuser communication - Part II: Perturbation," *IEEE Trans. Comm.*, vol. 53, no. 3, pp. 537–544, Mar. 2005.

[3] C. Windpassinger, R. F. H. Fischer, T. Vencel, and J. B. Huber, "Precoding in multiantenna and multiuser communications," vol. 3, no. 4, pp. 1305–1316, July 2004.

[4] C. Windpassinger and R. F. H. Fischer, "Low-complexity near-maximum-likelihood detection and precoding for MIMO systems using lattice reduction," in *Proc. IEEE Information Theory Workshop*, Paris, France, March/April 2003, pp. 345–348.

[5] C. Windpassinger, R. F. H. Fischer, and J. B. Huber, "Lattice-reduction-aided broadcast precoding," vol. 52, no. 12, pp. 2057–2060, Dec. 2004.

[6] D. Seethaler and G. Matz, "Efficient vector perturbation in multi-antenna multi-user systems based on approximate integer relations," in *Proc. of the European Signal Proc. Conf. (EUSIPCO)*, Sept. 2006.

[7] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Math. Ann.*, vol. 261, pp. 515–534, 1982.

[8] A. Burg, S. Haene, D. Perels, P. Luethi, N. Felber, and W. Fichtner, "Algorithm and VLSI architecture for linear MMSE detection in MIMO-OFDM systems," in *Proc. IEEE ISCAS*, 2006.