Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 4, April 2013, pg.146 – 154

RESEARCH ARTICLE

Hardware-Optimized Lattice Reduction Algorithm for WiMax/LTE MIMO Detection using VLSI

R.Ragumadhavan[1]

[1]Assistant Professor, Department of Electronics and Communication Engineering, PSNA College of Engineering and Technology, Dindigul, Tamilnadu, India

[1] *raguece85@gmail.com*

Abstract— This paper presents the first ASIC implementation of an LR algorithm which achieves ML diversity. The VLSI implementation is based on a novel hardware-optimized LLL algorithm that has 70% lower complexity than the traditional complex LLL algorithm. This reduction is achieved by replacing all the computationally intensive CLLL operations (multiplication, division and square root) with low-complexity additions and comparisons. The VLSI implementation uses a pipelined architecture that produces an LR-reduced matrix every 40 cycles, which is a 60% reduction compared to current implementations. The proposed design was synthesized in both 130µm and 65nm CMOS resulting in clock speeds of 332MHz and 833MHz, respectively. The 65nm result is a 4X improvement over the fastest LR implementation to date. The proposed LR implementation is able to sustain a throughput of 2Gbps, thus achieving the high data rates required by future standards such as IEEE 802.16m (WiMAX) and LTE-Advanced.

Key Terms: - WiMax; MIMO; Lattice; LTE

I. INTRODUCTION

Recently, lattice-reduction (LR) has been proposed in conjunction with MIMO detection schemes to improve their performance via transforming the system model into an equivalent one with a more orthogonal channel matrix, thereby lowering the likelihood of detection errors due to noise perturbations [1]. The LLL algorithm (due to Lenstra, Lenstra and Lovasz) [2] is the most commonly used LR method and has been shown to achieve ML diversity for low- complexity detectors [3] and significantly improve the performance of more complex detectors such as K-Best [4]. A more efficient, complex-valued extension to LLL (known as CLLL) was developed in [5]. However, the VLSI implementation of CLLL remains problematic due to its computationally intensive operations and its non- deterministic complexity.

Currently, only small number VLSI implementations of LR have been reported in the literature, such as [6], [7] and [8]. Each of these designs was implemented on an FPGA platform. The Clarkson algorithm (CA), presented in [8], is a variant of CLLL that achieves a lower complexity by modifying the CLLL reduction criterion. However, CA, like CLLL, has the drawback of variable complexity and it also relies on computationally intensive operations such as division and multiplication. Another complex LR algorithm known as Seysen's algorithm (SA) was presented in [9], however, we show that SA has a much higher computational complexity than both CA and CLLL. Thus, SA is even more problematic from an implementation point of view. Therefore to achieve an efficient and high-throughput VLSI implementation of LR, there is a need for an algorithm with significantly reduced and deterministic complexity.

In this paper we propose the design and ASIC implementation of a modified CLLL algorithm which achieves a 70% reduction in complexity over existing LR algorithms (including CLLL [5], CA [8], and SA [9]) with effectively the same BER performance. Our algorithm, which we named HOLLL (Hardware-Optimized LLL), eliminates the need for all computationally intensive LLL operations (such as division and multiplication)

and relies only on additions and comparisons. In addition, our algorithm is shown to outperform CLLL and CA when the number of iterations is limited, which is vital from a VLSI implementation perspective. Furthermore, our algorithm integrates the fixed complexity CLLL method presented in [10] to achieve a fixed runtime and allow for a pipelined implementation.

The VLSI implementation of our algorithm represents the first ASIC implementation of any LR algorithm that achieves ML diversity(such as LLL and SA type algorithms). Currently, the only LR ASIC implementation reported is the Brun's algorithm channel pre- coder [11]. However, Brun's algorithm makes several approximations leading to its inferior performance. In addition, since Brun's is a precoding scheme, it requires the channel to be known at the transmitter, which is often difficult to achieve.

Our proposed implementation uses a pipelined multi-stage architecture which produces an LR-reduced matrix every 40 clock cycles which is a 60% reduction over state-of-the-art LR implementations. We synthesized our design in both 0.13μm and 65nm CMOS and achieved speed increases of 1.6X and 4X over previously reported LR implementations. The resulting bit rates of 797Mbps and 2Gbps achieved by our design are complaint with the high data rate requirements of IEEE 802.16m (WiMAX) and LTE-Advanced.

II. LATTICE – REDUCTION – AIDED MIMO DETECTION

Consider a MIMO system with *NT* transmits and *NR* receives antennas. The equivalent baseband model of the channel between the transmitter and the receiver is described by a complex-valued *NR* × *NT* channel matrix *H*. Thus, the complex baseband equivalent model can be expressed as $\mathbf{y} = H\,\mathbf{s} + \mathbf{v}$, where **s** is the *NT* -dimensional complex transmit signal vector, **y** is the *NR* -dimensional received symbol vector, and **v** is the *NR* -dimensional complex Gaussian noise vector. We assume that the channel is quasi-static and updated every four channel uses. Popular MIMO detection schemes such as V-BLAST and K-Best require, as a preprocessing step, the QR-decomposition of the channel matrix as $H = QR$, where *Q* is a unitary *NR* × *NT* matrix and *R* is an upper triangular *NT* × *NT* matrix. Performing a nulling operation by Q^H *yields* the following,

$$\mathbf{z} = QH\,\mathbf{y} = R\mathbf{s} + QH\,\mathbf{v} \quad (1)$$

The objective of MIMO detection is to find an estimated ˆ**s** which minimizes the Eucledian distance × **z** − *R***s** χ ²

For a given matrix *H* , the lattice L(*H*) is the set of all linear combinations generated by the columns of *H* , which are referred to as the basis. Lattice reduction means to transform *H* , via a matrix *T* , into a new basis *H*- = *HT* that is more orthogonal. It can be shown that matrix *H*- will generate the same lattice as *H* , if and only if, the *NT* × *NT* matrix *T* is unimodular [5], i.e., *T* contains only complex integer entries with det(*T*) = ±1. By applying lattice reduction, the system model can be rewritten as

$$\mathbf{y} = H\,\mathbf{s} + \mathbf{v} = HT\,T^{-1}\mathbf{s} + \mathbf{v} = H\text{-}\,\mathbf{x} + \mathbf{v} = Q\text{-}R\text{-}\mathbf{x} + \mathbf{v}, \quad (2)$$

Where *Q*-*R*- is the QR-decomposition of *H*- and $\mathbf{x} = T^{-1}\mathbf{s}$. Note that *H* **s** and *H*- **x** describe the same point in a lattice but the LR-reduced matrix *H*- is usually much better conditioned than the original channel matrix *H* . Therefore, the detector finds an estimated ˆ**x** in the lattice- reduced constellation and estimates the original symbol via ˆ**s** = *T* ˆ**x**.

III. HARDWARE OPTIMIZED LLL ALGORITHM

Currently, the most popular LR method is the LLL algorithm, for which the complex version (CLLL) has been developed in [5]. Although CLLL has been shown to produce high quality reductions, however, the hardware implementation of CLLL is problematic as a result of its high computational complexity and non-deterministic nature. Our proposed algorithm, hardware-optimized LLL (HOLLL), introduces improvements to the CLLL algorithm (while maintaining its performance) which reduce its complexity, minimize its delay and make it more suitable for a VLSI implementation. Our proposed algorithm for efficient lattice reduction (HOLLL) is shown in Table I, and the list of improvements over CLLL are described below.

1) ***Quantization of the*** μ*:* One of the most computationally intensive operations in CLLL is the calculation of the $\mu = \text{I}R\text{-}l,k\,/R\text{-}l,l\,\text{1}$

Coefficient used in the LLL size reduction operations [5], where the I·J operation indicates rounding to the nearest integer. Our simulations show that the dynamic range of μ is quite limited and that over 99.9% of the values of μ lie within {0, ±1, ±2} for CLLL. Therefore instead of performing a division and rounding operation,

We simply quantize μ to these values. Thus requiring only additions and comparisons.

TABLE I
HARDWARE OPTIMIZED LLL ALGORITHM (HOLLL)

$(R, \mathbf{z}, T) = \text{HOLLL}(R, \mathbf{z}, \delta)$

1) $R = R;\ T = I_{N_T \times N_T};\ stop = \text{FALSE};$
2) **while** $stop = \text{FALSE}$
3) $\quad k = 2;\ stop = \text{TRUE};$
4) $\quad$ **while** $k \le N_T$
5) $\quad\quad$ **for** $l = k-1 : -1 : 1$
6) $\quad\quad\quad \mu_q = \text{QUANTIZE}(R_{l,k}/R_{l,l}, [0, \pm 1, \pm 2]);$
7) $\quad\quad\quad R_{1:l,k} = R_{1:l,k} - \mu_q \cdot R_{1:l,l};$
8) $\quad\quad\quad T_{:,k} = T_{:,k} - \mu_q \cdot T_{:,l};$
9) $\quad\quad$ **end**
10) $\quad\quad$ **if** $\delta \cdot |R_{k-1,k-1}| > |R_{k,k}|$
11) $\quad\quad\quad$ Swap $(k-1)$th and kth in R and T;
12) $\quad\quad\quad$ Update R and $\mathbf{z}$ using 2D CORDICs;
13) $\quad\quad\quad stop = \text{FALSE};$
14) $\quad\quad$ **end**
15) $\quad\quad k = k + 1;$
16) $\quad$ **end**
17) **end**

2) Using ***Siegel's Condition****:* The swapping condition for CLLL, known as the Lovasz condition, is as follows,

$$\delta |R_{k-1,k-1}|^2 \le |R_{k-1,k}|^2 + |R_{k,k}|^2 \quad \forall \quad 2 \le k \le N_T. \qquad (3)$$

Direct implementation of the Lovasz condition requires significant computational resources since it involves squaring, addition, multiplication and comparison. Based on the LLL improvement idea in [12], we propose replacing the Lovasz condition with the much simpler Siegel condition (line 10 in Table I). The CA implementation in [8] also used the simpler Siegel condition; however the squared version was used. In our algorithm we have taken the square root of both sides to eliminate the extra unnecessary calculations

3) Replacing with - *Q*-
Compute each of R and Q. However, the MIMO detection schemes that require QR decomposition such as V-BLAST and K-BEST do not use the Q matrix directly, rather $\mathbf{z} = Q^H \mathbf{y}$ must be computed first before the detection can take place. Therefore, a more efficient method would be to find $\mathbf{z}$ directly instead of Q. This is achieved in our algorithm by modifying the basis update equations (line 12 in Table I) in order to calculate $\mathbf{z}$ directly and thus avoiding the extra multiplication needed after finding Q.

4) Replacing Multiplications with 2D CORDICs*:* In the CLLL algorithm, if in a given iteration k the Lovasz condition holds true, then the kth and $(k-1)$th columns are swapped in R and T. Following this a Givens rotation operation is performed on each of R and Q.

These operations were performed using matrix multiplication in [6], [7] and [8]. Our approach, however, relies entirely on 2D CORDIC vectoring and rotation operations which replaces multiplications with additions, and thus greatly simplifies its hardware implementation

TABLE II
ALGORITHMIC COMPLEXITY OF LR ALGORITHMS FOR 4 × 4 MIMO

LR Algorithm	Iters	BUs	Add	Mult	Div	Sqrt	FLOPS[a]
CLLL (δ = 3/4)	5.6	1.5	451.6	546.8	27.0	1.5	1227.8
SA	5.5	5.5	1680.5	2400.3	284.0	0	6353.0
CA (δ = 1/2)	5.0	1.2	405.4	493.8	18.9	1.2	1059.6
This Work (HOLLL)	**3.8**	**0.3**	**380.6**	**0**	**0**	**0**	**380.6**

[a]FLOPS = Add + Mult + (8 x Div) + (8 x Sqrt)

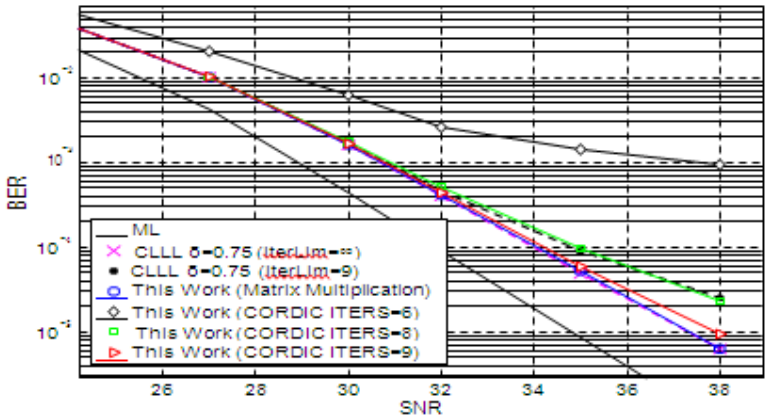

Fig. 1. Comparison of HOLLL performance using various CORDIC iteration values vs. matrix multiplication (using K-Best Real K=10 for 4×4 64-QAM).

*5) **Fixed Complexity**:* Unlike CLLL where the sequence of *k* values depends on the input *R*, we adopt the same approach shown in [10], where a pre-determined sequence of *k* values is traversed. For *NT* =4, this sequence is *k*=(2, 3, 4, 2, 3, 4, ...). As a result of this fixed order, the LR iterations can be implemented independently and in a pipelined high-throughput fashion (compared to the previous high-delay iterative implementations in which a single architecture was used for all LR iterations).

Table II compares the complexity of our algorithm to previously known LR algorithms: CLLL [5], SA [9], and CA [8]. Clearly our algorithm has the lowest complexity, where the FLOP count results show that our algorithm reduces the complexity of CLLL by nearly 70%. Furthermore, our algorithm has the lowest average number of iterations and basis updates indicating that it is a more computationally efficient LR algorithm.

As a result of adopting the fixed complexity method in [10], it is most efficient to implement the LR iterations as multiples of 3 such that complete passes through the R- matrix are performed. Based on this we simulated HOLLL with the iterations limited to each of (3, 6, 9, 12), and a limit of 9 was selected since it achieved the best trade-off between performance and complexity. After selecting this limit, the algorithm was simulated for various CORDIC iteration values. The result of this simulation, shown in Fig. 1, indicates that a CORDIC iteration count of 9 has a minimal performance loss compared to the ideal case of matrix multiplication.

IV. VLSI ARCHITECTURE

As a result of the large reduction in complexity of CLLL, the VLSI architecture we designed to implement HOLLL is significantly more efficient than previously reported implementations. Our implementation consists of 9 pipelined LR stages that perform the 9 LR iterations. The intermediate *R*, **z** and *T* values are stored in register banks between each stage. Control is implemented using a distributed approach, where each LR stage has its own dedicated controller for local operations. Following a bottom-up approach, we describe the architectures used to implement the Size Reduction and Basis Update steps (respectively lines 5-9 and 11-12 in Table I), followed by the architecture used for a single HOLLL iteration, and finally the pipelined integration of all the LR stages.

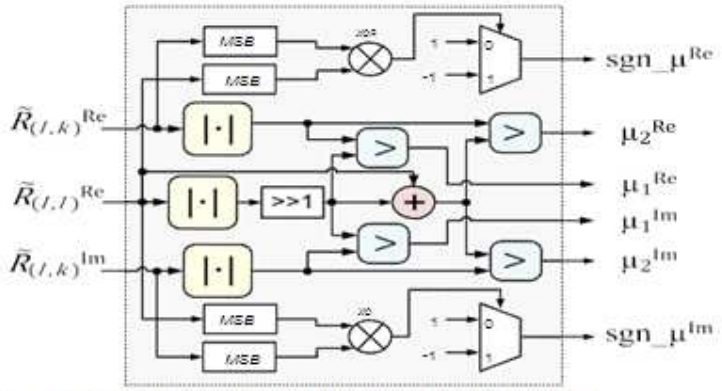

Fig. 2. Architecture for the quantized μ_q calculation block.

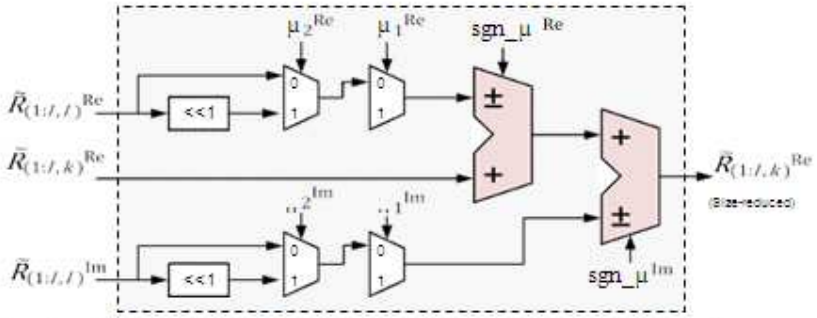

Fig. 3. Architecture for the Complex Size Reduction (CSR) block.

A. Implementation of the Size Reduction Step

The architecture for computing the quantized μq coefficient is shown in Fig. 2. The elementary operations, as a result of the quantization of μ, lead to a calculation delay of only one clock cycle. Note, that instead of explicit computation of μq , only the comparison results are needed. These results are stored in local registers and used to control the addition/subtraction in the Complex Size Reduction (CSR) block shown in Fig. 3. Here as well, the elementary operations in the CSR block allow execution in only one clock cycle.

B. Implementation of the Siegel Condition and the Basis Update Step

The δ factor in the Siegel Condition (line 10 in Table I) plays a key role in the performance of the LR algorithms. A larger δ typically produces higher quality LR-reduced matrices leading to a better BER performance, but a smaller δ results in a faster execution and a better iteration-limited performance. With this in mind, in our design we implemented the Siegel Condition to allow for dynamic control of δ as shown in fig. 4. The selectable δ values were set.

This flexibility allows the LR algorithm to adapt to varying input conditions (e.g. SNR and correlation) as well as enable dynamic control of δ even within a single LR reduction. This is important since, following suggestions by [12] and [13], our simulations show that the lattice reduction quality for limited iterations can be further increased by applying a smaller δ in the earlier LR iterations (to maximize speed) followed by a larger δ in the latter LR iterations (to maximize quality).

The Basis Update Step (lines 11-12 in Table I) consists of column swapping followed by Givens Rotations which we implement using 2D CORDIC vectoring and rotation operations. To increase the throughput, we unrolled the 2D CORDIC into 9 pipelined stages with a delay of 9 clock cycles. Each stage can be configured to be used in either vectoring or rotation mode thus achieving maximum utilization. Furthermore, instead of explicitly calculating the CORDIC rotation angles, we use control signals which encode the rotation angle as shown in [14]. This implicit angle calculation results in eliminating the angle data path and a hardware savings of approximately 30%.

The first and last CORDIC stages each include coarse rotation options for vectoring and rotation, respectively. The coarse rotation capability is necessary to extend the range of the CORDIC rotation angles from ($-\pi/2$, $\pi/2$)

to (-π, π) and thus allow the zeroing of vectors [*x*, *y*] with a negative *x* value. The last CORDIC stage also includes a scaling unit to apply the 2D CORDIC gain compensation factor of (0.6073). The scaling unit is implemented using shift and add operations to apply an approximate factor of (0.625).

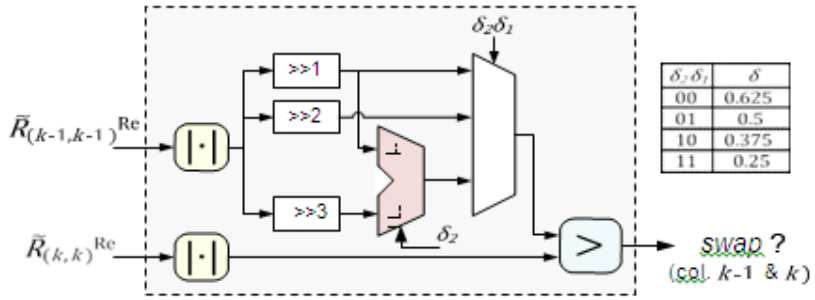

Fig. 4. Architecture for Siegel Condition with Dynamic δ Control.

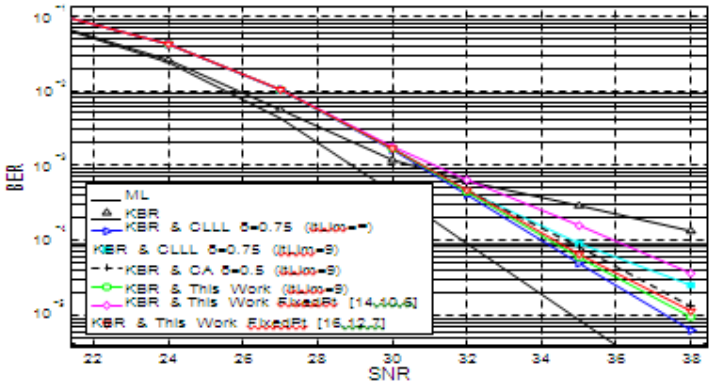

Fig. 5. Fixed Point Simulation of Hardware-Optimized LLL (HOLLL) for 4 × 4 64-QAM K-Best Real (KBR, K=10).

C. *Fixed Point Analysis and Simulation*

To efficiently select fixed point parameters, we performed extensive simulations of the floating point version of HOLLL and analyzed the dynamic range each matrix entry in *R*-, - and *T* throughout all the LR iterations. In the tradeoff between performance and implementation area, the following parameters were shown to achieve the best results: 4 integer bits and 12 fraction bits for both *R*- and - matrices, and 7 integer bits for the *T* matrix. These parameters are collectively denoted as [16,12,7]. Fig. 5 shows the BER performance of our algorithm (in both floating and fixed point versions) compared to other common LR algorithms in a practical iteration-limited scenario. The detection algorithm used is the near-ML K-Best with *K*=10 and 4 × 4 64-QAM MIMO. For high SNR values and an iteration limit of 9, our HOLLL algorithm outperforms CLLL and CA by 1.6dB and 0.5dB, respectively. In addition, the performance loss due to the fixed point implementation of HOLLL is only 0.25dB.

D. *Architecture for a Single HOLLL Iteration*

The block diagram for an individual HOLLL iteration is shown in Fig. 6 along with the corresponding clock cycles required for the three cases: *k*=2, *k*=3 and *k*=4. The scheduling shown in the figure was optimized to minimize hardware resources while maximizing throughput. Matrix data is read from the input register bank, followed by the μq and Siegel calculation blocks which execute in parallel and in one clock cycle. Two CSR blocks with wordlengths 16 and 7 are used for *R*- and *T* , respectively. To reduce hardware, each of these blocks is reused to complete all the size reduction operations. If the Siegel condition passes, then the vectoring operation starts in cycle 3 when the new size-reduced *R*-*k*,*k* value is available. Through an efficient scheduling sequence, all CORDIC operations are complete by clock cycles 44, 42 and 40 for *k*=2, *k*=3 and *k*=4, respectively. The final values are stored in the output register bank to be used by the next HOLLL iteration. Note that each HOLLL iteration has a dedicated controller that manages dataflow and scheduling.

Previous efforts to reduce the complexity of CLLL were based on performing size reduction against *R*-*k* 1,: only (i.e. only the first execution of the loop in lines 5-9 in Table I), this is known as Effective LLL

(ELLL) [7]. In our implementation, we have showed how *full* size reduction can be performed with minimal hardware (using only two CSR blocks) without affecting the latency of the overall HOLLL iteration. This is because the latency of pipelined CORDIC operations (which is itself at maximum throughput) is longer than that of size reduction in all cases.

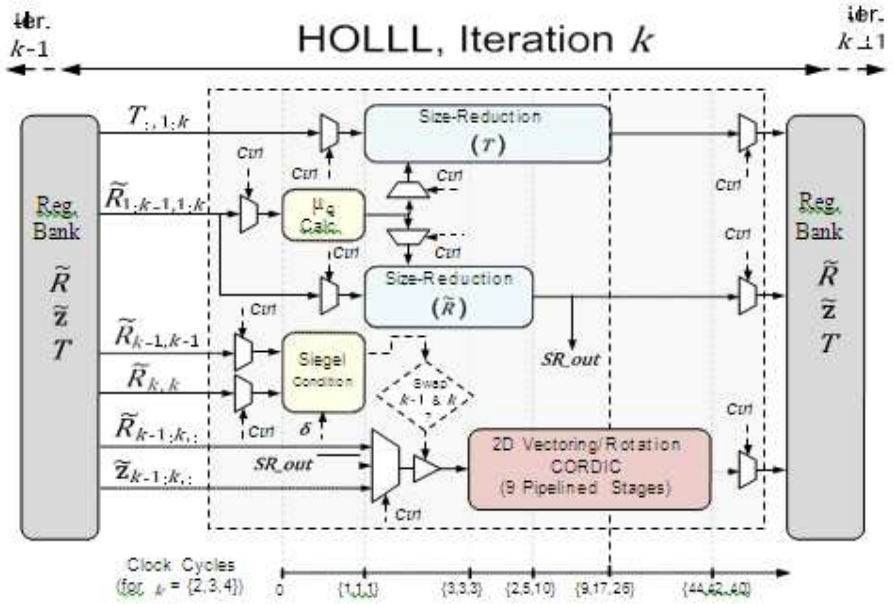

Fig. 6. Block Diagram for one HOLLL iteration including clock cycles.

E. Proposed Pipelined Architecture

The Complete Architecture consists of an input controller followed by 9 HOLLL pipelined stages and ending with an output controller. The LR reduction process proceeds as follows. The input controller accepts individual matrix entries for R and $\mathbf{z}$, where the real and imaginary values of each complex matrix entry are entered in parallel. All data entry is completed within 40 cycles. The matrix data then propagates through the HOLLL stages via the register banks in between each stage. Finally, every 40 cycles, the output controller produces the LR-reduced 4×4 R- matrix, the 4×4 T matrix, and four 4X1 $\mathbf{z}$ vectors (note, four detected vectors are needed due to the quasistatic channel being updated every four channel uses).

V. ASIC IMPLEMENTATION

The VLSI architecture presented above was implemented in both 0.13μm and 65nm CMOS technology. This is the first ASIC imple- mentation of any LR algorithm which achieves ML diversity (such as CLLL, CA and SA). Therefore, in comparing our post-synthesis implementations results we are restricted to comparisons with recent FPGA implementations of LR. This comparison is shown in Table III. Both our 0.13μm and 65nm implementations produce an LR- reduced matrix every 40 cycles which is more than 60% lower than the next best implementation. Moreover, our implementation has the significant advantage of completing all computations in a fixed number of cycles, whereas in other FPGA implementations the results represent an average, since their exact number of required cycles depends on the correlation of the input matrix R. Clearly, this variability is highly undesirable for a practical implementation of LR as a preprocessing step in a larger MIMO detection framework.

Our 65nm implementation operates at 833 MHz which is more than 4X faster than the fastest LR implementation to date. Since, emerging 4G wireless standards such IEEE 802.16m and LTE- Advanced envision data rates in the range of 1-2 Gbps, it is instructive to see how close various LR implementations are to supporting these very high rates. For this study, we assume a detection method in 64- QAM uses the LR block and is unrestricted in its ability to match the LR block in terms of its Cycles/Matrix throughput. Based on this, the resulting bit rates are calculated and included in Table III. Clearly, our proposed implementation of LR in 65nm achieves the required bit rates, and therefore it will not be in the critical path of MIMO detection systems implementing future 4G wireless standards.

TABLE III
COMPARISON OF VLSI IMPLEMENTATIONS OF LR FOR 4 × 4 MIMO

	[6]	[7]	[8]	This Work	
LR Algorithm	CLLL	ELLL	CA	HOLLL	
Platform	Virtex-5	Virtex-5	Virtex-II Pro	0.13μm	65nm
Core Area [mm^2]	-	-	-	1.04	0.40
Core Area [kGE]	78.7	64.7	-	144.8[a]	193.0
Slices	1,712	1,369	7,349	-	-
Clock Freq. [MHz]	163	205	100	332	833
Cycles/Matrix	130^{avg}	102^{avg}	420^{avg}	40^{fixed}	40^{fixed}
Time/Matrix [ns]	797.5	497.6	4200	120.5	48.0
Latency [μs]	-	-	-	0.96	0.38
Power [mW]	-	-	-	70.7	155.1
Voltage [V]	-	-	-	1.2	1.2
Achievable Bit Rate in 64-QAM [Mbps]	120	193	23	797	2000

One Gate Equivalent (GE) in 0.13μm and 65nm CMOS technology corresponds to the area of their respective 2-input drive-two NAND gates

VI. CONCLUSIONS

An ASIC implementation of hardware-optimized lattice reduction has been proposed for improving the performance of MIMO detection. Our implementation, based on a novel highly optimized version of the popular CLLL LR algorithm, uses pipelined multi- stage architecture to produce an LR-reduced matrix every 40 cycles. The proposed design was synthesized in both 130μm and 65nm, with the latter showing a 4X improvement over the fastest LR implementation to date. As a result, our design is able to sustain a throughput of 2Gbps which is in line with the requirements of emerging standards IEEE 802.16m (WiMAX) and LTE-Advanced. The chip implementation, fabrication and testing of our design is a subject of ongoing work.

REFERENCES

[1] H. Yao and G. Wornell, "Lattice-reduction-aided detectors for MIMO communication systems," IEEE Globecom, pp. 17–21, 2002.
[2] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, "Factoring polynomials with rational coefficients," Math Annual, vol. 261, pp. 515–534, 1982.
[3] D. Wubben, R. Bohnke, V. Kuhn, and K. Kammeyer, "Near-maximum- likelihood detection of MIMO systems using MMSE-based lattice- reduction," IEEE Intern. Conf. on Communic., vol. 2, pp. 798–802, 2004.
[4] M. Shabany and P. G. Gulak, "The application of lattice-reduction to the K-Best algorithm for near-optimal MIMO detection," IEEE Intern. Symp. on Circuits and Systems, pp. 316–319, May 2008.
[5] Y. H. Gan and W. H. Mow, "Complex lattice reduction algorithms for low-complexity MIMO detection," IEEE Global Telecommunications Conf., vol. 5, pp. 2953–2957, Nov. 2005.
[6] B. Gestner, W. Zhang, X. Ma, and D. V. Anderson, "VLSI implemen- tation of a lattice reduction algorithm for low-complexity equalization," IEEE Intern. Conf. on Circuits and Systems for Commun., pp. 643–647, May 2008.
[7] "VLSI implementation of an effective lattice reduction algo- rithm with fixed-point considerations," IEEE Intern. Conf. on Acoustics, Speech and Signal Processing, pp. 577–580, April 2009.
[8] L. G. Barbero, D. L. Milliner, T. Ratnarajah, J. R. Barry, and C. Cowan, "Rapid prototyping of Clarkson's lattice reduction for MIMO detection," IEEE Intern. Conf. on Communications, pp. 1–5, June 2009.
[9] D. Seethaler, G. Matz, and F. Hlawatsch, "Low-complexity MIMO data detection using Seysen's lattice reduction algorithm," IEEE Intern. Conf. on Acoustics, Speech and Signal Proc., vol. 3, pp. 53–56, 2007.
[10] H. Vetter, V. Ponnampalam, M. Sandell, and P. A. Hoeher, "Fixed complexity LLL algorithm," IEEE Transactions on Signal Processing, vol. 57, no. 4, pp. 1634–1637, April 2009.
[11] A. Burg, D. Seethaler, and G. Matz, "VLSI implementation of a lattice-reduction algorithm for multi-antenna

broadcast precoding," IEEE Intern. Symp. on Circuits and Systems, vol. 1, pp. 673–676, May 2007.
[12] H. Cohen, "A course in computational algebraic number theory, 3 ed," Graduate Texts in Mathematics, vol. 138, p. 90, 1996.
[13] B. A. LaMacchia, "Basis reduction algorithms and subset sum prob- lems," Master Thesis, MIT, May 1991.
[14] S. F. Hsiao and J. M. Delosme, "Householder CORDIC algorithms," IEEE Trans. on Computers, vol. 44, no. 8, pp. 990–1001, Aug 1995.

