Implementation of LLL Based Preprocessor for MIMO Detection

Midhun M. Pillai
M.Tech, VLSI Design, Dept. of ECE
SRM University, Kattangulathur, Tamil Nadu, India.

Archana T.M
M.Tech VLSI Design, Dept. of ECE,
SRM University, Kattangulathur, Tamil Nadu, India.

Mrs. K Ferenk Koni Jiavana
Asst. Professor,
SRM University, Kattangulathur, Tamil Nadu, India.

Abstract
In this paper, the objective, is the complete implementation of Lattice Reduction (LR) based MIMO Preprocessor. There are many prominent LR algorithm, among which, this paper makes use of the Lenstra, Lenstra, Lovasz (LLL) algorithm. The idea is for the full implementation, including the entire RTL synthesis and the implementation in EDA tool. The usual LLL algorithm implementation requires high hardware realization difficulties due to the complex operations involved. As to avoid the problem, an optimization in the hardware context is made so as to make the realization less complex. First of all, the digital design of the entire IP blocks involved in the algorithm is realized using Verilog HDL coding for 32 bit floating point data format. All the HDL realization is followed by accurate simulation and analysis. Then, the entire blocks are to be synthesized using the Cadence RC Compiler and the digital implementation is done using the Encounter Digital Implementation in 180 nm technology. This preprocessing technique is followed by its application to K-Best method of MIMO detection.

Introduction
MIMO (Multiple Input Multiple Output) is a recently emerged mode of communication which uses multiple antennas at both transmitting and receiving ends. It has more advantages than other communication modes of which a few are the high data rate, improved bit error rate (BER), high spectral density, increased range of communication etc. In MIMO antenna terminology, the data can be transmitted mainly in two ways, i.e., same data can be transmitted through multiple antennas which results in low bit errors or, multiple data can be transmitted through different antennas, which leads to better data rate.

As of having advantages of high spectral efficiency, increased range, and robustness MIMO technology has recently emerged as the technology of choice in next-generation wireless standards, such as long term evolution (LTE) and IEEE 802.16 (WiMAX). However, in order for the complete realization of this mode of technology, crucial design challenges need to be overcome. A critical challenge of this technology is the design of high-throughput near-maximum-likelihood (ML) detectors capable of supporting 4 G data rates (approximately1 Gb/s) in spite of the large number of antennas required in MIMO technology.

In order to achieve all those peculiarities of this efficient mode of communication, there are certain requirements that has to be carried out, of which, the main one is the preprocessing technique which improves the data reliability and information content. The prominent mode of preprocessing is the Lattice Reduction aided preprocessing technique which transforms the system model into a more or near orthogonal matrix thus, improving the BER performance of the MIMO detectors.

There are many LR algorithms, namely: 1) Lenstra, Lenstra, Lovasz algorithm (LLL algorithm); 2)Seysen’s algorithm; 3) Brun’s algorithm. In this paper, the proposed hardware optimized version of LLL algorithm, called HOLLL is being implemented which reduces the complexity of existing LR algorithm to a far extend, despite achieving the same BER performance. This optimized version eliminates complex computational processes such as division and multiplication and emphasizes mainly only on addition and comparison operations.

In this proposed design, a pipelined multistage architecture is used, having a fixed time complexity producing LR reduced matrix every 40 clock cycles. The implementation is done using 180 nm library of Cadence RC Compiler and Encounter Digital Implementation.

Objective of MIMO Detection
The Complex baseband equivalent model of MIMO system can be expressed as
\[ y = Hs + v. \]  
where \( s \) is the NT-dimensional complex signal vector, \( y \) is the \( N_R \)-dimensional received symbol vector, and \( v \) is the \( N_R \) dimensional complex Gaussian vector, \( H \) is the channel matrix.

Thus, the main objective of MIMO detection is to recover \( s \) from \( y \) on the knowledge of \( H \). That is, by transforming the
entire system model into a near orthogonal channel matrix, it lowers the likelihood of detection errors.

Need For Preprocessing
The channel matrix $H$ is a matrix of basis vectors in a lattice. So to reduce the correlation between the vectors and to make them less dependent with each other, we are reducing the correlation between the basis vectors linearly. This makes the data more informative and less redundant. There are many ways of preprocessing methods that has been used for different kinds of communication systems. In MIMO detection systems, Lattice Reduction has been widely used as pre-processing method due to its implementation benefits. In this project, implementation of hardware optimized LLL algorithm that achieves a large reduction in complexity over existing LR algorithms is proposed. Moreover, the proposed design out performs any other design till date while considering the number of iterations as deterministic, which is vital from a VLSI implementation context.

LR BASED MIMO DETECTION
Let us acknowledge a MIMO system with $NT$ transmit and $NR$ receive antennas. The complex-valued $NR \times NT$ channel matrix $H$ which describes equivalent baseband model of the channel between the transmitter and the receiver sides of the channel. Prominent MIMO detection schemes such as V-BLAST and K-best algorithm based detectors as a preprocessing step, requires the QR-decomposition of the channel matrix as $H$ into $H = QR$, where $Q$ is a unitary $NR \times NT$ matrix and $R$ is an upper triangular $NT \times NT$ matrix. Performing a nulling operation by $Q^H$ yields $z = Q^Hy = Rs + Q^Hv$. Thus, the objective of MIMO detection hence can be considered to determine an estimated $\hat{s}$ that minimizes the Euclidian space $\| z-Rs \|^2$.

The term basis is referred to as the set of all possible linear combinations generated by the columns of $H$. LR is meant to transform $H$ through a matrix $T$, into a new basis $\hat{H} = HT$. In short, the idea behind the orthogonalizing the basis vectors is to reduce the correlation of the channel matrix and to make the decision regions closer to that of ideal regions of ML detectors. The matrix $\hat{H}$ will generate the same lattice as that of $H$, if and only if the $T$ matrix ($NT \times NT$) is unimodular i.e. $T$ contains only complex integer entries with $\text{det}(T) = \pm 1$. The system model can be rewritten by applying the LR as shown below:

$$Y = Hs + v = HT^{-1}s + v = \hat{H}x + v = \hat{Q} \cdot \hat{R}x + v.$$  

LR ALGORITHMS
As the objective is to find a basis with near/short orthogonal vectors and since, the problem of finding such orthogonal basis is NP-hard, several near optimal algorithms have been proposed in the mathematical literatures, among which a most widely used are:

1) LLL algorithm;
2) SEY algorithm;
3) Brun’s algorithm.

Selection of the Proper LR Algorithm
In order to determine which LR algorithm should be chosen for further optimizations and hardware implementation, there is a need for a comprehensive complexity and performance analysis. To achieve this, three criteria, i.e., “LR iteration and basis update (BU), ” “number of operations,” and “algorithm variations and scaling,” are defined and considered in the following.

LR Iteration and BU:
LLL and SEY are fundamentally different algorithms. Therefore, for a fair comparison, there is a need to clearly define the LR iteration and LR BU operation for each of the LLL and SEY, independent of their individual underlying calculation method. In this regard, consider the LR process during which a series of partially reduced channel matrices $Hi$ are produced satisfying $Hi = HTi$ (4).

Number of Operations:
It is also necessary to compare LLL and SEY in terms of the number of distinct real-valued operations, namely, addition, multiplication, division, and square root, through calculating the number of real floating point operations (FLOPS).

Algorithm Variations and Scaling:
To offer a balanced view of LLL and SEY, this paper analyzes both the real and complex version of LLL with various values of $\delta$ ($\delta \in \{3/4, 1\}$) as well as both the Greedy and Lazy versions of SEY.

LLL Algorithm
The LLL algorithm is essentially the generalization of Gaussian Reduction (GR) technique to arbitrarily higher dimensions. The lengths of the basis vectors are hence, reduced by subtracting each vector of its integer components with each of the last smaller vectors (i.e., those vectors that have already been reduced or processed) which is carried out in size reduction operation. By this operation successively, it contributes to increasing the orthogonality of the basis vectors by reducing their length in a pair wise manner. Once all possible size reduction operations have been completed, the next step in LLL is to compare the length of the current basis vector to the previous one and swap each other if they are not following ascending order. This reordering is done with the aim of allowing further size reductions to take place.

CLLL Algorithm:
To begin with, the Complex LLL(CLlll) algorithm gets the QR-decomposition of the channel matrix $H$ as an input, and iteratively lowers the correlation between the basis vectors of the channel matrix to produce a near-orthogonal basis for the matrix $H = \hat{Q} \cdot \hat{R}$ that satisfies the following conditions:

$$|R(R_{k, l})|, |I(R_{k, l})| \leq 1/2|R_{l, l}| \forall 1 \leq k \leq N_T$$  

(5).

$$\delta |R_{k-1, k-1}| \leq |R_{k-1, k-1}|^2 + |R_{k, k}|^2 \forall 2 \leq k \leq N_B.$$  

(6).
where $\delta$ is the quality factor that lies in the range of $[1/4, 1]$, and $R$ and $Q$ are the lattice-reduced $Q$ and $R$ matrices. These two conditions are known as the size reduction and the Lovász basis swapping condition, respectively. Moving from $k = 2$ to $N_T$, the algorithm performs basis reduction operations to size-reduce each $k$th column of $R$ against its previous $1:k−1$ columns (lines 4–8). The $[•]$ operation in line (5) indicates rounding to the nearest integer. After the size reduction, the Lovász condition is checked for the $k^{th}$ and $(k−1)^{th}$ columns of $R$; if it passes, then the two columns are swapped followed by application of Givens Rotations carried out to maintain the upper-triangular nature of $R$ (lines 9–14), otherwise the algorithm proceeds to the next column pair. The $\delta \in [1/4, 1]$ factor controls the tradeoff between the speed of the algorithm and the quality of the reduced basis, e.g., $\delta = 1$ gives the highest quality but the slowest execution time, while $\delta = 1/4$ gives the fastest execution time with the lowest quality moderate choice of $\delta = 3/4$ achieves a good balance between speed and quality. The outputs of CLLL are the updated $Q$, $R$, and $T$ matrices. These two conditions are known as the size reduction and the Lovász basis swapping condition, respectively. Moving from $k = 2$ to $N_T$, the algorithm performs basis reduction operations to size-reduce each $k$th column of $R$ against its previous $1:k−1$ columns (lines 4–8). The $[•]$ operation in line (5) indicates rounding to the nearest integer. After the size reduction, the Lovász condition is checked for the $k^{th}$ and $(k−1)^{th}$ columns of $R$; if it passes, then the two columns are swapped followed by application of Givens Rotations carried out to maintain the upper-triangular nature of $R$ (lines 9–14), otherwise the algorithm proceeds to the next column pair. The $\delta \in [1/4, 1]$ factor controls the tradeoff between the speed of the algorithm and the quality of the reduced basis, e.g., $\delta = 1$ gives the highest quality but the slowest execution time, while $\delta = 1/4$ gives the fastest execution time with the lowest quality moderate choice of $\delta = 3/4$ achieves a good balance between speed and quality. The outputs of CLLL are the updated $Q$, $R$, and $T$ matrices.

\[
(\tilde{R}, \tilde{Z}, T) = \text{HOLLL}(R, Z, \delta)
\]

1) $R = R$; $T = I_{N_T \times N_T}$; $stop = \text{FALSE}$;
2) while $stop = \text{FALSE}$
3) $k = 2$; $stop = \text{TRUE}$;
4) while $k \leq N_T$
5) for $l = k − 1 : 1$
6) $\mu_q \rightarrow \text{QUANTIZE}(\tilde{R}_l/k/\tilde{R}; 0, \pm 1, \pm 2))$
7) $\tilde{R}(1 : l, k) = \tilde{R}(1 : l, k) - \mu_q \cdot \tilde{R}(1 : l, l)$
8) $T(:, k) = T(:, k) - \mu_q \cdot T(:, l)$;
9) end
10) if $\delta \cdot |\tilde{R}_{k-1,k-1}| > |\tilde{R}_{k,k}|$
11) Swap $(k-1)^{th}$ and $k^{th}$ columns in $\tilde{R}$ and $T$;
12) Update $\tilde{R}$ and $\tilde{Z}$ using 2-D CORDICs;
13) $stop = \text{FALSE}$;
14) end
15) $k = k + 1$
16) end
17) end

Algorithm 1: CLLL Alg.

Algorithm 2: HOLLL Alg.

**HOLLL Algorithm**

Based on the design complexity and performance analysis, the CLLL algorithm has been modified into the proposed novel design, here after referred to as the Hardware Optimized LLL Algorithm (HOLLL). The HOLLL flow diagram has been represented in fig. 1. The main functional blocks are listed below:

Algorithm 2: HOLLL Alg.
MU Calculation:
The calculation of the quantized complex $\mu q$ value consists of separately calculating the real and imaginary components ($\mu r$ and $\mu i$) as well as their respective signs (Algorithm 3). However, because of the way in which the $\mu q$ factor is needed in the subsequent size reduction block and the fact that the quantized values are limited to $\{0, \pm1, \pm2\}$, it is possible to avoid explicit calculation of the $\mu r$ and $\mu i$ components. This is done by decomposing $\mu r$ and $\mu i$ into the intermediate results of the conditional statements and using these binary values as multiplexor controls in the size reduction operation. The intermediate results of the conditional statements are denoted using,

$$
\mu r = \mu 1_{Re} + \mu 2_{Re} \quad \text{and} \quad \mu i = \mu 1_{Im} + \mu 2_{Im}.
$$

Complex Size Reduction (CSR) Block:
The size reduction operations are achieved via a CSR block using the control outputs from the $\mu$ quantization block. There are three CSR blocks used in each iteration:
1) One for calculating the real value of $R (1: l, k)$;
2) One for the imaginary component calculation; and
3) One for $T$ matrix size reductions.

The size-reduced values of $R$ are time-critical, in the sense that they are required immediately by CORDIC rotation operations, thus any delay incurred in their calculations results in more processing latency.

Siegel Calculation Block:
The $\delta$ factor in the swapping condition (using either Siegel or Lovász conditions) plays a key role in the performance of the LR algorithms. Therefore, a flexible architecture is proposed to implement the Siegel condition which allows the dynamic control of $\delta$. The value of $\delta$ can be controlled via primary inputs to the LR core, where the allowable $\delta$ values were selected from the set $\{1/8, 3/8, 1/2, 5/8\}$. This flexibility allows the LR algorithm to adapt to varying input conditions (e.g., SNR and correlation) as well as a dynamic control of $\delta$ within a single LR reduction. This dynamic control can be utilized effectively such that smaller $\delta$ values can be used in the earlier LR iterations (to maximize speed) followed by larger $\delta$ values in the latter LR iterations (to maximize the quality).

Basis Update Block:
The BU step (lines 11 and 12 in Algorithm 2) consists of column swapping followed by Givens rotations, which are all implemented using 2-D CORDIC vectoring and rotation operations in this paper. Since a large number of 2-D CORDIC rotations must be performed after each of the three required vectoring operations (see Fig. 5), an unrolled 2-D CORDIC with nine pipeline stages is proposed to maximize the throughput. Fig. 13 shows the proposed architecture where each CORDIC stage can be configured to be used in either the vectoring or the rotation mode, thus achieving the maximum utilization. Furthermore, instead of explicitly calculating the CORDIC rotation angles, direction signals were used to encode the rotation angle, resulting in a 30% hardware savings.
180 NM ASIC IMPLEMENTATION.

The simulation of each block has been carried out and the results has been shown below. The simulation is carried out using Mentor Graphics Model Sim-Altera 10.1b Edition and the results are recorded accordingly. The tabular illustrations of the combined HOLLL alg. core along with inputs and corresponding outputs are given accordingly. The proposed size reduction and BU blocks are then combined to build a functional block for one single HOLLL iteration (Fig.6&7). The scheduling shown was optimized to minimize hardware resources while maximizing throughput. The data values for single iteration i.e. the combined iteration of every block has to be done by invoking values for \( F, Z \) and \( T \) from the input register bank and the preprocessed or reduced correlated outputs are stored in the output register bank.

The proposed VLSI Design has been synthesized using Cadence RC Compiler tool at a clock time period of 10 ns. The library used for this purpose is slow normal for 180 nm technology. The implemented design has been shown in fig. 8.

DETECTION.

Among several MIMO detection algorithms linear detection algorithm such as Minimum Mean Square Error (MMSE) or Successive Interference Cancellation (SIC) detectors can greatly reduce computational complexity but at the same time they have reduced performance. ML detectors provide the optimal solution at high computational complexity. To solve the tradeoff between complexity and performance loss, near optimal receiver detection algorithms which provide near ML output at reduced complexity than ML detectors were proposed which include depth first and breadth first algorithm. Execution of nodes in breadth first algorithm is shown in fig 9.

Now we consider level \( l \) of the tree and suppose that the set of \( K \)-best candidates in level \( l+1 \) is known. Each node in level \( l+1 \) has \( \sqrt{M} \) possible children, so there are \( K\sqrt{M} \) possible children.
in level I. One of the key elements of our proposed design is to find the children of each node on-demand and in the order of increasing PED.

First child or Next child calculation
In on-demand scheme, the first and next child are required to be determined. The proposed scheme is pictorially depicted in Fig. 4 for level I where \( M = 4 \) and \( K = 3 \). The input to the algorithm is the best selected nodes of level that are the present parents with corresponding PEDs of 0.1, 0.4, and 0.6. Each parent can be further expanded to four offsprings resulting in 12 children whose PEDs are shown in Fig. 3. The child in with the lowest PED is found by sorting the PED values. This child should be added to \( l \). To find the next best child in \( K \)-best list its corresponding PED is removed and replaced by its next best sibling. This procedure repeats \( K = 3 \) times to find all the \( K \)-best candidates. The final children in the \( K \)-best list are with PEDs 0.2, 0.5, and 0.7, respectively. Note that using the proposed scheme, only 5 children of 12 possible children are visited in Fig. 3. This savings becomes increasingly significant for large values.

VLSI Implementation of K-Best Detector

Description
The proposed architecture with all intermediate parameters for a 4x4, 64-QAM MIMO system with \( K = 10 \) and is shown in Fig. 4. There are 8 levels in the tree. For each of the nodes in \( k_{j_1} \), the first child is found and its PED is updated using the FC-Block in Level II. Then the FC with the lowest PED should be determined, for which all the FCs are to be sorted. This can be done using the Sorter block in Fig. 11.

Sorter block output will be the sorted values of FC’s in level 7, which are all loaded simultaneously to the next stage which can be denoted as \( k_7 \). PE I block’s input is all the first child’s of each level and it generates the \( K \)-best list out of this FC’s of that level one-by-one. The node with the lowest PED will be one of the \( K \)-best candidates in level 7. This \( K \)-best value is passed to the PE II block. By removing this FC, its next sibling is calculated, which is done by the NC-Block in the feedback loop of the PE I block, and substitutes the FC. Then the PED of this next child needs to be compared with the other FCs, already present in this stage. The next \( K \)-best candidate has the lowest PED among this new group. This process is repeated up to 10 times and until all the \( K \)-best values of the second level of the tree are generated.

The PE II receives the \( K \)-best candidates of level 7, one after the other, and generates the FC of each received \( K \)-best candidate step-by-step and sorts them as they arrive. It finally transfers them to its following PE I block. This process repeats for all the levels down to the first level.

VLSI Architecture

Level 1
The architecture for Level I is shown in Fig. 5. The input to Level I is \( r_{10} \) and \( \bar{z}_6 \). This architecture employs a 5x5 bit multiplier, a few adders and the absolute value block. The absolute value block represents the \( l^2 \)-norm. Output of Level I is the PEDs of all nodes in 8th level of tree.

Level 2
The input to Level II is the Partial Euclidean distance values of the 8th level and its output is the PED values of the first children in the 7th level of the tree. To find the first child in the 7th level \( \bar{z}_7 \) is applied to the input of the Mapper/Limiter block whose output is the first child. The architecture of the Level II block is shown in Fig. 6.
The Sorter block’s input is the set of 8 PED values of the 7th level FCs and it will produce sorted list of these PED values. Fig. 7 shows the architecture of sorter. The eight inputs are denoted by, and the outputs are stored in eight registers labeled by letter “N”. The Ctrl signal is used to load the data.

Processing Element I(PE I)
The architecture of the PE I block is shown in Fig. 8. It take first child of each level as input and generates K-best list of that level one by one. It selects the node with lowest PED value.

Processing Element II(PE II)
Architecture for PE II block is shown in Fig. 10. It consist of first child block. The output of PE I is the serial list of K-best candidates of the current level, generated one-by-one at the output. This serial list is given as input to PE II block.

When each of the K-best candidates are generated, it is sent to the PE II block. This block calculate the first children of the next level and sort them according to their PED values. First,
the FC of the K-best candidate of the preceding stage and its updated PED value are calculated by the FC-Block, and then by the use of a sequential sorter, the calculated PED values are sorted. In the proposed architecture for PE II, the sorted PEDs are stored in the registers, as shown in Fig 15. The functionality of the sorter is such that the larger values are shifted to the right while the smaller values are shifted to the left. First child block inside PE II block is shown in Fig 16.

Figure 17: Architecture of First child block

K-Best algorithm can be used as an efficient detection method for MIMO detection. It can achieve power efficiency of at least 30% than that of other existing technologies. In this architecture we have used on demand expansion and distributed sorting scheme in a pipelined fashion so that this will reduce the time and increases efficiency.

References


Simulation Results

Simulation Result of K-Best Algorithm

Figure 18: single iteration of k-best detector.