VLSI Implementation of a Staggered Sphere Decoder Design for MIMO Detection

*Pankaj Bhagawat,*Sasidharan Ekambavanan, **Sabyasach Das, *Gwan Choi, *Sunil P Khatri
* Department of ECE, Texas A&M University , ** Synplicity, Inc.

***Abstract*— MIMO is a key technology for future high speed wireless communication applications. Present MIMO systems are constrained by the lack of efficient detection algorithms/hardware architectures. In this paper we present a novel algorithm/architecture which takes a hybrid approach toward MIMO detection in the sense that it simultaneously searches along the depth and breadth of the tree. We have implemented our architecture using an ASIC design flow in a 100nm process technology. Our implementation results are compared with fully serial, fully parallel, and sequential architectures. We refer to the proposed algorithm as Staggered Sphere Decoding with Ordering (SSDO). SSDO is an excellent match for high throughput MIMO-OFDM systems like 802.11n etc. ASIC estimates (using a 100nm technology) indicate that SSDO achieves a guaranteed minimum throughput of 287 Mbps, with an area utilization of 2070 μm^2. Our approach achieves significantly higher throughput than the serial and sequential approaches, and half the throughput of the parallel approach (using a fifth of its area). The throughput per unit power and throughput per unit area are significantly improved over the parallel, serial and sequential approaches.**

***Index Terms*— MIMO systems, K-best algorithm, Implementation/Time complexity, Hybrid architecture.**

I. INTRODUCTION AND PREVIOUS WORK

Recently, multiple input-multiple output (MIMO) wireless systems have received a great deal of attention in the wireless communication community, due to their ability to deliver very high data rates without any overhead in terms of bandwidth or transmitter power as compared to single input single output (SISO) wireless system. MIMO systems, however, pose a major problem as far as implementation complexity is concerned. One of the major challenges in implementing a MIMO based system is the high complexity of the detection algorithm at the receiver. Among the earliest algorithms used for signal detection for un-coded MIMO was VBLAST [1]. Although computationally efficient, VBLAST suffers from a substantial degradation of Bit Error Rate (BER) performance.

A family of algorithms under active consideration relies on a tree based search. Two of the most notable approaches in this family are the Sphere Decoding (SD) algorithm [2], which is a Depth First Search (DFS) based algorithm, and the K-best algorithm which is a Breadth First Search (BFS) based algorithm [6]. Authors in [5] provide two ASIC implementations of the SD algorithm. Implementation of K-best algorithm has been reported in [7]. The SD implementation in [5] takes a sequential approach, whereas in [6-7] the authors take a highly parallel approach to MIMO decoding. Both approaches have their merits and demerits. For instance, being serial in nature the implementation of the SD algorithm requires smaller silicon area than the K-best algorithm, but has a highly variable throughput. Hence, it is difficult to integrate it into the overall communication system. On the other hand, the K-best algorithm provides a fixed (lower) throughput while utilizing a much higher silicon area due to extensive sorting operations [7]. In general, the tree based algorithms offer a good trade-off between performance and complexity [8].

Recently a very low complexity algorithm has been reported in [9]. This algorithm, called the Fixed-Throughput Sphere Decoder (FSD), relies on a special ordering (FSD ordering) of the columns of the channel matrix, which results in a significantly reduced search space. The resulting implementation has a very simple data path. The authors in [9] have proposed a fully parallel architecture (BFS based) to implement the algorithm, which provides very high and fixed throughput. However a fully parallel approach consumes large area and incurs many redundant computations. A sequential architecture [10], in conjunction with radius reduction technique reduces redundant computations and consumes less silicon real estate, but has lower throughput than the parallel scheme. In contrast, our solution requires significantly lower power and area than the parallel architecture. It also achieves much higher throughput than the fully serial and the sequential architecture by using a novel architecture that performs depth and breadth search simultaneously.

The key contributions of this work are: A novel MIMO decoding algorithm and its associated hardware architecture. This algorithm leads to faster tree pruning, and hence a higher throughput. Also, the runtime variability of the algorithm is much less than the sequential and fully serial architectures. We compare the fully parallel, fully serial, and sequential [10] architectures with the new proposed approach, based on an ASIC implementation of each. The results indicate that our approach achieves the best performance in terms of throughput per unit power and throughput per area.

This paper is organized as follows. In Section II, we briefly describe the channel model and review some of the existing detection algorithms. In this section, we also describe the enumeration scheme that we employ in our implementations. In Section III, we describe the proposed architecture along with the fully parallel, fully serial, and sequential architectures. In Section IV, we provide the detailed implementation of the proposed architecture, while briefly describing the other architectures. In Section V, the implementation results of our approach is compared with that of the fully parallel, sequential, and fully serial architectures. We conclude in section VI.

II. MIMO DETECTION

A. Channel model:

The baseband system model for a MIMO system with M_T transmit and M_R receive antennas can be expressed as follows [3].

$$\mathbf{y}=\mathbf{H}\mathbf{s}+\mathbf{n} \quad (1)$$

where $\mathbf{s}$ is $M_T\times1$ transmitted vector or vector symbol, $\mathbf{n}$ is $M_R\times1$ zero mean complex Gaussian noise vector, and $\mathbf{H}$ is $M_R x M_T$ – dimensional complex matrix. The $(i,j)^{th}$ element, h_{ij}, of the matrix $\mathbf{H}$ denotes the complex channel gain from the j^{th} transmit antenna to the i^{th} receive antenna. In this paper we will assume $M_T= M_R=4$, unless specified otherwise.

B. Sphere decoding.

The objective of the MIMO detection algorithm is to compute an estimate $\hat{\mathbf{s}}$ of $\mathbf{s}$ such that:

$$\hat{\mathbf{s}} = \arg \min_{\mathbf{s} \in \Omega^{M_T}} \left\| \mathbf{y} - \mathbf{H}\mathbf{s} \right\|^2 \quad (2)$$

where Ω is set of complex entries from the QAM constellation and Q is the cardinality of the set Ω. A straightforward approach to solving (2) is an exhaustive search over all possible candidate vector symbols $\mathbf{s} \in \Omega^{M_T}$. However, since the number of possible solutions grows exponentially with M_T, the implementation of an exhaustive search becomes impractical as M_T increases. For example, in case of a 4x4 MIMO system with 16-QAM modulation an exhaustive search would require the evaluation of 65536 candidate vector symbols.

One way to circumvent the exhaustive search is to evaluate only a small subset of all the possible vectors. Notice that **H** can be triangularized using QR factorization: **H=QR**. Hence, the cost function given by (2) can be rewritten as equation (3) [3],

$$\hat{\mathbf{s}} = \left\| \mathbf{y} - \mathbf{H}\mathbf{s} \right\|^2 = \left\| \hat{\mathbf{y}} - \mathbf{R}\mathbf{s} \right\|^2 \quad \text{, and} \quad \hat{\mathbf{y}} = \mathbf{Q}^H \mathbf{y} \quad (3)$$

Where, **R** is an upper triangular matrix, and $\mathbf{Q}^H$ is the Hermitian of a unitary matrix **Q** [3]. Vector $\hat{\mathbf{y}}$ as defined by (3) is the unconstrained zero forcing solution. The fact that **R** is upper-triangular ensures that each term in the summation depends only on the current level decision, as well as the history of the path to reach that level in the tree.

Equation (3) can now be rewritten to form summation across each transmit antenna.

$$d_i(\mathbf{s}^{(i)}) = d_{i+1}(\mathbf{s}^{(i+1)}) + \left| e_i(\mathbf{s}^{(i)}) \right|^2 \quad (4)$$

$$\left| e_i(\mathbf{s}^{(i)}) \right|^2 = \left| c_{i+1}(\mathbf{s}^{(i+1)}) - R_{ii}.s_i \right|^2 \quad (5)$$

$$c_{i+1}(\mathbf{s}^{(i+1)}) = \hat{y}_i - \sum_{j=i+1}^{M_T} R_{ij}.s_j \quad (6)$$

The quantity $d_i(\mathbf{s}^{(i)})$ is called the cumulative metric. The quantity $|e_i(s^{(i)})|^2$ is called the incremental metric. The vector $\mathbf{s}^{(i)} = [s_i, s_{i+1}, ..., s_{M_T}]^T$ in Equations (4)-(6) denotes a partial vector symbol candidate. Term $d_i(\mathbf{s}^{(i)})$ for i>1will be called Partial Distance (PD), and Distance (D) for i=1. Because the PD's depend only on $s^{(i+1)}$,they can be associated with corresponding nodes in a Q-*ary* tree with M_T levels. Alternatively, the computation of the terms $d_i(\mathbf{s}^{(i)})$ can be interpreted as a traversal of the tree from the root to the leaf corresponding to **s**. Note that i=1 corresponds to leaf nodes. The estimate $\hat{\mathbf{s}}$ can now be obtained by searching the leaf with smallest D and returning the path from the top level (i=M_T) to that leaf as $\hat{\mathbf{s}}$. The PD's and D's in (4) are equivalently referred to as the node's metric in the sequel. For a 4x4 MIMO system with 16-QAM modulation the total number of nodes in the tree would be $16+16^2+16^3+16^4=69904$, a very large search space. To reduce the search space aggressive pruning of the tree is needed. Sphere decoding does this by pruning a branch of the tree whenever the PD of a node on the branch exceeds a certain limit (radius r). Appropriately setting the radius r restricts the number of nodes visited in the search, since $d_i(\mathbf{s}^{(i)})$ in (4) is monotonically increasing. Hence any node that has a PD more than r need not be considered any further. This is called the Sphere Criterion or SC [5]. This effectively prunes the whole sub-tree rooted at that node, thus greatly reducing the search space. Fixing the value of r is in general difficult. If r is too small then we may have to restart the decoder as no point $\hat{\mathbf{s}}$ may be found. Similarly if r is too large the decoder may visit a very large number of nodes leading to excessive latency. To counter this problem a scheme known as *radius reduction* [3] is used, wherein the initial value of r is set to infinity, and is updated whenever the D of a newly evaluated leaf is less than the current value of the radius. This scheme works best if the nodes are processed in an ascending order of their PD's (i.e. the most promising candidates are processed first).This scheme is called Schnorr-Euchner (SE) ordering [5]. We will omit further details for the sake of brevity. For more details on the sphere decoder refer to [5] and references therein.

III. ARCHITECTURE DEVELOPMENT

In this section we begin by reviewing the FSD algorithm, and then we discuss the sequential, fully serial, and parallel architectures to implement the same. Finally, we propose a new architecture called Staggered Sphere Decoding with FSD ordering (SSDO).

A. Review of the FSD Algorithm.

The FSD algorithm is essentially based on reordering the columns of the **H** matrix such that the tree search is simplified [8]. Basically, all the nodes at the top level (all children of root node) are evaluated, but from then onwards only the best child (child with least PD/D) is considered. For a 4x4 system with 16 QAM, this entails computing PD's/D's for a *maximum* of 16x4=64 nodes, significantly reducing the search space. Figure 3.1 shows the tree traversal for FSD (thick lines) for the case of 3x3 with 4 QAM modulation. Each circle represents a computation of the equations from (4)-(6).

B. Parallel, Sequential and Fully Serial Architectures

In this section we describe different architectures with that can be used to implement a sphere decoder using FSD ordering. In Fig. 3.2 we show the serial, sequential [10] and fully parallel [9] architectural dataflow. The numbers next to nodes indicates the clock cycle in which the node was evaluated. The sequential architecture consists of two units. The first unit is capable of computing the top level (i=4) PD's in ascending order and perform radius check. The second unit concurrently computes PD's/D's for i=1,2,3 (second unit is idle for the first clock cycle) and performs radius check. PD's/D's are computed using a Metric Computation Unit (MCU). The sequential architecture makes use of the radius reduction scheme. The nodes labeled 4,7,…can potentially update the radius value. Note that in sequential scheme the radius update *may* happen every 3 cycles (after first 4 cycles). The major drawback of this scheme is that it provides highly variable throughput. The latency for detecting one vector symbol is dependant on the channel conditions and noise level.

In parallel architecture, *all* the 16 nodes at *every* level are evaluated in parallel using 16 units. Each unit is capable of computing PD's/D's, but *does not* perform a radius check. After reaching level i=1, a compare/select logic picks the path with least D, and declares it as $\hat{\mathbf{s}}$. This scheme has a fixed throughput because it takes 4 cycles to compute the estimate $\hat{\mathbf{s}}$ irrespective of channel conditions and noise level. The parallel architecture does not do any radius update, however, it incurs redundant computations. In fully serial architecture exactly one node is evaluated in one clock cycle. The difference between sequential and serial approach lies in the fact that in sequential approach there is one dedicated unit to operate at

i=4, whereas, in fully serial same unit is shared to compute top level PD's (i=4) and the nodes at levels i=1,2,3. This difference can be clearly seen in Fig. 3.2 and 3.3. A unit in this scheme too computes PD's/EDs and performs radius check. Note that the labeling in case of serial starts with 5, this is because the first 4 cycles are spent in computing the metric frontier.

C. Metric Computation Unit (MCU)

At each level of the tree we need to compute the $d_i(\mathbf{s}^{(i)})$ metrics using equations (4)-(6). For i=4,3,2,1, the equations (4)-(6) can be expanded as follows:

$$d_4\left(\mathbf{s}^{(4)}\right)=\left|y_4-R_{44}.s_4\right|^2 \quad (7)$$

$$d_3\left(\mathbf{s}^{(3)}\right)=d_4\left(\mathbf{s}^{(4)}\right)+\left|y_3-R_{34}.s_4-R_{33}.s_3\right|^2 \quad (8)$$

$$d_2\left(\mathbf{s}^{(2)}\right)=d_3\left(\mathbf{s}^{(3)}\right)+\left|y_2-R_{24}.s_4-R_{23}.s_3-R_{22}.s_2\right|^2 \quad (9)$$

$$d_1\left(\mathbf{s}^{(1)}\right)=d_2\left(\mathbf{s}^{(2)}\right)+\left|y_1-R_{12}.s_2-R_{13}.s_3-R_{14}.s_4-R_{11}.s_1\right|^2 \quad (10)$$

Notice that the number of computations required in (7)-(10) is dependant on the level i of the tree. The largest number of computations are required at i=1 i.e. at the leaf nodes (equation (10)). Fig. 3.3 shows an MCU which can handle computations at i=1. Note that for, the parallel, fully serial, and sequential architectures the same MCU must be capable of computing metrics at different values of i. This can be achieved by reconfiguring the MCU to perform the computations of equations (9),(8) and (7). For the sequential architecture, however, the MCU has to be *reconfigured* between levels i=1,2,3 only, since at the i=1 level we have a dedicated unit to compute PD's.

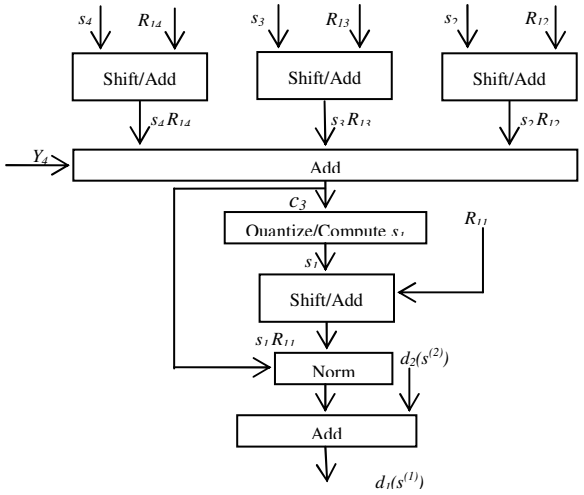

Fig. 3.3 Architecture for MCU_1

It is clear from equations (7)-(10), that the reconfiguration can be carried out simply by input reassignment and by setting some of the operands to zero. In the sequel we will refer to MCU *configured* for a level i as MCU_i. Note that physically the MCU is the same; it is merely reconfigured to operate at different levels i. Also note that when the MCU is configured to evaluate nodes at level i>1 many adders remain idle during the computation.

The terms in equations (7)-(10) involving multiplications of the form $R_{ij}.s_i$ can be computed using shift and add operation. This is possible because the real and imaginary parts of s_i can only take values {-3,-1, 1, 3}.

D. Discussion

Both sequential and parallel architectures have their advantages and disadvantages. Parallel architecture provides a very high and fixed throughput, since only 4 clock cycles are needed to compute the ML solution. The price to pay for this is a high silicon area since 16 MCU's have to be implemented. However, the parallel architecture does not take advantage of radius reduction technique, which leads to redundant computations and hence more power consumption.

The sequential architecture on the other hand consumes much less area (since only 2 MCU's are utilized), but has lower and variable throughput. The variable throughput is a major problem as far as practical systems are concerned. To alleviate this problem, the authors of [8] have proposed a scheme called Block Early Termination or BET. Using this scheme, a runtime constraint is established for a block of vector symbols. The runtime is decided by dynamically allocating a limited number of cycles to processing a vector symbol. A scheduling algorithm is used to distribute the available run time over the vector symbols in the block. The strategy allocates a maximum runtime equivalent of $clk_{max}(n)$ to the n^{th} vector symbol in a block according to

$$clk_{\max}(n)=N_{\max}.clk_{ave}-\sum_{i=1}^{n-1}clk(i)-(N_{\max}-n)M_T \quad (10)$$

where $clk(i)$ denotes the actual number of clock cycles used up for the i^{th} vector symbol. The idea behind Equation (10) is that a vector symbol is allowed to use up all of the remaining run time within the block up to a safety margin of $(N_{max}-n)M_T$ visited nodes, which allows to find at least the zero-forcing decision feedback solution for the remaining vector symbols [8]

E. Our Approach: Staggered Decoding Architecture.

In this section we will develop the motivation for designing an architecture based on the discussion in earlier sections.

Runtime variability: The fully serial scheme suffers from high variability, this is because potentially a new value of radius is obtained every 4 clock cycles (see Fig. 3.2). This problem also exists in the sequential architecture, albeit it is slightly less pronounced. This is because in the case of a sequential architecture a new leaf is potentially reached every 3 clock cycles. The fully parallel architecture does not suffer from this problem as it does not follow the radius reduction scheme.

Redundant computations: The fully parallel architecture often computes metrics for unnecessary nodes as well, since the tree is not pruned by the radius checks during the computation.

Hardware usage: Since the MCU's, for the serial, sequential and parallel architectures need to be reconfigured for each level; some of the computation units are idle during many cycles.

The above mentioned issues can be alleviated to a great extent by using the architectural dataflow shown in Fig. 3.4. The proposed algorithm/architecture is based on the FSD ordering and has a schedule that is staggered in time. One copy of MCU is deployed at each level. The staggered computation is explained as follows. In the first cycle a node is evaluated at level 4 (labeled 1 in the Fig.3.4). In second cycle a node at level 4 and level 3 are evaluated (labeled 2 in the Fig. 3.4). From the 4^{th} cycle onwards one node (subject to SC) from each level is evaluated (resulting in a potential radius update at every cycle). The proposed approach is a hybrid solution between the fully serial and the fully parallel architecture, in that it initiates a new breadth and depth search every cycle (Fig. 3.4). It does not deliver a fixed throughput in contrast to the fully parallel implementation of the FSD algorithm, but

has variability which is much less than serial and sequential architecture due to per cycle radius update. Notice that in the staggered architecture the MCU's are "custom" made for each level, and do not have to be *reconfigured*, leading to improved hardware usage. The staggered decoding scheme follows the radius reduction technique with pruning, thus reducing redundant computations in comparison to the parallel architecture.

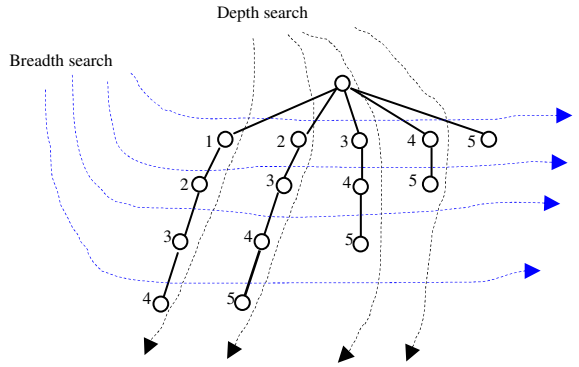

Fig.3.4 Staggered Decoding Flow

The initial value of radius is set to ∞, whenever a leaf node with a metric less than current value of radius is encountered the radius value is updated with the metric of that leaf node. Also, after the first four cycles, potentially a new leaf node is reached which leads to faster radius reduction. This results in reduced variability of runtime.

F. Enumeration Strategies

To process the children nodes of a node in ascending order of their PD's, the distances of c_{i+1} from the *scaled* (by R_{ii}) QAM points need to be ordered (see equation (4)-(5)). A straightforward way to do this is to directly compute all distances and sort them. However, this approach is not efficient as only one node from the top level is needed at a time.

A more efficient way to achieve this is through direct enumeration of QAM symbols based on the location of c_{i+1} in the constellation. Many enumeration techniques exist in literature. In [3] the author proposes a method based on dividing the constellation into concentric circles [3,5]. Recently a new technique has been proposed in [11,10]. The basic idea is to divide the constellation (Fig.3.6) in a number of columns (four in case of 16-QAM). The order of points in these columns is already known given the location of c_{i+1}. For the example shown in the Fig. 3.5, the order is shown on the left hand side of the constellation diagram. This order is identical for every column. Also notice that only one element in SF and MF will change at a time. For instance, in Fig. 3.5 the SF at time t=1 is {2, 6, 10, 14}. Since the minimum distance symbol is '10', the SF is updated at t=2 to {2, 6, **9**, 14}.The set of best QAM symbols (or simply symbol) from every column will be called a Symbol Frontier or SF. The SF can at most have 4 elements in it. Using symbols in the SF their PD's can be computed using Equations (4)-(6). The set of PD's of the elements in the SF will be called Metric Frontier or MF. The best symbol (closest to c_{i+1}) out of all 16 symbols can now be found by picking the symbol associated with the minimum element in the MF. Note that the SF and hence MF keep changing with time. Note that only the location of the imaginary part of c_{i+1} is required to compute the order of symbols in each column. Symmetry of the QAM constellation can be exploited to reduce the number of decisions we have to make in order to locate c_{i+1} in the constellation. For instance if c_{i+1} was in the third quadrant of the constellation, we can ignore the signs of its real and imaginary part to get c'_{i+1}. The SF can be computed for c'_{i+1}. To get back the SF associated with the original c_{i+1} we simply change the signs of real and imaginary parts of the SF elements. This way we only need to compare the imaginary part of c_{i+1} with R_{ii}, and $2.R_{ii}$. Also note that for i=4, $c_{i+1}= \hat{y}_4$.

IV. Implementation Details

In this section we describe the detailed architecture of the staggered decoder. The sequential and parallel architectures use similar building blocks and will be described only briefly. Before we begin describing the details of the architecture we would briefly talk about the use of simplified norms.

Simplified Norm Computation: The MCU as shown in Fig. 3.3 has a block labeled "Norm". The Euclidean norm or ℓ^2 norm involves a squaring operation which requires multipliers. Multipliers are in general expensive in terms of hardware cost. In [5] it has been shown that the use of simplified norms leads to significant reduction in hardware cost with some BER degradation. The Euclidean norm in (3) can be replaced with ℓ^1 or ℓ^∞ norm to simplify the implementation of (4)-(6).

The ℓ^1 norm approximation for (4) is given by:

$$d_i(\mathbf{s}^{(i)}) = d_{i+1}(\mathbf{s}^{(i+1)}) + \left|\mathrm{Re}\{e_i(\mathbf{s}^{(i)})\}\right| + \left|\mathrm{Im}\{e_i(\mathbf{s}^{(i)})\}\right| \qquad (11)$$

and the ℓ^∞ norm approximation is given by:

$$d_i(\mathbf{s}^{(i)}) = \max[d_{i+1}(\mathbf{s}^{(i+1)}), \left|\mathrm{Re}\{e_i(\mathbf{s}^{(i)})\}\right|, \left|\mathrm{Im}\{e_i(\mathbf{s}^{(i)})\}\right|] \qquad (12)$$

where Re{} and Im{} denotes real and imaginary parts.

The use of ℓ^1 norm is of particular interest as it causes the BER to degrade only by about 0.4dB-0.5dB [5]. The use of ℓ^∞ norm on the other hand causes about 1.4 dB loss [5]. Both these norms have almost the same hardware complexity [5]; however the algorithm takes much longer to converge when the ℓ^1 norm is used [5,8]. In the rest of this paper, we will use the ℓ^1 norm due to its better BER.

A. High Level Architecture

Fig. 4.1 shows high level architecture for the staggered decoding scheme. It has four units MCU_4 through MCU_1 operating in a pipeline. The task of MCU_4 is to generate QAM symbols (s_4) and their metric in ascending order (using the enumeration scheme in Fig.3.5) and to perform the radius check. The outputs of MCU_4 are: SC_4 (a '1' indicates an SC violation at i=4), s_4 , the QAM symbol currently under consideration, and $d_i(\mathbf{s}^{(4)})$, the PD associated with s_4.

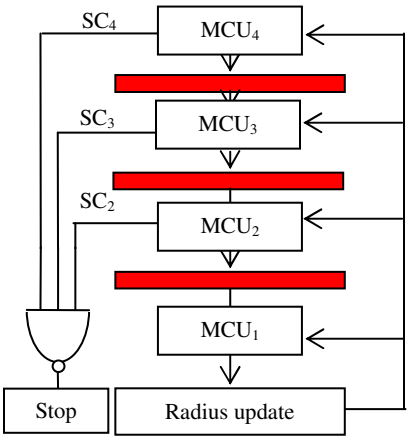

Fig.4.1 High Level Architecture for Staggered Sphere Decoding

MCU_3, MCU_2, and MCU_1 differ from MCU_4 in that they do not *enumerate* the QAM symbols. Instead they find the best child of the input node, compute the associated PD (or D), and perform the radius check (Note that MCU_1 does not perform a radius check, since it is implicitly done in the "Radius update" block).

B. Architecture of MCU_4

Fig. 4.2 shows the detailed structure of MCU_4. As mentioned earlier, MCU_4 outputs symbols and associated PD's in their ascending order. To this end it follows the enumeration strategy outlined in Section III.F. Recall that knowing the location of c_{i+1} in the constellation tells us the order of the QAM points in each column. Once the location of c_{i+1} is known, the SF is computed using a bank of PC_k blocks (k=1,2,3,4). A bank of Partial Distance Units (PDUs) operates on the SF to compute MF. A PDU block (Fig. 4.4) computes the distance of c_{i+1} from a scaled QAM symbol. The compare select logic picks the best symbol based on the minimum PD. Fig. 4.3 shows details of the PC_k block. It consists of a combinational logic block (F), which outputs the imaginary parts of the QAM symbols in signed magnitude format. It has four outputs of three bits each. These outputs correspond to the unique pattern of imaginary parts of the four symbols in a column depending on location of c_{i+1} in the constellation. These imaginary parts are finally concatenated with the real part (notice that in each column the symbols have identical real parts). The counter labelled $cntr_k$, is a 3 bit counter whose two LSBs select the best available symbol in column k. Note that we need a three bit counter in order to identify the *exhaustion* of a column. Since the column has only 4 symbols in it, a value of 4 (counter starts from value '000') indicates that all the symbols in that column have been exhausted. Hence, whenever counter value is 4 a signal f_{ck} is generated. This signal alerts the PDU to saturate its output. This essentially excludes symbols from that column from further consideration.

The outputs of a PC_k block (a frontier symbol from k^{th} column FS_k and a control signal fc_k) are fed to a PDU. The PDU computes the associated PD of the FS_k. The compare/select logic block along with the demultiplexer identifies which frontier symbol needs to be replaced. Fig. 4.4 shows details of a PDU. The PDU at the top level differs with the one shown in Fig. 4.4 in that there is no $d_{i+1}(\mathbf{s}^{(i+1)})$ input, since $d_5(\mathbf{s}^{(5)})=0$. Also, for PDUs at levels $i<4$ there is no f_{ck} input, and no multiplexer M2. This difference is indicated in Figure 4.4 by means of the dotted lines.

C. Architecture of MCU_1, MCU_2 and MCU_3

The top level architecture of MCU_3 through MCU_1 is shown in Figure 4.5. MCU_3 through MCU_1 differ from MCU_4 in that they don't have to explicitly enumerate QAM symbols in sorted order; all that is needed to be done by MCU_3-MCU_1 is to find the best child. Finding best child essentially means "quantizing" c_{i+1} (Fig. 3.5) to the nearest QAM symbol. Fig. 4.6 shows the details of the "Quantize" block.

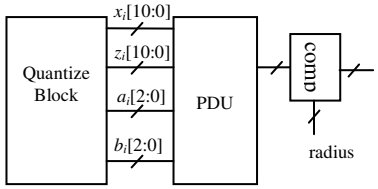

Fig. 4.5 Architecture of MCU_1, MCU_1, and MCU_3

Let $y_i = \alpha_i + j.\beta_i$, $s_i = a_i + j.b_i$, $R_{ij} = u_{ij} + j.v_{ij}$, and $c_{i+1} = x_i + j.z_i$. Substituting these expressions in (6) and after some algebraic manipulations equation (6) can be written as:

$$x_i = \alpha_i + \sum_{k=1}^{M_T - i} \left[\left(-u_{i,i+k}\right)\left(a_{i+k}\right) + \left(v_{i,i+k}\right)\left(b_{i+k}\right) \right] \quad \text{for } i=1,2,3 \qquad (9)$$

$$z_i = \beta_i + \sum_{k=1}^{M_T - i} \left[\left(-u_{i,i+k}\right)\left(b_{i+k}\right) + \left(-v_{i,i+k}\right)\left(a_{i+k}\right) \right] \quad \text{for } i=1,2,3 \qquad (10)$$

Note that for $i=4$ $x_i=\alpha_i$, and $y_i=\beta_i$.

Since c_{i+1} is also a complex number we need to quantize it along both the real and imaginary dimensions. For this purpose we use the X-block (for the real part) and the Z-block (for the imaginary part). These blocks compute x_i and z_i using (9) and (10). Note that here we have exploited the symmetry of the QAM constellation. First we find the magnitude of real and imaginary parts of c_{i+1}, so that the new $c'_{i+1}=|x_i|+j.|y_i|$ is now in the first quadrant. Using the sign bits x^s and z^s along with the location of the c'_{i+1} we can find the location of c_{i+1}. This function is carried out by F' block which is a very simple combinational block which directly outputs the QAM symbol (indicated by a_i and b_i in the Fig.4.6) in signed magnitude format. Fig.4.7 shows more details of the X block. The X-block carries out computations according to (9). Product terms in (9) are implemented using shift and add function (shown by the dotted boxes). The multi-operand additions are performed by an arithmetic Sum of Products (SOP) block [19].

The Z-block is structurally similar to the X-block with inputs reassigned, as can be seen from equation (10).

Sequential architecture implementation: Recall that this architecture has two MCU's. The first MCU computes the PD's of children of the root node in their ascending order. The second MCU serially evaluates nodes at levels i=3, 2, and 1. Both the units are capable of carrying out a radius check. The MCU at the top level stops evaluating nodes whenever the SC is violated. The second MCU continues until all the nodes evaluated by the first unit are processed. Notice that the top level MCU is exactly same as that in the staggered architecture. The second MCU has to be reconfigured according to the level of the node it is computing.

Fully serial architecture implementation: In this architecture there is only one MCU that is shared between all the levels of the tree. Notice that the first four cycles are spent to compute the SF/MF. Hence, the first node at the top level is available for computation only at the 5^{th} clock cycle (see Fig. 3.2). Notice that in this scheme the unit has to be reconfigured for i=4, 3, 2, 1.

Fully parallel architecture implementation: In this architecture, there are 16 MCU's operating concurrently. These MCU's evaluate all 16 nodes at a given level in a single cycle. In this architecture the MCU's also have to be reconfigured for i=4, 3, 2, 1. The difference between the MCU's in the parallel and the staggered architectures is that in the parallel architecture, the enumeration is not carried out. The quantization operation (of c_{i+1}), however, is still needed for levels i=1, 2, 3 in the parallel architecture.

V. Results and Discussion

The implementation of the above mentioned architectures was done using a customized *Compare/Select* and *Comparator* blocks and standard cell based design in 100 nm technology [13]. The customized *Compare/Select* block was simulated in

SPICE3 [14] to obtain its delay and power. The customized *Comparator* block is implemented as designed in [18]. Hence the area, delay and power of the comparator as reported in [18] (for a 100 nm technology) are used as such. For the standard cell based design, the delay was computed from a sensitizable timing analysis tool (Sense) [15] and the power was computed using a script written in the SIS [16] logic synthesis environment. This script computes the re-convergence adjusted signal probability, which in turn is used to compute the dynamic power consumption of the design. For both designs, the active area was computed and used to estimate the area of the architectures discussed above.

The serial, the sequential, and the staggered algorithms were simulated under BET scheme. For BET N_{max}=16 for all the three algorithms, and clk_{ave} = 10, 20, 35 for the staggered, the sequential, and the serial algorithms respectively. These parameters were chosen such that the BER performance (Fig. 5.4) for each algorithm is very close to that of the fully parallel algorithm (at uncoded BER of 10^{-3}). Note that all the schemes discussed in this paper use ℓ^1 norm, which incurs a penalty of about 0.4dB-0.5dB.

Fig. 5.1 shows the runtime variability of the sequential, serial, and staggered algorithm as a function of Eb/No. Staggered algorithm clearly outperforms the sequential and the serial algorithms by a factor of almost 2 and 3 respectively.

Compare/Select

The *Compare/Select* block uses the *Longest Prefix Matching* (LPM) circuit designed in [17]. The logic behind the *Compare/Select* block is illustrated in Fig.5.2 with an example. Consider the problem of finding the maximum among three 4 bit operands as shown in Fig.5.2 (a). The LPM looks each column of bits at a time (starting from the MSB) and eliminates the operand that has a '0' in that column, if anyone of the other bits is a '1'. Hence in Fig.5.2 (b) operand '0101' is eliminated. Hence, only the first two operands contend. Since both these operands have the same bit in column 2 (Fig.5.2(c)), the LPM looks for the next column. In column 3, the second operand has a '0' and hence gets eliminated. Thus the LPM declares the first operand as the largest value.

✓	1011	✓	1011	✓	1011	✓	1011
✓	1001	✓	1001	✓	1001	➢	1001
✓	0101	➢	0101	➢	0101	➢	0101
	(a)		(b)		(c)		(d)

Fig.5.2 Example illustrating the Compare/Select logic

Based on the SPICE3 [14] simulation in 100 nm technology [13], the *Compare/Select* had a pre-charge delay of 100 ps and an evaluation delay of 71.92 ps and 73.83 ps for the 4 and 16 operand *Compare/Select* blocks.

Sum-of-Product (SOP)

The remainder of the design is standard cell based, implemented in 100 nm technology. The multi-operand addition is done using a *Sum-of-Product* (SOP) design [19]. The SOP is better than a tree of adders, since it has a single carry chain. The SOP block consists of 3 steps – Partial Product generation, Partial Product reduction using half and full adders and a single final 2 operand binary adder. The final adder is implemented using a fast *Kogge-Stone* adder [20].

Based on Table 5.1, we note that the Staggered approach achieves about half the throughput of the parallel approach, using about a fifth of the area of the parallel scheme. Also, the throughput of the Staggered approach is twice that of the Sequential approach and about four times that of the Serial approach. In terms of the power per decoded symbol vector, the Staggered approach consumes twice the power as Sequential approach and 4 times the power as Serial approach. If we consider the throughput of each approach per unit power (calculated at a SNR of 12dB), the Staggered approach is better than the Serial and Sequential approaches by 10% and better than the parallel approach by 27%. Also the Staggered approach has a throughput per unit area which is 2.3, 1.4 and 1.2 times larger than the Parallel, Serial and Sequential approaches respectively.

VI. CONCLUSIONS

In this paper, we have proposed a novel algorithm/architecture which takes a hybrid approach toward MIMO detection. We have implemented our architecture using an ASIC design flow in a 100nm process technology. Our implementation results are compared with fully serial, fully parallel, and sequential architectures. ASIC estimates (using a 100nm technology) indicate that our approach achieves a guaranteed minimum throughput of 287 Mbps, with an area utilization of 2070 μm^2. Our approach achieves significantly higher throughput than the serial and sequential approaches, and half the throughput of the parallel approach (using a fifth of its area). The throughput per unit power and throughput per unit area are also improved over the parallel, serial and sequential approaches.

REFERENCES

[1] W. Wolniansky, G. Foschini, G. Golden, and R. Valenzuala, "V-BLAST:An architecture for realizing very high data rates over the rich-scattering wireless channel," Proc. IEEE ISSSE 1998, pp.295-300, Sept. 1998.

[2] U. Fincke and M. Pohst , "Improved methods for calculating vectors of short length in a lattice, including complexity analysis" Math. Compute., vol.44, pp463-471, April 1985.

[3] B Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel", IEEE Transactions on Communications, Volume 51, Issue 3, March 2003 Page(s):389 - 399

[4] A. Burg, N. Felber, and W. Fichtner, "A 50 Mbps 4x4 maximum likelihood decoder for multiple-input multiple output systems with QPSK modulation," in Proc. IEEE Int. Conf Electron., Circuits, Syst.(ICECS), vol.1, 2003, pp.332-335.

[5] Burg, A., Borgmann, M., Wenk, M., Zellweger, M., Fichtner, W., Bolcskei, H.,VLSI implementation of MIMO detection using the sphere decoding algorithm, IEEE J. Solid State Circuits, vol.40, pp 1566-1577, July 2005.

[6] Kwan-wai Wong, Chi-ying Tsui, Cheng, R.S.-K., Waiho Mow, A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels, Proc. IEEE ISCAS, Vol. 3, pp273-276.

[7] Z Guo and P Nilsson, "Algorithm and implementation of the K-best sphere decoding for MIMO detection", Selected Areas in Communications, IEEE Journal on, Volume 24, Issue 3, March 2006, pp 491-503

[8] A.Burg, M.Borgmann, M.Wenk, C.Studer and H. Bolcskei, "Advanced Receiver algorithms for MIMO wireless communication", Proceedings, Design Automation and Test in Europe (DATE) Conference, vol 1. March 2006.

[9] L. Barbero and J. Thompson, "Rapid Prototyping of a Fixed-Throughput Sphere Decoder for MIMO Systems", in IEEE International Conference on Communications (ICC '06), Istanbul, Jun. 2006.

[10] C.Hess, M.Wenk, A.Burg, P.Luethi, C.Studer, N.Felber and W.Fitchtner, "Reduced-complexity mimo detector with close-to ML error rate performance", Proc. of the 17th Great Lakes Symposium on VLSI, 2007.

[11] Pankaj Bhagawat, Gwan Choi, "Staggered Sphere Decoder for MIMO detection", submitted to Globecom 2007.

[12] Pankaj Bhagawat, Gwan Choi, "Staggered Sphere Decoder", Technical Report TAMU-ECE-2007-13, March 2007.

[13] Y Cao, T Sato, D Sylvester, M Orshansky and C Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design", Proc. of IEEE Custom Integrated Circuit Conference, June 2000, pp201-204

[14] L Nagel, "SPICE: A Computer Program to Simulate Computer Circuits", University of California, Berkeley UCB/ERL Memo M520, May 1995.

[15] P McGeer, A Saldanha, R Brayton and A Sangiovanni-Vincentelli, "Delay Models and Exact Timing Analysis", in
Logic Synthesis and Optimization, Kluwer Academic Publishers, 1993. pp 167-189.

[16] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis", UCB/ERL M92/41, Univ. of California, Berkeley, CA 94720, May 1992.
[17] B Gamache, Z Pfeffer and S Khatri "A Fast Ternary CAM Design for IP Networking Applications". 12th International Conference on Computer Communications and Networks (IC3N-03), Dallas, TX, October 2003.
[18] E Menendez, M Dumezie, R Garg and S Khatri, "CMOS Comparators for High-Speed and Low-Power Applications". IEEE International Conference on Computer Design (ICCD), Oct 1-4, 2006, San Jose, CA.
[19] S Das and S Khatri, "A Timing-Driven Hybrid-Compression Algorithm for Faster Sum-of-Products". IASTED Fifth International Conference on Circuits, Signals and Systems (CSS) 2007, July 2-4, Banff, Alberta.
[20] P Kogge and H Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations" in IEEE Transactions on Computers, Vol. C-22, Number 8, pp 783-91, 1973.

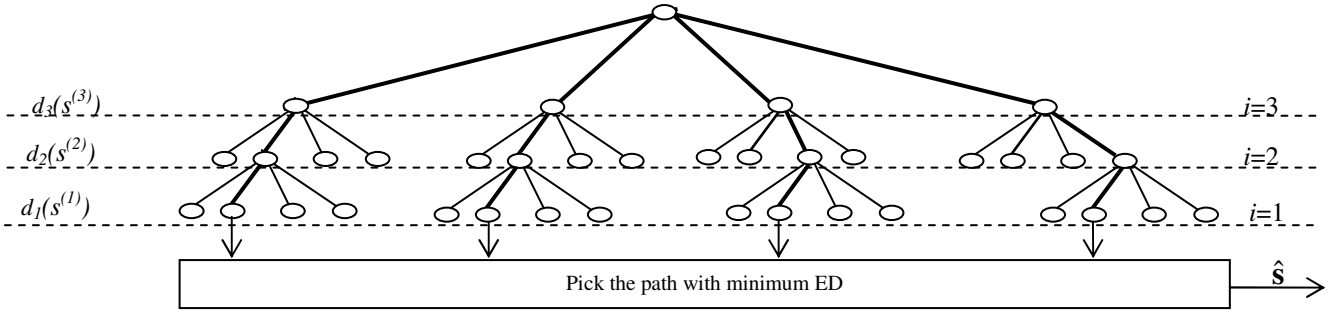

Fig.3.1 Tree Structure for Spherical Search after Reordering.

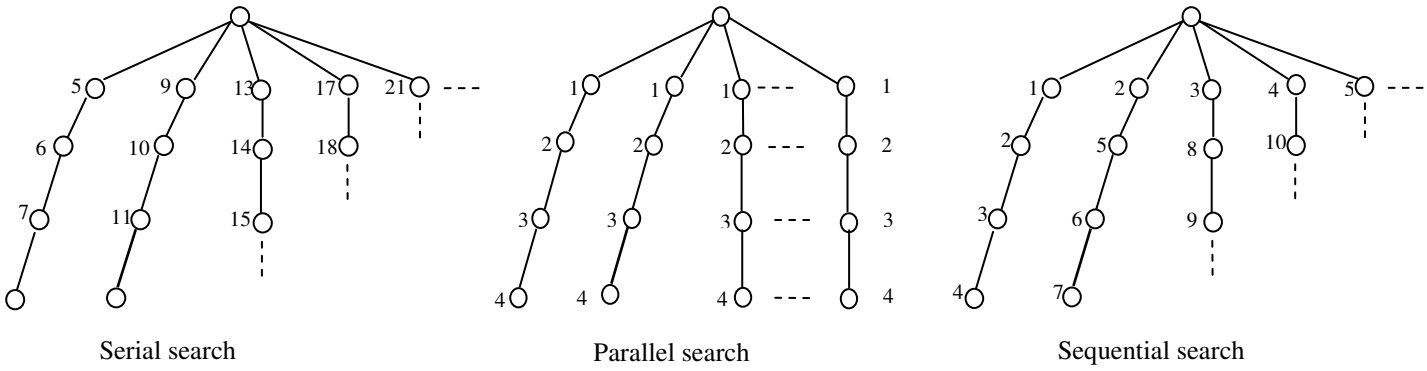

Fig. 3.2 Serial, Parallel, and Sequential Architectural Data-Flow

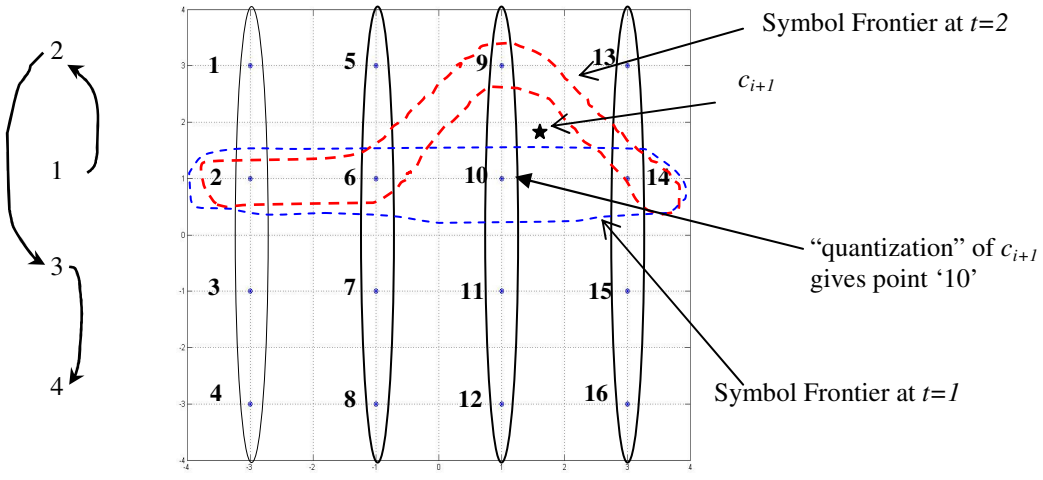

Fig. 3.5 Enumeration Scheme for 16-QAM

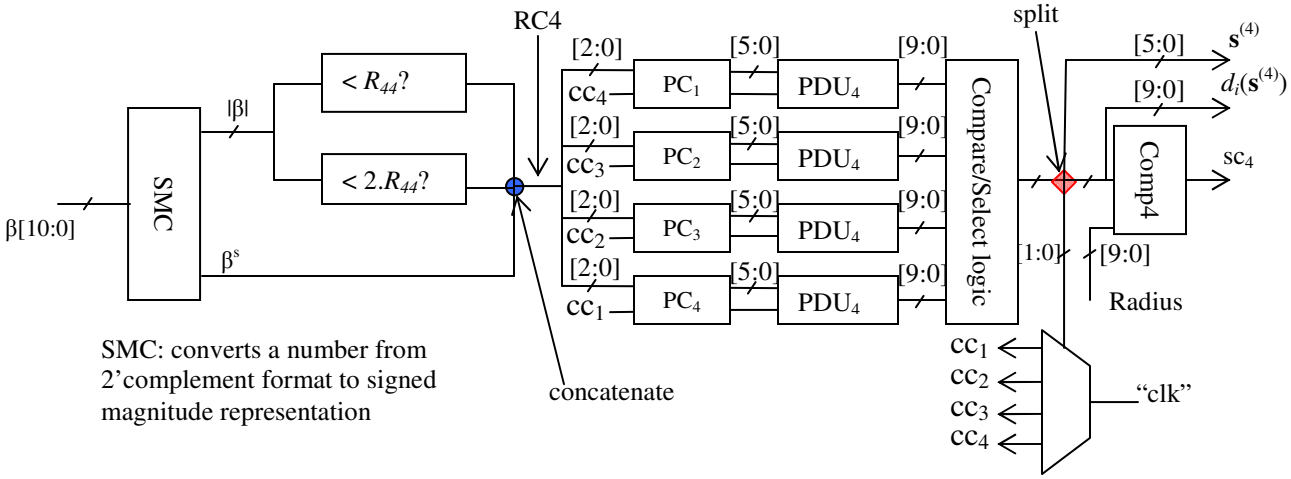

Fig.4.2 Details of MCU_4

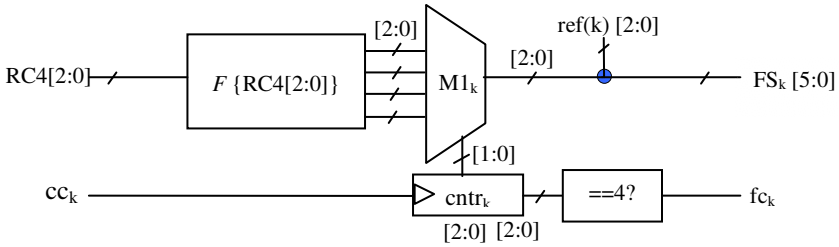

Fig. 4.3 Details of PC_k k=1,2,3,4

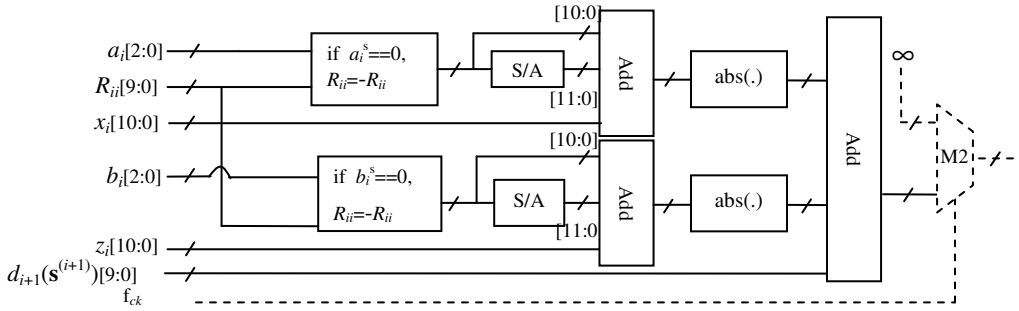

Fig. 4.4 Details of the PDU

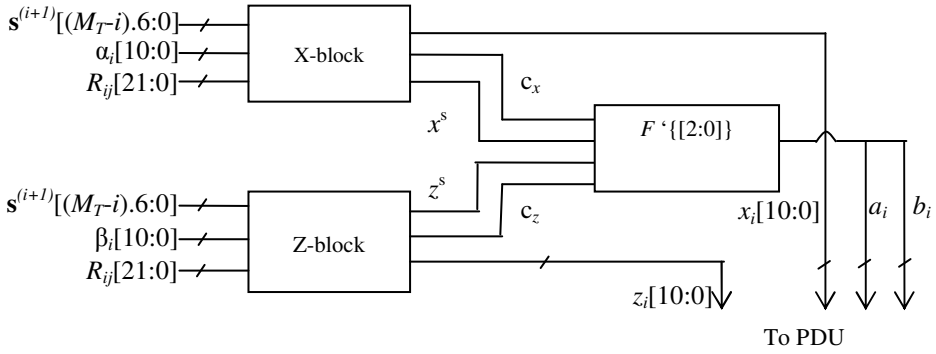

Fig. 4.6 Details of Quantize Block

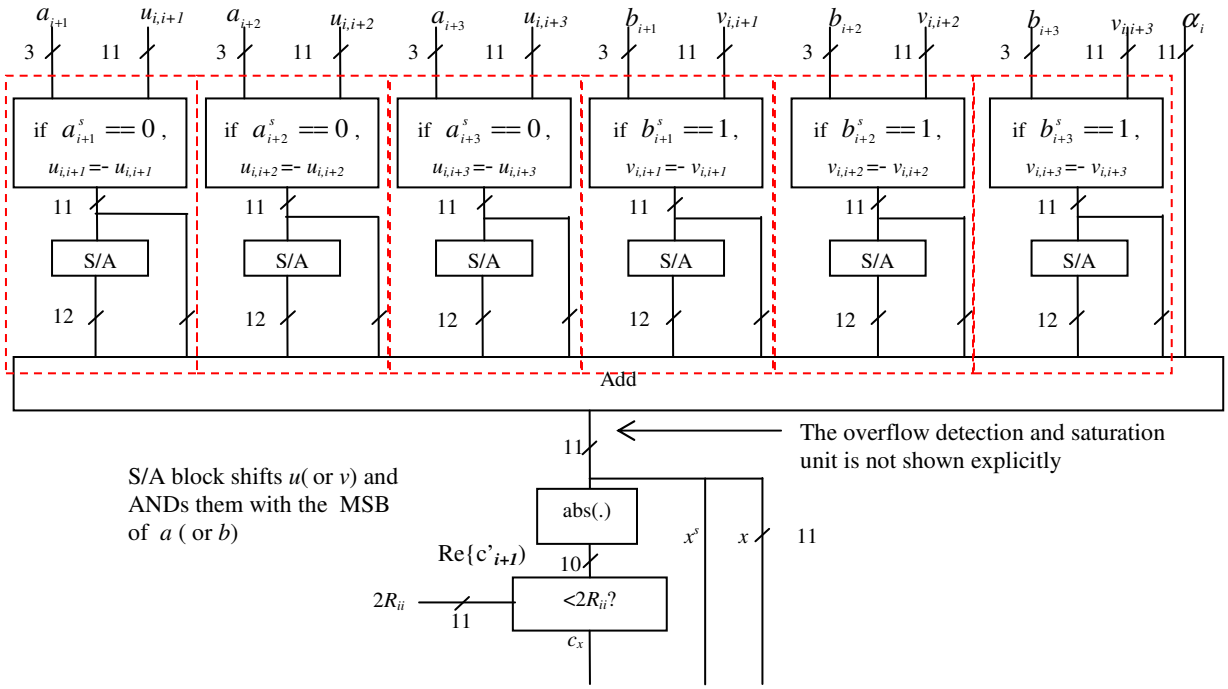

Fig. 4.7 Details of the X-block

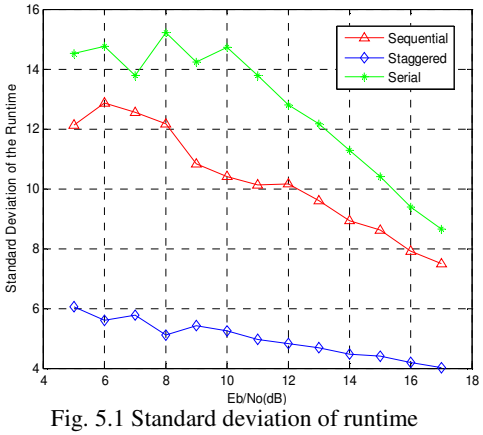

Fig. 5.1 Standard deviation of runtime

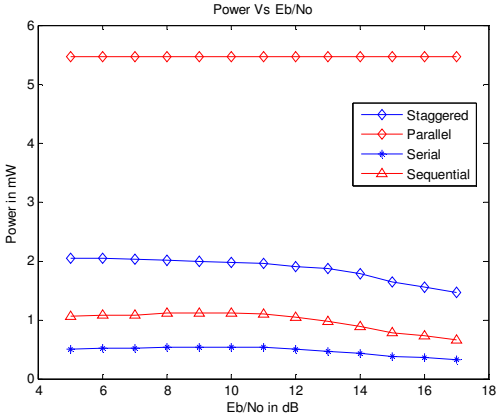

Fig 5.3 Power consumption per vector symbol

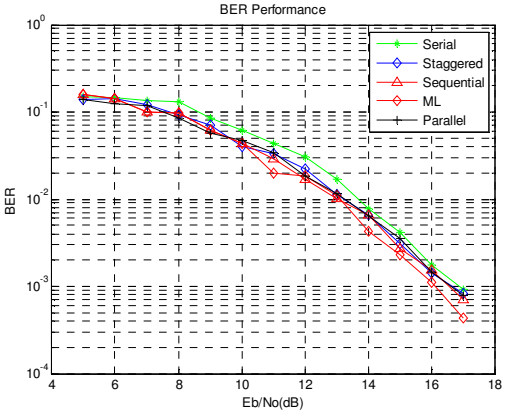

Fig 5.4 BER as a function of Eb/No.

Architecture	f_{clk} (MHz)	Throughput (T) (Mbps)	Area (μm^2)	T/Power (Mbps/mW)	T/Area (Mbps/μm^2)
Staggered [Ours]	179.53	287.25	2069.73	150.31	0.14
Parallel [9]	161.55	646.20	10350.88	118.29	0.06
Serial	151.29	69.16	693.34	136.89	0.10
Sequential [10]	179.53	143.62	1214.33	136.91	0.12

Table 5.1 Delay and Area of various architectures