5-20-2005

Design and Implementation of an Universal Lattice Decoder on FPGA

Swapna Kura
University of New Orleans

DESIGN AND IMPLEMENTATION OF AN UNIVERSAL LATTICE DECODER ON FPGA

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
The Department of Electrical Engineering

by

Swapna Kura

B.Tech J.N.T.U, 2001

May 2005

ACKNOWLEDGEMENTS

I would like to express my special thanks to Dr. Jing Ma for being my advisor throughout my thesis research. I appreciate her patience, guidance and supervision of my work which helped me in progressing in right path.

I acknowledge Dr. Bhaskar Kura for his support throughout my graduate program without which it would have been impossible for me to get thru my master's degree. His patience, guidance and insight served as invaluable assets in both my personal and academic lives.

I would also express my sincere thanks to Dr. Xinming Huang and Dr. Edit Bourgeois for their willingness to serve as members in my thesis committee.

I would express my heartfelt thanks to my parents and all family members. Their blessings and love were always with me and encouraged me in stepping forward in life.

I would thank my colleagues for being eager and prompt enough to help me when I needed them. Finally, I would like to thank all my friends and cousins for their encouragement and motivation.

GLOSSARY OF ABBREVIATIONS

MIMO – Multiple Input Multiple Output

FPGA – Field Programmable Gate Arrays

PLD – Programmable Logic Device

ASIC – Application Specific Integrated Circuit

IC – Integrated Chip

SOC – System-On-Chip

FSM – Finite State Machine

AWGN – Additive White Gaussian Noise

PAM – Pulse Amplitude Modulation

BER – Bit Error Rate

ML – Maximum Likelihood

DSP – Digital Signal Processor

VHDL – Very High speed integrated Description Language

RTL – Register Transfer Level

ISE – Integrated Software Environment

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In wireless communication, MIMO (multiple input multiple output) is one of the promising technologies which improves the range and performance of transmission without increasing the bandwidth, while providing high rates. High speed hardware MIMO decoders are one of the keys to apply this technology in applications. In order to support the high data rates, the underlying hardware must have significant processing capabilities. FPGA improves the speed of signal processing using parallelism and reconfigurability advantages.

The objective of this thesis is to develop an efficient hardware architectural model for the universal lattice decoder and prototype it on FPGA. The original algorithm is modified to ensure the high data rate via taking the advantage of FPGA features. The simulation results of software, hardware are verified and the BER performance of both the algorithms is estimated. The system prototype of the decoder with 4-transmit and 4-receive antennas using a 4-PAM (Pulse amplitude modulation) supports 6.32 Mbit/s data rate for parallel-pipeline implementation on FPGA platform, which is about two orders of magnitude faster than its DSP implementation.

1 INTRODUCTION

1.1 Motivations and Background

Although wireless technologies have been around for a while, there has been a recent and rapid surge in the evolution of new standards that enable and accelerate the convergence of telecommunications and IP networking to provide new multimedia services. To keep up with the demands of wireless network services, the capacities of systems are increased. The most brute-force approach to increasing wireless data rate is to use more frequency channels to increase modulation rate [Jones 2003]. This "channel bonding" approach will not meet the needs of wireless network consumers for the following reasons: First, while channel bonding increases data rate, it decreases the transmission range for the same transmit power. Second, channel bonding robs channels from other systems that operate nearby.

MIMO (multiple input multiple output) antenna technology is considered as one of the solutions to support the wireless network services. It essentially multiplies data throughput, with a simultaneous increase in range and reliability, without consuming any extra frequency spectrum [Jones 2003]. The multi-antenna wireless communication systems are capable of providing data transmission at potentially very high rates. Furthermore, to secure high reliability of the data transmission, special attention has to be given to the receiver design. The data streams are separated at the receiver using algorithms that rely on estimates of all channels between each transmitter and each receiver. The low complexity suboptimal detection algorithm for MIMO signals was the Vertical Bell Labs Layered Space-Time (VBLAST) algorithm. This is an iterative cancellation method that depends on computing a matrix inverse to solve the zeroforcing function [Jones 2003]. While the iterative detection can increase receiver sensitivity, there are substantial problems with a real implementation.

The optimal detection strategy for a MIMO receiver is to perform a maximum-likelihood search over all possible transmitted symbol sets. ML decoding is equivalent to finding the closest lattice point to the received point in a lattice constellation. ML detection at the receiver becomes an essential part in high-performance MIMO communication systems [Burg 2004]. Thus, ML decoding algorithms and their architecture are active research areas in wireless communication that motivated the research in MIMO systems.

For decoding the lattices with no regular structure at the receiver follows two main branches. Pohst [Pohst 1981] in 1981 examined lattice points lying inside a hyper sphere, whereas Kannan in 1983 used a rectangular parallelepiped. Both methods later appeared in revised and extended versions. Pohst method is intended as practical tool while Kannan's is a theoretical tool. In [Viterbo 1999], a technique referred to as the "sphere decoding" (based on the Fincke-Pohst algorithm) was proposed for lattice code decoding [Eriksson 2002]. This performs a bounded distance search among the lattice points falling inside the sphere centered at the received point.

The sphere decoder provides the maximum-likelihood estimate of the transmitted signal sequence with complexity comparable, at high signal-to-noise ratios (SNRs), to VBLAST nulling/canceling algorithm [Bertrand 2003]. It is later stated that sphere decoding often significantly outperforms heuristic nulling and canceling. Developing an efficient sphere decoder with reduced complexity has received significant attention due to its applications to wireless communications as in [Viterbo 1999]. However, most modifications suggested are well suited for implementations using DSPs, for example BLAST system [Adjoudani 2003]. In the VBLAST algorithm the front end of the receiver is implemented on FPGA whereas actual decoder function is implemented on a DSP processor.

As the performance requirements of today's communication systems are outstripping the capabilities of general-purpose DSP processors, the need for DSP implementations to seek hardware solution arises [Dan 2004]. FPGAs provide an ideal platform for DSP implementation, combining the reprogrammability, architectural flexibility, and support of parallelism. FPGA-based hardware platforms also meet the critical requirements such as processing speed, time-to-market, system integration etc. Due to the significant processing capabilities of FPGAs, high data rates are ensured for signal processing applications implemented on FPGAs. With advanced FPGA architectures such as the Xilinx Virtex-II devices, a new hardware alternative is available for DSP implementations combining all the benefits of DSP processors with the performance advantages of ASICs [Dan 2004].

The key advantages of FPGAs when compared to DSP implementations include performance, integration, and customization. Because of this, an FPGA-based solution of a high-performance DSP system will typically have fewer devices than a processor-based one resulting in less power consumption, lower overall cost, and significantly less board area [Kevin 2003]. Due to the support of parallelism, FPGAs achieve huge gains in performance compared to DSP implementations. The computational throughput is also at least an order of magnitude higher with FPGA platforms.

Comparing to ASICs, FPGAs are reprogrammable and when combined with HDL design flow can greatly reduce the design and verification cycle. In addition to this, increased time-to-market demands, low FPGA development costs, and FPGA capacities well in excess of million gates are increasing the number of applications of FPGAs in programmable form [ED 2000].

1.2 Research Objective

The main objective of this thesis is to develop an efficient architecture of a sphere decoder simulated in VHDL and prototype it on device technology of XILINX VirtexII-1000 FPGA platform. The architectural model deploys the parallelism offered by FPGA and ensures the high data rate of the MIMO system.

1.3 Contribution of Thesis

The main contribution in this thesis is the design and implementation of an universal lattice decoder on FPGA. Firstly, the functionality of original sphere decoding algorithm is examined using Matlab simulations. Then a VHDL model is developed for core decoder function and simulated at RTL level of abstraction using Mentor Graphics' Modelsim SE 5.8a. Based on the simulation results, we observed that the original sphere decoder is not feasible for parallel-pipeline implementation. Modifications are applied to the original algorithm and as a result an improved form of universal lattice decoder is proposed. Functionality testing procedure similar to that of original algorithm is carried out for the improved algorithm. Based on the data dependency analysis, a parallel-pipeline architectural model is developed for the improved sphere decoding algorithm. Both sequential and parallel-pipeline architectural models are developed in VHDL and are simulated at RTL level of abstraction. All the hardware architectural models are prototyped on a XC2V1000-6FF896C, a device technology of Xilinx VirtexII-1000 FPGA platform. BER performance of original and improved sphere decoding algorithms is compared for both fixed point and floating point simulations. For a 4-transmit and 4-receive antennas system using 4-PAM transmitted signals, a decoding throughput of 6.32

Mbits/s is achieved. The performances of FPGA and DSP implementations are compared. The details of the results are presented in Chapter 6.

1.4 Organization of Thesis

Chapter 2 introduces FPGA and MIMO channels. Their concepts and features are explained in detail. Chapter 3 describes the original sphere decoding algorithm. It also discusses the data flow path by partitioning the algorithm into various states. Eventually the Finite state machine (FSM) design is proposed, state transitions are discussed and simulation times for each state are also presented. Chapter 4 presents the modifications applied to the original algorithm by avoiding square root. Thus, an improved sphere decoding algorithm is developed. In addition, data dependency analysis of the improved sphere decoder is discussed. Chapter 5 gives the detail description of the FSM design for the improved sphere decoder. Also, the parallel-pipeline structure of sphere decoding algorithm is described, and the design optimization techniques are presented. Finally Chapter 6 gives the experimental results obtained during the thesis research.

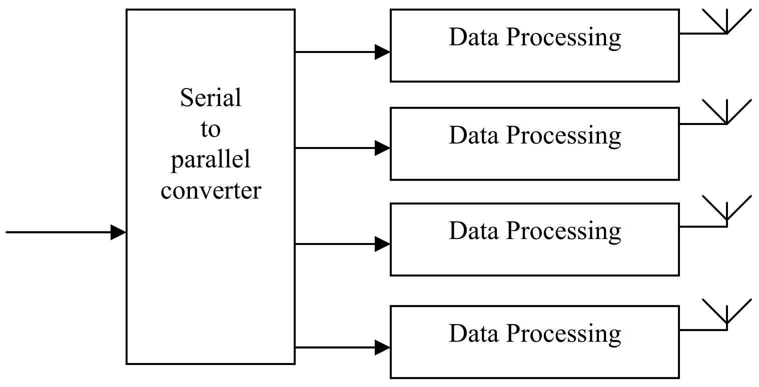
2 FPGAs and MIMO Channels

This chapter gives a brief introduction of FPGA and MIMO channel. A detail description explaining the basic concept, features is also given.

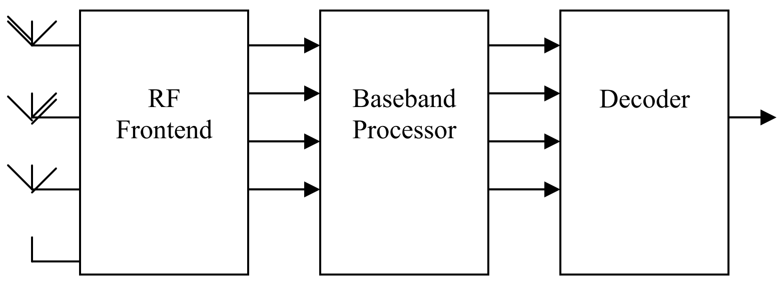
2.1 MIMO Channels

The ever increasing demands of multimedia services have led to high speed wireless communications with much higher data rates. Multiple transmit and receive antennas are most likely the dominant solution in future broadband wireless communication systems as they are the key technology to produce high rates.

MIMO systems consist of an array of transmit and receive antennas combined in such a way that the quality (bit error rate) or the rate (Bit/sec) of the communication is improved [Gesbert 2005]. Use of multiple transmit and/or receive antennas produce enormous gain in spectral efficiency by exploiting a rich multi-path fading environment and increased the system capacity without requiring an increase in the transmit power or bandwidth of the system. These channels also provide radio-link reliable communication when multiple users are sharing the spectrum by reducing the fading environments which is sometimes possible through the use of diversity technique. The spatial diversity in the MIMO systems is to send the signals that carry the same data through different paths. Due to this multiple independently faded replicas of the same data symbol can be obtained at the receiver end and hence more reliable reception is achieved. If the path gains between individual transmit-receive antenna pairs fade independently, the channel matrix well conditioned with high probability such that multiple parallel spatial channels are created [Zheng 2003]. The spatial multiplexing of the MIMO system which helps in achieving high data rates is to split a single data stream into multiple sub-streams, and each of these independent sub-streams is transmitted in parallel through those spatial channels with same frequency. In wireless channels the data streams transmitted from multiple transmit antennas can be separated, thus leading to the parallel data paths. Under these conditions, the capacity of the radio channel grows linearly with the number of antennas used either at the transmitter or receiver. The scattering of signals, which interferes with one another in a single-antenna system, if exploited properly can enhance, rather than degrade the transmission accuracy and huge channel capacities are intended to achieve [Garrett 2002]. Multi-path propagations can make the output of receiver antenna to be equal to a linear combination of the multiple transmitted data streams. Thus with sophisticated coding at the transmitter and substantial signal processing at the receiver, the MIMO channel can be provisioned for higher data rates [Love 2004].



(a)



(b)

Figure 2.1: A MIMO system. (a) MIMO Transmitter. (b) MIMO receiver

Figure 2.1 shows a schematic representation of this multiple input multiple output (MIMO) system [Adjoudani 2003]. The complexity of the MIMO systems is involved in designing an optimal receiver for the system. The optimal receiver is a maximum-likelihood sequence detector and is computationally complex due to system parameters like number of antennas and type of constellation used. Therefore the optimal detection strategy is to equivalent to performing a maximum-likelihood search over all possible transmitted symbol vectors. When there is a perfect knowledge of channel state information at the receiver the sphere decoding algorithm is considered as the maximum likelihood decoder.

There are two typical lattice decoding algorithms. One is the Pohst strategy based algorithm [Viterbo 1999]. This tries to find lattice points inside a sphere of given radius. Another is the Schnorr-Euchner strategy based algorithm [Eriksson 2002]. This method divides the lattice into hyper-planes and starts the search for the closet point in the nearest hyper-plane.

2.2 Field Programmable Gate Array (FPGA)

FPGA is an integrated circuit that contains configurable (programmable) logic blocks and interconnects between these blocks. In other words, it is a general purpose chip which can be reconfigured any number of times to carry out specific hardware functions. It provides an opportunity of instantaneous changes in designing and debugging. It allows for system reuse, parallel design and SOC design. This is the result of combinatorial features of PLD and ASIC. PLD is a digital IC that can be programmed by the user to perform a wide variety of logical operations. ASIC is an IC product customized to perform specific functions to a particular system or application. Like PLD, FPGA is completely prefabricated and contain special features for customization. FPGA is subclass of ASIC which can be reprogrammable. Designs started in

FPGA can be migrated to ASICs. A comparison between ASIC, FPGA, and DSP implementations of the any decoder shows that the performance of FPGA-based designs lean more toward that of ASICs but retain flexibility more like DSP [Gregory 1999]. ASICs provide the most optimized hardware implementation of an algorithm. Using a dedicated ASIC for each mode of radio leads to a very large silicon area. DSPs have excellent programmability but cannot handle the complex algorithms at the required speeds with reasonable power consumption. FPGAs on the other hand use hardware reconfiguration, which allows implementation of complex high-speed algorithms [Srikanteswara 2003]. Compared to FPGA implementation, DSP implementations require low cost and less development time. But once an efficient architecture is developed and the parallelism of the algorithm is explored, FPGAs can be used to significantly improve the speed of the signal processing or wireless communication systems. Thus, FPGA is considered as an ideal platform for performing the computationally complex operations for reasons of performance, power consumption and configurability. Compared to DSP chip, parallelism is an additional feature in FPGA. The architecture of the Xilinx Virtex-II FPGA is shown in Figure 2.2. The device is organized as an array of logic elements and programmable routing resources used to provide the connectivity between the logic elements, FPGA I/O pins and other resources such as on-chip memory, delay lock loops and embedded hardware multipliers.

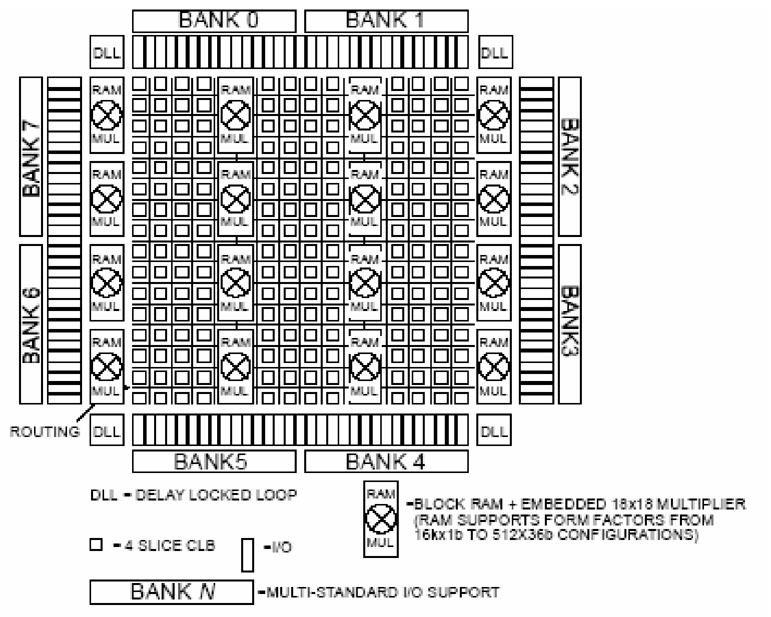

BANK 0
BANK 1
BANK 2
BANK3
BANK 4
BANK5
BANK 6
BANK 7
DLL
RAM
MUL
ROUTING
DLL = DELAY LOCKED LOOP
= 4 SLICE CLB
=I/O
=BLOCK RAM + EMBEDDED 18x18 MULTIPLIER (RAM SUPPORTS FORM FACTORS FROM 16kx1b TO 512X36b CONFIGURATIONS)
BANK N
=MULTI-STANDARD I/O SUPPORT

Figure 2.2: Virtex-II FPGA architecture [Chris]

The FPGA resources of particular interest to the signal processing engineer are configurable dual-port block memories, distributed memory, and the multiplier array [Xilinx 2003]. The multiplier array is composed of 18x18-bit precision multipliers for addressing advanced sign al processing applications. The smallest Virtex-II device provides a modest 4 multipliers while the largest supplies an impressive 192 multipliers [Chris].

3 Sphere Decoding Algorithm

This chapter describes the Pohst's lattice point enumeration algorithm [Viterbo 1999] widely known as sphere decoding, and also called universal lattice decoding. The data flow path and state transition details are elaborated. High level description of the algorithm and decoder architecture scheduling are also elucidated. The FSM diagram is shown. The table showing the processing time taken by each state is presented.

3.1 The Sphere Decoder

In digital communications, lattice codes generate signal constellations for high rate transmission. The high-rate data streams and spatial multiplexing leave MIMO technology as the most desirable option in communication systems. The complexity of MIMO systems is involved in designing a MIMO receiver. For designing a MIMO receiver, a ML decoding is employed. ML decoding of a arbitrary lattice code used over an additive white Gaussian noise (AWGN) channel is equivalent to finding the closest lattice point to the received point. To reduce the complexity of an exhaustive search procedure, the bounded distance search among the lattice points is formulated. Therefore, for decoding the optimal receiver output of these MIMO systems, Pohst's enumeration based sphere decoding algorithm searches for the closest lattice point to the received point within the sphere with radius $\sqrt{C}$. The center point i.e., the signal or vector at the receiver is known before hand. The choice of C is very crucial to the speed of the algorithm. In practice the choice of C can be adjusted according to the noise variance so that the probability of a decoding failure reported is negligible. The complexity of the algorithm is independent of the lattice dimension size, which is very useful for high data rate transmission [Viterbo 1999]. Pohst first proposed the strategy for enumerating all the lattice points within the sphere with a certain radius in [Pohst 1985]. Then it was introduced into the field of digital communications for the first time in [Viterbo 1993] and further analyzed in [Viterbo 1999].

3.1.1 Maximum-Likelihood Criterion

Considering a MIMO system with m transmit and n receive antennas, and a perfect knowledge of channel state information is known at the receiver then the maximum likelihood decoding requires minimization of metric

$$\sum_{i=1}^{n} \| r_i - x_i \|^2 \ \forall \text{ valid lattice points.} \qquad \text{Equation (3-1)}$$

Where, $r = uM + V$, the received vector. When the data streams interfere with each other in the channel and is distorted by an AWGN component V then, the resultant is the received vector.

u is the transmitted signal.

M is the channel matrix which generates the lattice.

V is the AWGN noise vector with zero mean and N_0 variance.

x is the information symbol vector mapped into the output vector which is the received vector r. Thus x is considered as one of the transmitted lattice code points.

The representation of lattice points is given as $\{x = uM\}$ where $u = \{u_1, u_2, ... u_n\}$ is the integer component vector, and M is the channel transfer matrix which generates the lattice Λ structure. Any lattice Λ is given as the combination of set of basis vectors represented by $v = \{v_1, v_2 .... v_n\}$

If $v_i = (v_{i1}, v_{i2} .... v_{ib})$, $i = 1 \ldots\ldots\ldots n$, and b is the dimension of the lattice then the generator matrix M of the lattice Λ is defined as

$$M = \begin{pmatrix} v_{11} & \cdots & v_{1b} \\ \vdots & & \vdots \\ v_{n1} & \cdots & v_{nb} \end{pmatrix}$$

The same lattice structure Λ can have any number of generator matrices. For example the matrix of the form $M^{'} = TM$, where T is an integer orthogonal matrix $(\det(T) = \pm 1)$, is also the generator matrix of the lattice Λ . Assuming matrix M to be non-singular square matrix i.e., $n = b$, the Gram matrix of the lattice Λ is given by

$$G = MM^{T} = \begin{pmatrix} g_{11} & \cdots & g_{1b} \\ \vdots & & \vdots \\ g_{b1} & \cdots & g_{bb} \end{pmatrix}$$

The elements of the matrix G are the Euclidean square products of the pairs of vectors of the lattice basis.

3.1.2 ML Decoding In Sphere Decoder

The lattice decoding algorithm attempts to minimize the metric in Equation (3-1) but employs the bounded distance search procedure. Thus it searches through the points of lattice that are falling inside the sphere of radius $\sqrt{C}$ and centre at the received point.

Thus, sphere decoding problem is to solve

$$\min_{x \in \Delta} \| r - x \| = \min_{w \in r - \Delta} \| w \| \qquad \text{Equation (3-2)}$$

So we search for the shortest vector w in the translated lattice $r - \Lambda$ in the n-dimensional Euclidean space R^{n}. We write

$$x = uM \text{ with } u \in z^{n}$$

$$r = \rho M \text{ with } \rho = (\rho_1, \rho_2 ... \rho_n) \in R^{n}$$

$$w = \xi M = \sum_{i=1}^{n} \xi_i v_i \text{ with } \xi = (\xi_{1,} \xi_2 .... \xi_n) \in R^{n} \text{ and } \xi_i = \rho_i - u_i, i = 1,....n$$

Where, ρ and ξ are real vectors.

$\rho = rM^{-1}$ i.e., ρ is equal to the matrix product of the received vector r, and the inverse of generator matrix M^{-1}. ξ defines the translated coordinated axes in sphere of the integer component vectors u of the cubic lattice Z^n

3.2 Flow-Chart

The flow chart showing of a Lattice decoding algorithm [Viterbo 1999] or a Universal lattice decoder is shown in Figure 3.1. The lattice decoding algorithm can be divided into two parts (1) Pre-processing part (2) Decoding part.

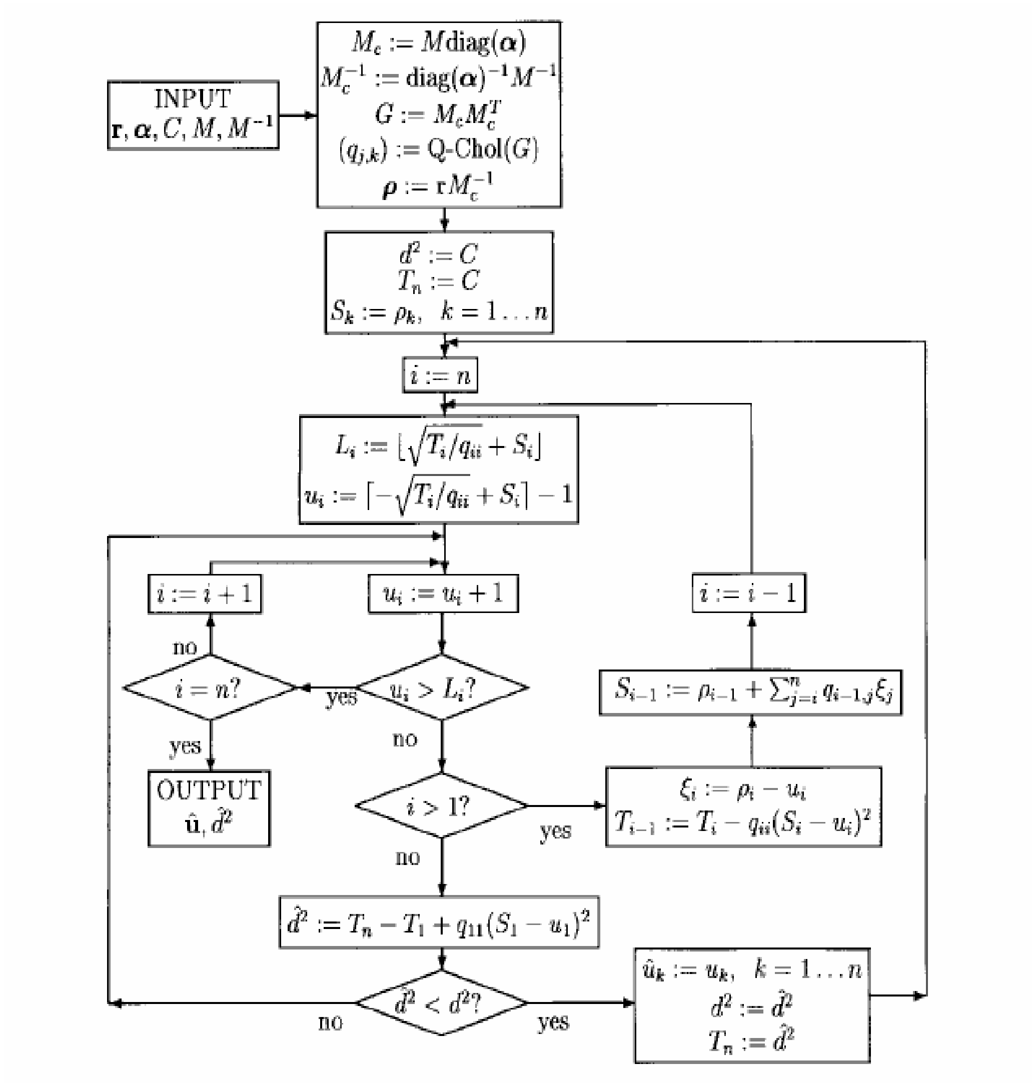

INPUT
$\mathbf{r}, \boldsymbol{\alpha}, C, M, M^{-1}$
$M_c := M\mathrm{diag}(\boldsymbol{\alpha})$
$M_c^{-1} := \mathrm{diag}(\boldsymbol{\alpha})^{-1}M^{-1}$
$G := M_c M_c^T$
$(q_{j,k}) :=$ Q-Chol(G)
$\boldsymbol{\rho} := \mathbf{r}M_c^{-1}$
$d^2 := C$
$T_n := C$
$S_k := \rho_k,\ k = 1\ldots n$
$i := n$
$L_i := \lfloor\sqrt{T_i/q_{ii}} + S_i\rfloor$
$u_i := \lceil-\sqrt{T_i/q_{ii}} + S_i\rceil - 1$
$i := i + 1$
$u_i := u_i + 1$
$i := i - 1$
no
$i = n?$
yes
$u_i > L_i?$
$S_{i-1} := \rho_{i-1} + \sum_{j=i}^{n} q_{i-1,j}\xi_j$
yes
no
OUTPUT
$\hat{\mathbf{u}}, \hat{d}^2$
$i > 1?$
yes
$\xi_i := \rho_i - u_i$
$T_{i-1} := T_i - q_{ii}(S_i - u_i)^2$
no
$\hat{d}^2 := T_n - T_1 + q_{11}(S_1 - u_1)^2$
$\hat{d}^2 < d^2?$
no
yes
$\hat{u}_k := u_k,\ k = 1\ldots n$
$d^2 := \hat{d}^2$
$T_n := \hat{d}^2$

Figure 3.1: Flowchart of a Sphere decoding algorithm [Viterbo 1999]

3.2.1 Pre-Processing

The pre-processing stage of the sphere decoding algorithm involves the complex computations like Cholesky decomposition of the Gram matrix *G*, finding inverse and transpose of generator matrix *M*. The resultant matrices are passed to the decoding part where they are further exploited to carry on other computations, thereby reducing the complexity of the decoding part. The variables and specialized functions used at this stage are described in detail below.

An inverse matrix of the lattice generator matrix is computed. Another important function carried out in the preprocessing stage in the algorithm is the Cholesky factorization of the Gram matrix G . Gram matrix is equal to the product of lattice generator matrix M and its transpose,

$G = MM^T$ yields $G = R^T R$ where, R is the upper triangular matrix.

From the algorithm $(q_{j,k})$ is the element of Cholesky factor matrix.

3.2.2 Decoding

In the decoding part, the integer component of lattice point vector *u* closest to the transmitted signal constellation *x* is found as an output when the Cholesky factor matrix $(q_{j,k})$, the square radius of the sphere *C* and the received vector with respect to lattice ρ are taken as inputs.

Considering the metric properties of the lattice, we can say that the minimum squared Euclidean distance between any two points of lattice equals the minimum of the quadratic form $Q(\xi)$.

$$Q(\xi) = \xi G \xi^T = \xi M M^T \xi^T \qquad \text{Equation (3-2)}$$

If the lattice point being searched is within the sphere with square radius C and centered at the received point then

$$\|w\|^2 = Q(\xi) = \xi M M^T \xi^T = \sum_{i=1}^{n}\sum_{j=1}^{n} g_{ij}\xi_i\xi_j \le C \qquad \text{Equation (3-3)}$$

Thus the sphere of square radius C and centered at the received point is transformed into an ellipsoid centered at origin of the new coordinate system defined by ξ.

Cholesky factorization yields $G = R^T R$, where R is an upper triangular matrix. By further analyzing the above equations we get

$$Q(\xi) = \xi R^T R \xi^T = \|R\xi\|^2 = \sum_{i=1}^{n}(r_{ii}\xi_i + \sum_{j=i+1}^{n} r_{ij}\xi_j)^2 \le C \qquad \text{Equation (3-4)}$$

Substituting $q_{ii} = r_{ii}{}^2$ and $q_{ij} = r_{ij} / r_{ii}$ for $i = 1,\dots, n, j = i + 1,\dots, n$, we can write (3-4) as follows

$$Q(\xi) = \sum_{i=1}^{n} q_{ii}(\xi_i + \sum_{j=i+1}^{n} q_{ij}\xi_j)^2 \le C \qquad \text{Equation (3-5)}$$

We find the equations of the border of the ellipsoid to estimate the upper and lower bounds of the integer component value u_i at the i^{th} layer. Therefore the ranges for the integer component value at i^{th} layer are given by

$$\left\lceil -\sqrt{\frac{1}{q_{ii}}\left(C - \sum_{l=i+1}^{n} q_{ll}(\xi_l + \sum_{j=l+1}^{n} q_{lj}\xi_j)^2\right)} + \rho_i + \sum_{j=i+1}^{n} q_{ij}\xi_j \right\rceil \le u_i \le$$
$$\left\lfloor \sqrt{\frac{1}{q_{ii}}\left(C - \sum_{l=i+1}^{n} q_{ll}(\xi_l + \sum_{j=l+1}^{n} q_{lj}\xi_j)^2\right)} + \rho_i + \sum_{j=i+1}^{n} q_{ij}\xi_j \right\rfloor \qquad \text{Equation (3-6)}$$

Thus the upper bound, L_i and the index, u_i are simplified as follows

$$L_i = \left\lfloor \sqrt{T_i / q_{ii}} + S_i \right\rfloor \qquad \text{Equation (3-7)}$$

$$u_i = \left\lceil -\sqrt{T_i / q_{ii}} + S_i \right\rceil - 1 \qquad \text{Equation (3-8)}$$

Where, the variables S_i and T_i are written as

$$S_i(\xi_{i+l}......\xi_n) = \rho_i + \sum_{l=i+1}^{n} q_{il}\xi_l \qquad \text{Equation (3-9)}$$

$$T_{i-1} = T_{i-1}(\xi_i.....\xi_n) = C - \sum_{j=l+1}^{n} q_{lj}\xi_j = T_i - q_{ii}(S_i - u_i)^2 \qquad \text{Equation (3-10)}$$

Thus the variables S_i, T_i and one of the outputs of the pre-processing part q_{ii} are used to determine and recursively update the values of bounds.

The index u_i is initially fixed at the lower bound and incremented in steps until it exceeds the upper bound of that layer. Search procedure starts at the bottom layer i.e., at $i = 4$ and continues switching the layers step by step by checking various conditions at each layer until it reaches the top layer and a valid lattice point vector is reported. When the vector inside the sphere is found, its square distance from the center is computed which is given by

$$\hat{d}^2 = C - T_1 + q_{11}(S_1 - u_1)^2 \qquad \text{Equation (3-11)}$$

This value is compared to the minimum square distance d^2 (initially set equal to *C*) found so far in the search. If it is smaller then we have a new candidate closest point and new value for d^2 updated with $\hat{d}^2$. Thus the search continues like this until all the vectors inside the sphere are tested.

If no point in the sphere is found the sphere is declared empty and the search fails. In this case the squared radius *C* must be increased and the search is restarted. Thus finally we search the lattice point closest to received point.

The advantage of this method is that we never test the vectors which are present outside the sphere.

3.3 Decoding Procedure

The original sphere decoding algorithm performs step-by-step procedure as follows,

The inputs are C, ρ, Q and output is $\hat{u}$

Step 1. (Initialization)

Set $i = n, T_n = C, d^2 = C$ (current sphere square radius) and

$$S_k = \rho_k, k = 1.........n$$

Step 2. (Bounds on index u_i)

Compute the upper and lower bounds. Assign the upper bound to L_i and the lower bound to index u_i initially. Thus

$$L_i = \left\lfloor \sqrt{T_i / q_{ii}} + S_i \right\rfloor$$

$$u_i = \left\lceil -\sqrt{T_i / q_{ii}} + S_i \right\rceil - 1$$

Step 3. (Natural spanning of the interval)

Increment the index u_i by one step, i.e., $u_i = u_i + 1$

If $u_i \le L_i$ and $i > 1$, i.e., the index is within the range and layer is not the top layer then go to Step 5, else if $u_i \le L_i$ and $i = 1$, i.e., the index of the top layer is within the bound then go to Step 6, else if $u_i > L_i$ go to Step 4.

Step 4. (Increase *i*: move one level down)

If $i = n$ terminate, i.e., the end of the search procedure is reached and closest lattice point to received point is found, else set $i = i + 1$, i.e., the search procedure goes one level down in the hierarchy, and go to Step 3.

Step 5. (Decrease *i*: move one level up)

$$\text{Let } \xi_i = \rho_i - u_i$$

$$S_{i-1} = \rho_{i-1} + \sum_{l=i}^{n} q_{i-1,l}\xi_l$$

$$T_{i-1} = T_i - q_{ii}(S_i - u_i)^2$$

$i = i - 1$ and go to Step 2.

The variables needed to recursively update the lower and upper bounds are computed at this step and the search procedure goes one layer up in the hierarchy to re-compute the upper bound and index u_i.

Step 6. (A valid point is found)

Compute $\hat{d}^2 = C - T_1 + q_{11}(S_1 - u_1)^2$, the square distance of the vector found from the center. Then compare this value to the minimum square distance d^2 i.e., If $\hat{d}^2 < d^2$ then save the lattice point, $\hat{u}_k = u_k, k = 1....n$ and reduce the search area by assigning the minimum square distance value d^2 with $\hat{d}^2$ and the variable T_n at the bottom layer with $\hat{d}^2$ and again set $i = n$.

Then go to Step 2 repeat the whole process once again. Else go to Step 3, where the index value $d^2 = \hat{d}^2, T_n = \hat{d}^2$
u_i at each layer is incremented and the search procedure continues as mentioned.

3.4 High Level Simulation of the Sphere Decoding Algorithm

Before actually carrying out the implementation of the sphere decoding algorithm in the next section, which is the main concern of our thesis, it was felt necessary to visualize the functionality and working of the sphere decoder. Therefore the whole algorithm, including both pre-processing and decoding parts is initially developed in Matlab for simulating at behavioral level. The complete system is brief below:

- Generation of Lattice generator matrix based on normally distributed random numbers generated using MATLAB function "randn"
- Generating the upper triangular matrix by Cholesky decomposing of the gram matrix.
- After the input information to the decoder is ready, sphere decoding algorithm which finds the closest lattice point is simulated using Matlab. Its functionality is verified by comparing the obtained lattice point with the transmitted signal constellation vector.

The functionality of the decoder is verified at high level of abstraction and behavior of the decoder design is simulated using Matlab. Thus preliminary information of outputs is obtained. After ensuring the functionality of the decoder design, the corresponding hardware architecture is planned.

3.5 Decoder Architecture Scheduling

The hardware architectural model of Sphere decoder is planned in accordance with the simulated version. Each of the operations like calculating the bounds, calculating variables needed to update the bounds, spanning of index at each level and finding the Euclidean distance of a point from the received point are dealt in separate blocks. Different components are designed for specific set of operations at each block. Each of these blocks are designed in VHDL and tested for their functioning with the help of stand alone test benches and different sets of data. Digital circuit designs are invariable faced with the need to design circuits that perform specific sequence of operations, for example controllers used to control the operation of other circuits [Smith 1997]. Thus the decoder controller is designed for the hardware architecture of sphere decoder. The flowchart of the decoder controller of the original sphere decoding algorithm divided into states is shown in Figure 3.2. FSMs are proven to be a very efficient means of modeling sequencer circuits. By modeling FSMs in a HDL for use with synthesis tools, focus could be on modeling the desired sequences of operations without being overly concerned with circuit implementation. In Figure 3.4 the state diagram of the decoder controller is given. Sequences of operations which are almost independent of each other are combined into one single state. While state division, care is taken in regard of processing time needed at each state to maintain balance at the end of simulation of the algorithm.

Here, in our case, the six steps of the sphere decoder are modeled to four states FSM. This is because sequence of operations at some steps which do not really need separate states are combined with others and modeled into a single state. The Step 1, Step 2 and Step 3 are combined and modeled as State A. Step 5 as State B. Step 3 is combined with Step 4 and modeled into State C. Step 6 as State D. Since Step 3 involves simple index increment it need not be a separate state. It could be a part of State A or State C based on the requirement. If index has to be incremented immediately after it is assigned with lower bound, then it is part of State A. If only spanning of the interval with existing bounds, then it is part of State C.

3.5.1 Data Flow of the Algorithm

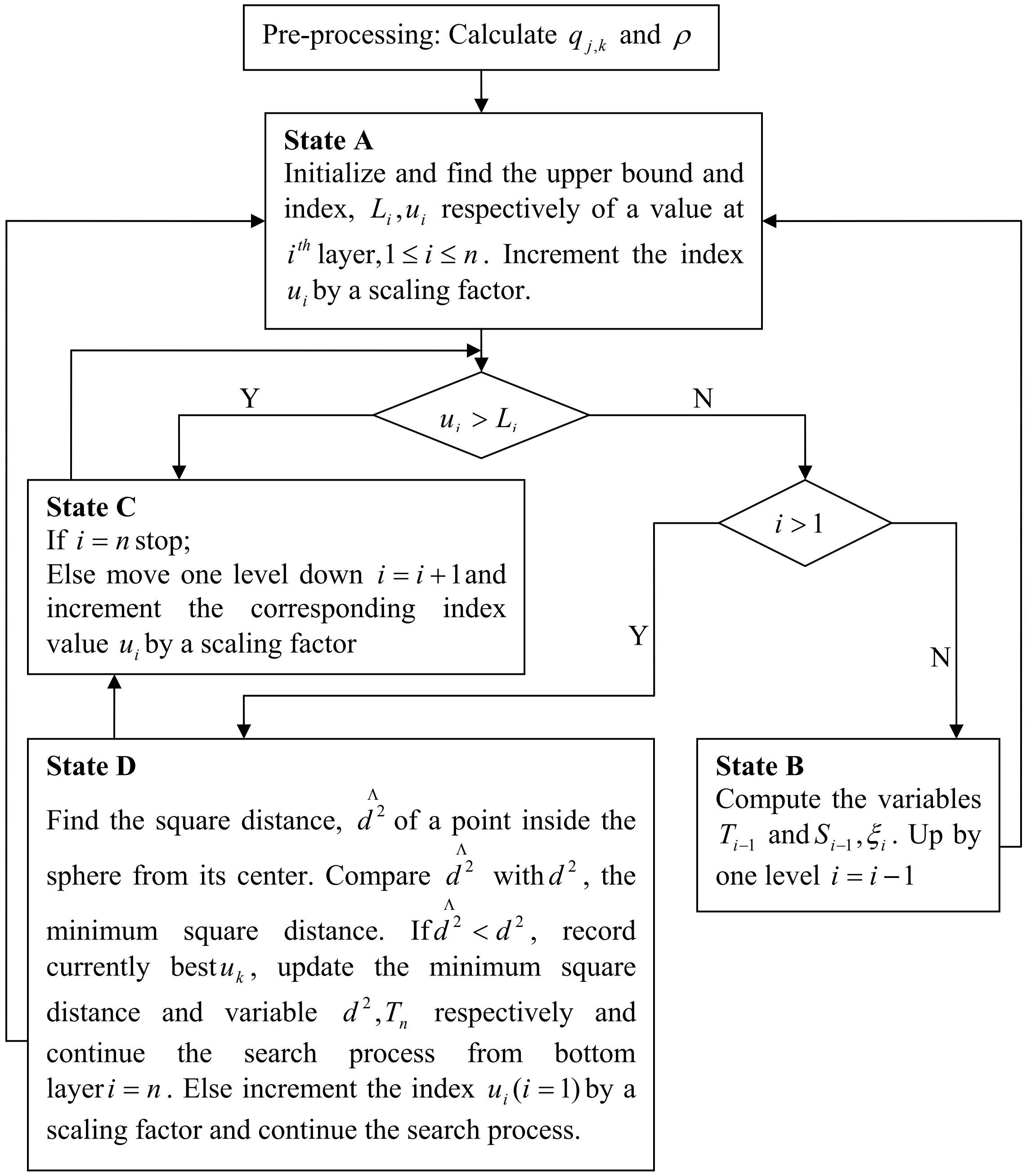

Pre-processing: Calculate $q_{j,k}$ and ρ
State A
Initialize and find the upper bound and index, L_i, u_i respectively of a value at i^{th} layer, $1 \le i \le n$. Increment the index u_i by a scaling factor.
$u_i > L_i$
Y
N
$i > 1$
Y
N
State C
If $i = n$ stop;
Else move one level down $i = i + 1$ and increment the corresponding index value u_i by a scaling factor
State D
Find the square distance, $\hat{d}^2$ of a point inside the sphere from its center. Compare $\hat{d}^2$ with d^2, the minimum square distance. If $\hat{d}^2 < d^2$, record currently best u_k, update the minimum square distance and variable d^2, T_n respectively and continue the search process from bottom layer $i = n$. Else increment the index $u_i (i = 1)$ by a scaling factor and continue the search process.
State B
Compute the variables T_{i-1} and S_{i-1}, ξ_i. Up by one level $i = i - 1$

Figure 3.2: Flowchart of a Sphere decoding algorithm showing states

As seen in Figure 3.2, the computations at each of the four states in the recursive lattice decoding algorithm are discussed in detail here. Along with the states and state transitions, the components enabled at each state are also discussed in detail. As we said earlier the computations are divided into four components.

In State A, it finds the upper and lower bounds of an integer component value at each layer. The variable L_i is assigned an upper bound and the index u_i is initially set at lower bound. Separate hardware component is designed for computing the square root. The decoder controller when in State A, enables the all the functional blocks designed to compute the above variables.

In State B, it computes the variables T_i, S_i and move one layer up. These variables are used to recursively update the lower and upper bounds at that layer. A functional block to compute the above variables in enabled at this state by the decoder controller. In addition to that the functional blocks active in previous state are disabled by the decoder controller.

In State C, check the layer at which search procedure is currently present. If it is the bottom most layer, terminate the search procedure and declare the last saved u as the closest lattice point. If the search procedure is at layers other than the bottom most layer move one layer down and increment the index value u_i at that layer by the scaling factor. At this state, the spanning of the interval at each layer, i.e., incrementing u_i is performed by the enabled functional block. All other details are taken care by the decoder controller.

In State D, the $\hat{d}^2$, the square distance of $\hat{u}$ the lattice point present inside the sphere from center of the sphere or the received point is computed and is compared with the minimum square distance d^2. Based on this, decision about the next state is made by the decoder controller.

At each state after obtaining the output from the blocks the decoder controller makes the decision about the next state in the current state. Decoder controller is designed in such a way that it disables the active functional blocks of previous state in addition to enabling the functional blocks of current state in the first clock cycle of current state itself. Thus when all the required conditions are met and all the sequence of operations are completed the results are output. The functioning of the decoder controller and all its components is tested using a test bench.

The pin diagram of the decoder controller for the original sphere decoding algorithm and its functionality is shown in is shown in Figure 3.3 and Table 3.1.

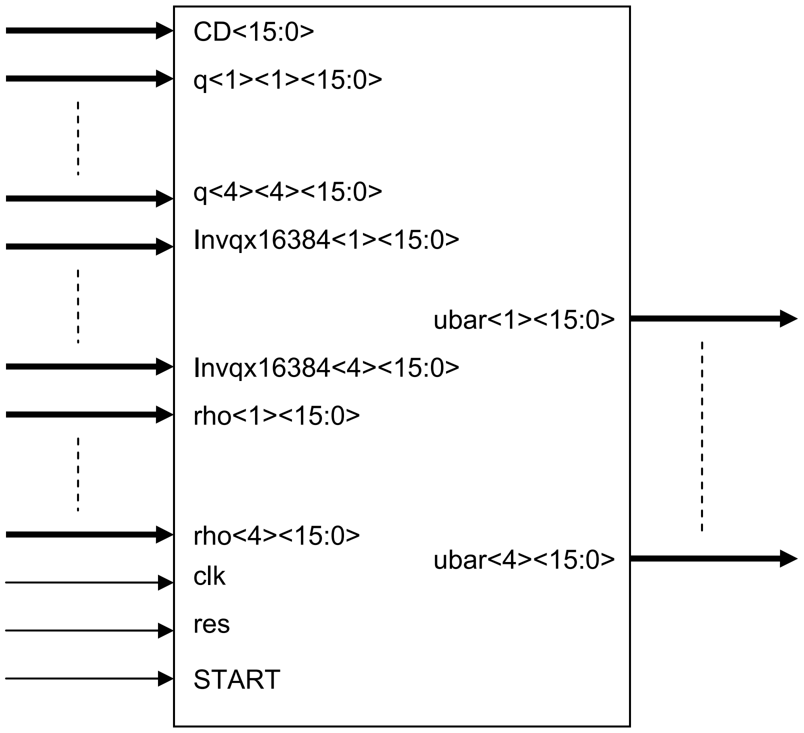


Figure 3.3: Input and Output pins for original sphere decoder

Table 3.1: Pin descriptions for the decoder controller of the original sphere decoding algorithm

Pin	Width	Type	Description
CD	16	Input	square radius of the sphere
q(1,1) - q(4,4)	16	Input	elements of Cholesky factor matrix
Invqx16384(1) – invqx16384(4	16	Input	Inverse of diagonal elements of the Cholesky factor matrix
rho(1) - rho(4)	16	Input	coordinates of received point vector with respect to lattice
clk)	1	Input	clock signal
res	1	Input	reset signal
START	1	Input	control signal to initialize the current state
ubar(1) - ubar(4)	16	Output	coordinates of the closest lattice point being searched

3.5.2 FSM Design

A finite state machine (FSM) of a decoder controller is designed to control and organize the sphere decoding algorithm and it synchronizes the operations between functional blocks. The five parameters u_i, L_i, the index and upper bound respectively at the current investigated layer of the lattice, the layer $i(1 \leq i \leq n)$, the square distance of the lattice vector inside the sphere from the received vector $\hat{d}^2$ and the minimum square distance d^2 determine the state transitions as shown in the Figure 3.4 below.

If the search procedure is in State A then after computing the index u_i and upper bound L_i, it checks for the conditions if the index is within range of the upper bound or equal to upper bound and the current layer is not the top layer then the control goes to State B. At State B, the variables needed to update the index and upper bound at State A are computed. Every time after State B control goes back to State A and continues to carry out the operations at this state. Again when in State A, it looks for the condition if the index is within the range of upper bound or equal to it and the current layer is the top layer then control moves to State D from State A. And if index exceeds the upper bound at any layer then the control moves to State C from State A. When the decoder controller is in State D, it computes square distance $\hat{d}^2$ and compares it with the minimum square distance d^2, if it is less then control goes to State A from State D and whole search procedure repeats once again. And if $\hat{d}^2$ is greater than or equal to the value of d^2 then controller moves from State D to State C. At State C the index value is incremented and the conditions are checked. The state transition from State C to other states is same as it was from State A to other states.

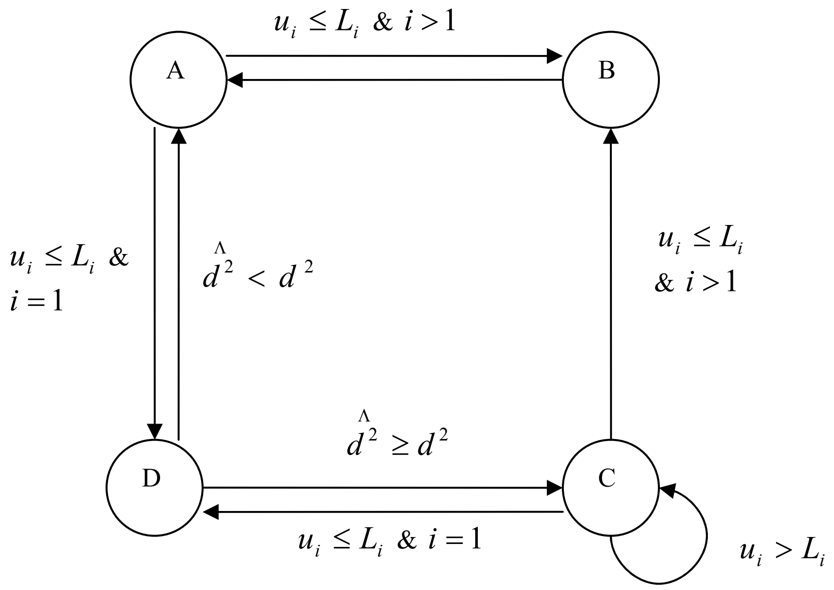


Figure 3.4: The FSM diagram of Sphere decoding algorithm

3.6 Simulation Results

The decoder core is designed in VHDL at register transfer level (RTL). Mentor Graphics' Modelsim SE 5.8 tool is used to create, compile and simulate the VHDL source code of the decoder core. A design library named work is automatically created in the project directory upon creating the new project and all the necessary design files and test bench are held together in the project directory. The VHDL source code is compiled to test its syntax. Successfully compiled source code is simulated using different sets of data. At the simulation step, initially the design is loaded successfully if no errors are reported. View the signals of the design and add the necessary signals to the waveform window. Run the wave until output results of the whole design are obtained. Processing time taken by each state of the decoder controller individually can be acquired from the wave. Table 3.1 shows the processing time of each state of FSM of the Sphere decoder after successful VHDL simulation.

Table 3.1: Simulation Times of each state in original algorithm

State	**A**	**B**	**C**	**D**
Simulation Time in clock cycles	37	7	2	7

The determination of lower and upper bounds of an integer component value at a particular layer involves a 32-bit square root computation. To compute the square-root, here we made use of non-restoring algorithm explained in Section 3.6.1

3.6.1 Non-Restoring Square Root Algorithm

In this algorithm [Piromsopa 2001], the radicand is a 32-bit unsigned number. The square root is a 16-bit unsigned number. R is the remainder $(R = D - (Q)^2)$ which is a 17-bit integer.

Since this is a redundant representation for a square root, exact bit can be obtained in each iteration.

```
Let
D be 32-bit unsigned integer.
Q be 16-bit unsigned integer (Result)
R be 17-bit integer $(R = D - Q^2)$
Algorithm
$Q = 0;$
$R = 0;$
For i = 15 to 0 do
    If $(R \geq 0)$
        $R = (R << 2) or (D >> (i + i) and 3);$
        $R = R - ((Q <<) or 1);$
    Else
        $R = (R << 2) or (D >> (i + i) and 3);$
        $R = R - ((Q <<) or 3);$
    End if
    If $(R \geq 0)$ then
        $Q = (Q << 1) or 1;$
    Else
        $Q = (Q << 1) or 0;$
    End if
```

The above non-restoring algorithm for calculating the square-root of a number is explained clearly by considering an example. Here in the example we consider D as an 8-bit radicand equal to value $140 (10001100_2)$. The 4-bit solution Q should be $11 (1011_2)$ and remainder R should be equal to $19 (10011_2)$.

Set Q = 0000 and R = 000000

$i = 3,$

$R \geq 0, R = 000010 - 000001 = 000001$

$R \geq 0, Q = 0001$

$i = 2,$

$R \geq 0, R = 000100 - 000101 = 011111$

$R < 0, Q = 0010$

$i = 1,$

$R < 0, R = 011111 + 001011 = 001010$

$R \geq 0, Q = 0101$

$i = 0,$

$R \geq 0, R = 101000 - 010101 = 010011$

$R \geq 0, Q = 1011$

To correctly determine value of R, one more extra bit is added (Consider as sign bit). Thus the result Q is obtained.

From the simulation results of the sphere decoder core it is seen that sequence of operations at State A take 37 clock cycles. Out of this, 32 clock cycles are needed for a square root computation. The sequences of operations at other states take less than 10 clock cycles. Comparing with the other states, processing time of State A is remarkably high. Due to this imbalance and very high processing time, the throughput of the system is affected noticeably. This imbalance has to be removed for efficient and high throughput implementations. This eventually results in an un-efficient hardware implementation of the sphere decoding algorithm.

An improved form of the algorithm is suggested with modifications in the sequences of operations of each functional block. These modifications are such that the square root computation is no longer necessary. They can be explained in detail in the next chapter.

4 Improved Sphere Decoding Algorithm

The improved sphere decoding algorithm is derived with modifications applied to the sequences of operations at each state of the original algorithm in this chapter. The dataflow of the improved algorithm is discussed. A table showing hardware processing time needed by each state is given. Data dependency of the algorithm is also analyzed.

4.1 Improved Sphere Decoding Algorithm

An improved sphere decoding algorithm is proposed. The need for the improved algorithm arises from the simulation results of the original algorithm. As we have seen, State A of the original algorithm requires 37 clock cycles for completion, out of which 32 clock cycles are taken by square root itself. On the other hand the processing time required by each of the remaining states is limited to very few clock cycles (Refer Table 3.1). The sequences of operations at other states have to wait for the completion of State A if they are depending on the results of State A. This time delay can be reduced if the square root computation is avoided. Therefore we suggest some modifications to the original algorithm such that square root is avoided in its sequences of operations and as a result emerges an improved sphere decoding algorithm. The derivation of modifications is given in Section 4.1.1. The sequence of operations at State A of the improved algorithm use simple adders and multipliers to compute the upper bound L_i and the index u_i. Since there is no square root computation involved, a hardware component to compute square root is no longer needed. In the improved algorithm, modifications are present at the sequences of operations, whereas the state division and the state transition decisions depending on the outputs obtained from the functional blocks at each state remains the same.

4.1.1 Derivation of Modifications

As we know the minimum squared Euclidean distance between any two points of the lattice equals the minimum of quadratic from $Q(x)$ for any $x \in Z^n$ [Viterbo 1993]. Applying this to the sphere decoder, squared Euclidean distance between any point inside the sphere and received point must be less than or equal to the square radius of the sphere.

$$\|w\|^2 = Q(\xi) \leq C \qquad \text{Equation (4-1)}$$

$$\sum_{i=1}^{n} q_{ii}(\xi_i + \sum_{j=i+1}^{n} q_{ij}\xi_j)^2 \leq C \qquad \text{Equation (4-2)}$$

Expanding this, we get

$$q_{11}(\xi_1 + q_{12}\xi_2 + ...q_{1n}\xi_n)^2 + q_{22}(\xi_2 + q_{23}\xi_3 + ...q_{2n}\xi_n)^2 + ... +$$

$$q_{(n-1)(n-1)}(\xi_{n-1} + q_{(n-1)n}\xi_n)^2 + q_{nn}\xi_n^{\ 2} \leq C \qquad \text{Equation (4-3)}$$

We know that $\xi_i = \rho_i - u_i$ Equation (4-4)

Substituting equation (4-4) in (4-3), we get

$$q_{11}(\rho_1 - u_1 + q_{12}(\rho_2 - u_2) + ... + q_{1n}(\rho_n - u_n))^2 + q_{22}(\rho_2 - u_2 + q_{23}(\rho_3 - u_3) + q_{24}(\rho_4 - u_4))^2 + ... +$$
$$q_{nn}(\rho_n - u_n)^2 \leq C \qquad \text{Equation (4-5)}$$

Equation (4-5) cannot be solved because of presence of *n* unknowns. Therefore we need to split the expression and solve it. Due to the upper triangular form of Cholesky factor matrix, equation (4-5) represents a set of conditions.

at *i* = *n*, $q_{nn}(\rho_n - u_n)^2 \leq C$ Equation (4-6)

at *i* = *n-1*, $q_{n-1,n-1}(\rho_{n-1} - u_{n-1} + q_{n-1,n}\xi_n)^2 + q_{nn}(\xi_n)^2 \leq C$ Equation (4-7)

and so on.

Equation (4-6) can be solved easily because of only one unknown i.e., u_n. Considering the above conditions in the order from n to 1 i.e., starting at the bottom layer and carrying on the backward substitution, we obtain the admissible values of each symbol u_i for known values of $u_{i+1},\ldots,u_n$.

The range of the index u_i as found in the original algorithm is given as

$$\left\lceil -\sqrt{\frac{1}{q_{ii}}\left(C-\sum_{l=i+1}^{n}q_{ll}(\xi_l+\sum_{j=l+1}^{n}q_{lj}\xi_j)^2\right)}+\rho_i+\sum_{j=i+1}^{n}q_{ij}\xi_j\right\rceil \le u_i \le$$

Equation (4-8) $$\left\lfloor \sqrt{\frac{1}{q_{ii}}\left(C-\sum_{l=i+1}^{n}q_{ll}(\xi_l+\sum_{j=l+1}^{n}q_{lj}\xi_j)^2\right)}+\rho_i+\sum_{j=i+1}^{n}q_{ij}\xi_j\right\rfloor$$

In equation (4-8), the upper and lower bounds of index u_i are found by using a square root computation. The main idea in the improved algorithm is to avoid square root

At i^{th} layer, equation (4-5) can be written as

$$q_{ii}(\rho_i-u_i+q_{i,i+1}(\rho_{i+1}-u_{i+1})+\ldots+q_{in}(\rho_n-u_n))^2+q_{i+1,i+1}(\rho_{i+1}-u_{i+1}+q_{i+1,i+2}(\rho_{i+2}-u_{i+2})+\ldots+$$

$$q_{nn}(\rho_n-u_n)^2\le C \qquad \text{Equation (4-9)}$$

Simplifying it further,

$$q_{ii}(\rho_i-u_i+\sum_{j=i+1}^{n}q_{ij}\xi_j)^2+\sum_{l=i+1}^{n}q_{ll}(\rho_l-u_l+\sum_{j=l+1}^{n}q_{lj}\xi_j)^2\le C \qquad \text{Equation (4-10)}$$

When the search procedure completes, index vector u should be the closest point to the transmitted signal. Because signal constellation is known at the receiver part, a new method of determining the search range of lattice index can be achieved by directly substituting each symbol from the signal constellation into equation (4-10). Here we assume the integer component value u_i as one among the signal constellation elements $x_k, k=1....n$ (For a 4-PAM signal, symbol set is ranging as {-3, -1, 1, 3}) then equation (4-10) can be written as

$$q_{ii}(\rho_i - x_k + \sum_{j=i+1}^{n} q_{ij}\xi_j)^2 + \sum_{l=i+1}^{n} q_{ll}(\rho_l - u_l + \sum_{j=l+1}^{n} q_{lj}\xi_j)^2 \le C$$ Equation (4-11)

If we redefine variable T_l as

$$T_l = q_{ll}(S_l - u_l)^2$$ Equation (4-12)

and variable S_i holds the same definition as in the original algorithm described in equation (3-9)

$$S_i = \rho_i + \sum_{j=i+1}^{n} q_{ij}\xi_j$$ Equation (4-13)

Finally by substituting equation (4-12), (4-13) in equation (4-11), we get the expression

$$q_{ii}(S_i - x_k)^2 + \sum_{l=i+1}^{n} T_l \le C$$ Equation (4-14)

$$\therefore \; p_k = q_{ii}(S_i - x_k)^2 + \sum_{l=i+1}^{n} T_l \le C \;\forall \text{ values of } k = 1....n$$ Equation (4-15)

The upper bound, $L_i = \max(x_k) \; \forall \;\; p_k \le C$

The index, $u_i = x_r - 1$ for $p_r = \min(p_k) \le C \; \forall$ values of $k = 1....n$

If vector p is empty, then the upper bound L_i and index u_i are assigned with maximum and minimum values of signal constellation.

Considering an example to explain this in detail, at SNR = 20 dB and generator matrix M is given as

$$M = \begin{bmatrix} 0.2944 & -0.6918 & -0.4410 & 0.8156 \\ -1.3362 & 0.8580 & 0.5711 & 0.7119 \\ 0.7143 & 1.2540 & -0.3999 & 1.2902 \\ 1.6236 & -1.5937 & 0.6900 & 0.6686 \end{bmatrix}$$

Then the received signal obtained after scaling and rounding is equal to $[385 \quad 119 \quad -130 \quad -376]$ when the transmitted signal constellation is equal to $\begin{bmatrix}384 & 128 & -128 & -384\end{bmatrix}$. Assuming the appropriate choice of squared sphere radius, $C = 512$ (after scaling and rounding). In such a case, the sequence of operations to find the index u_i, and upper bound L_i go as follows.

at $i = 4, p = [2714 \quad 1193 \quad 289 \quad 0]$

The upper bound, $L_i = \max(x_k)$ = max (-128, -384) = -128

The index, $u_i = x_r - 1$ for $p_r = \min(p_k) \leq C$

$$p_r = 0$$

$$\therefore u_i = -512$$

This avoids square root computation while finding upper and lower bounds. And thus the index u_i takes the value within the range of signal constellation. The main advantage achieved from this improved sphere decoding algorithm is the significant reduction in the processing time of State A when the algorithm is prototyped on hardware. The flowchart of the improved algorithm is given in Figure 4.1.

4.1.2 Flow-Chart

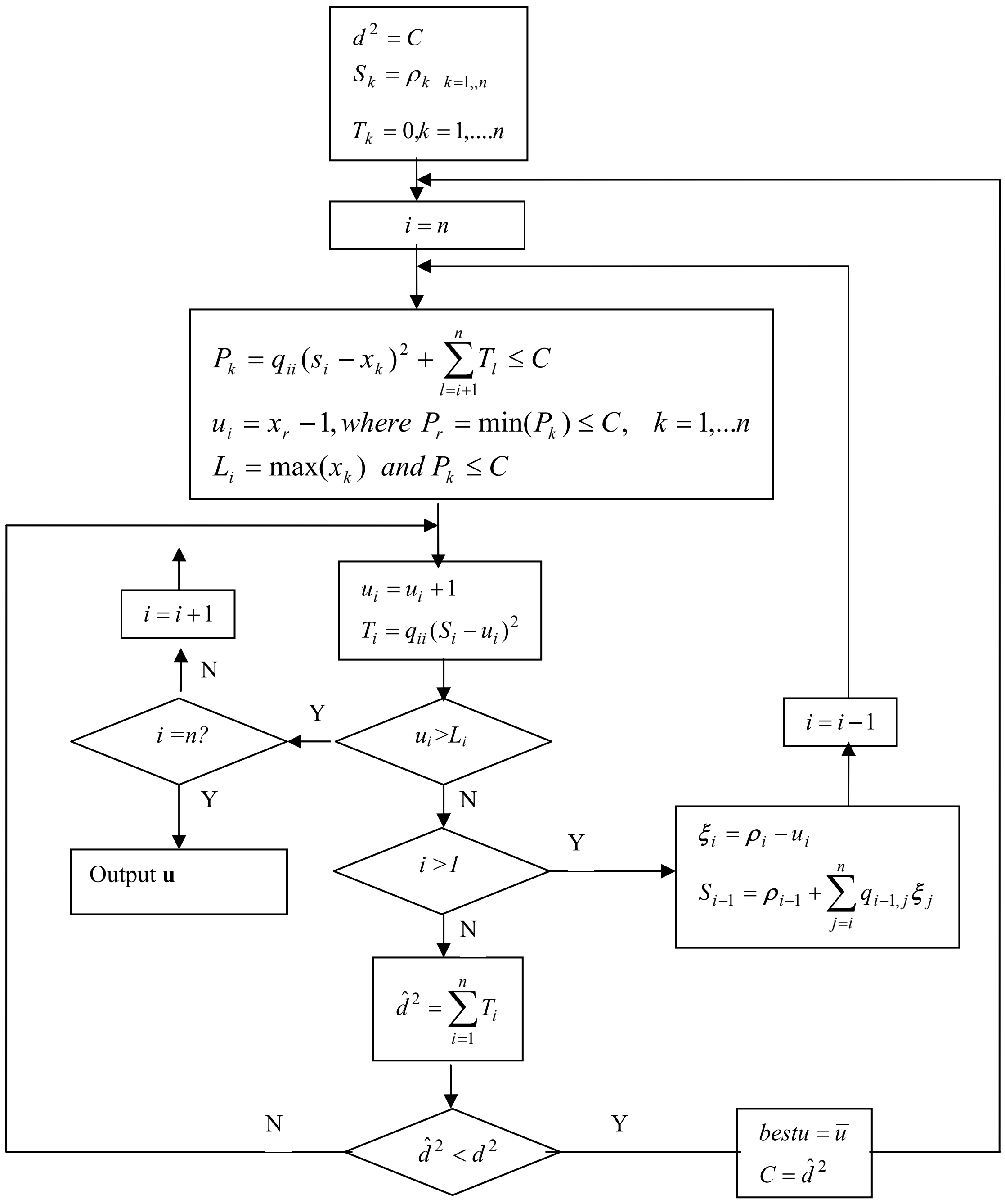

$d^2 = C$
$S_k = \rho_k \quad k=1,,n$
$T_k = 0, k = 1,....n$
$i = n$
$P_k = q_{ii}(s_i - x_k)^2 + \sum_{l=i+1}^{n} T_l \le C$
$u_i = x_r - 1, where\ P_r = \min(P_k) \le C, \quad k = 1,...n$
$L_i = \max(x_k)\ and\ P_k \le C$
$u_i = u_i + 1$
$T_i = q_{ii}(S_i - u_i)^2$
$i = i + 1$
N
$i = n?$
Y
$u_i > L_i$
Y
N
Output **u**
$i > 1$
Y
N
$\xi_i = \rho_i - u_i$
$S_{i-1} = \rho_{i-1} + \sum_{j=i}^{n} q_{i-1,j}\xi_j$
$i = i - 1$
$\hat{d}^2 = \sum_{i=1}^{n} T_i$
N
$\hat{d}^2 < d^2$
Y
$bestu = \overline{u}$
$C = \hat{d}^2$

Figure 4.1: Flow chart of improved algorithm

4.2 Decoding Procedure

The original sphere decoding algorithm performs step-by-step procedure as follows,

The inputs are C, ρ, x, Q and output is $\hat{u}$

Step 1. (Initialization)

Set $i = n, T_k = 0, d^2 = C$ (current sphere square radius) and

$$S_k = \rho_k, k = 1.........n$$

Step 2. (Bounds on index u_i)

Compute the parameter p_k such that the upper bound and index values are found.

$$\text{Thus } P_k = q_{ii}(s_i - x_k)^2 + \sum_{j=i+1}^{n} T_l \leq C \text{ , } (k = 1,.....n)$$

$$L_i = \max(x_k), where \quad P_k \leq C$$

$$u_i = x_r - 1, where\ P_r = \min(P_k) \leq C, \quad (k = 1,.....n)$$

Here when signal constellation vector is known, the upper bound and index can be computed.

Step 3. (Natural spanning of the interval)

Increment the index u_i by one step, i.e., $u_i = u_i + 1$ and compute the variable T_i at each layer *i*. Thus $T_i = q_{ii}(S_i - u_i)^2$

If $u_i \leq L_i$ and $i > 1$, i.e., the index is within the range and layer is not the top layer then go to Step 5, else if $u_i \leq L_i$ and $i = 1$, i.e., the index of the top layer is within the bound then go to Step 6, else if $u_i > L_i$ go to Step 4.

Step 4. (Increase *i*: move one level down)

If $i = n$ terminate, i.e., the end of the search procedure is reached and closest lattice point to received point is found, else set $i = i + 1$, i.e., the search procedure goes one level down in the hierarchy, and go to Step 3.

Step 5. (Decrease *i*: move one level up)

Let $\xi_i = \rho_i - u_i$, $S_{i-1} = \rho_{i-1} + \sum_{l=i}^{n} q_{i-1,l}\xi_l$

$i = i - 1$ and go to Step 2.

The variables needed to recursively update the lower and upper bounds are computed at this step and the search procedure goes one layer up in the hierarchy to re-compute the upper bound and index u_i.

Step 6. (A valid point is found)

Compute $\hat{d}^2 = \sum_{i=1}^{n} T_i$, the square distance of the vector found from the center. Then compare this value to the minimum square distance d^2 i.e., If $\hat{d}^2 < d^2$ then save the lattice point, $\hat{u}_k = u_k, k = 1....n$ and reduce the search area by assigning the minimum square distance value d^2 with $\hat{d}^2$ and again set $i = n$. Thus $d^2 = \hat{d}^2$

Then go to Step 2 repeat the whole process once again. Else go to Step 3, where the index value u_i at each layer is incremented and the search procedure continues as mentioned.

4.3 High Level Description of the improved Sphere decoder

For improved sphere decoding algorithm, we follow the same order of steps as in original form. The functionality and working of the improved form of the sphere decoding algorithm is visualized and tested using Matlab simulation. For this the complete algorithm including the preprocessing and decoding parts is initially developed in Matlab. For detail description, follow Section 3.4.

4.4 Decode Architecture Scheduling

The hardware architectural model of the improved form of sphere decoder is designed in accordance with the simulated version. Sequences of operations like finding the upper bound and index value, calculating variables needed in computing the index value, spanning of index and partial Euclidean distance variable, and finding Euclidean distance of a currently investigating point from the received point are individually dealt in separate functional blocks. Different hardware components are designed for each set of functional block operations. Each of these blocks are designed remotely in VHDL and tested for their functioning with the help of stand alone test benches and different sets of data. The decoder controller is designed for the hardware architecture of the improved sphere decoding algorithm. The flowchart showing the states and sequences of operations at each state for the improved algorithm are shown in Figure 4.2. Details about the state division are same as for the original algorithm (refer Section 3.5.2). Therefore the state diagram for the FSM decoder controller of improved sphere decoding algorithm is same as Figure 3.4.

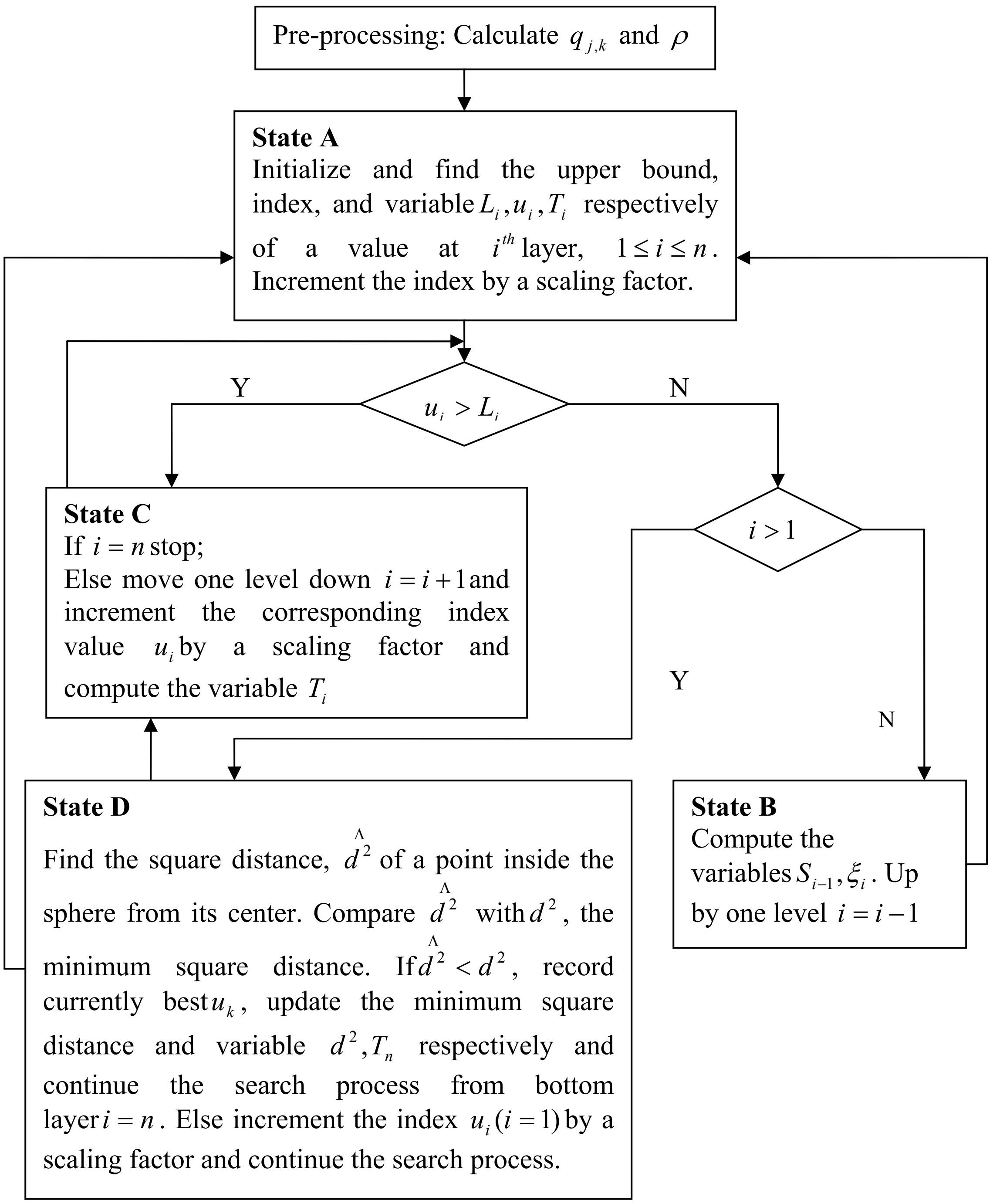

Pre-processing: Calculate $q_{j,k}$ and ρ
State A
Initialize and find the upper bound, index, and variable L_i, u_i, T_i respectively of a value at i^{th} layer, $1 \le i \le n$. Increment the index by a scaling factor.
$u_i > L_i$
Y
N
$i > 1$
Y
N
State C
If $i = n$ stop;
Else move one level down $i = i + 1$ and increment the corresponding index value u_i by a scaling factor and compute the variable T_i
State B
Compute the variables S_{i-1}, ξ_i. Up by one level $i = i - 1$
State D
Find the square distance, $\hat{d}^2$ of a point inside the sphere from its center. Compare $\hat{d}^2$ with d^2, the minimum square distance. If $\hat{d}^2 < d^2$, record currently best u_k, update the minimum square distance and variable d^2, T_n respectively and continue the search process from bottom layer $i = n$. Else increment the index $u_i (i = 1)$ by a scaling factor and continue the search process.

Figure 4.2: Flow chart of an improved algorithm showing states

Similar to original algorithm, a state machine with four states is developed for the improved sphere decoding algorithm. The Figure 4.2 depicts the states, state transitions and sequences of operations at each state. Operations at each state are nothing but the operations of functional block enabled at that state. For each functional block, an entity - architecture model is developed in VHDL. Each of these hardware components is tested for its functionality using corresponding test benches.

In State A, it finds the upper bound L_i, of an integer component value, index u_i, and partial Euclidean distance variable, T_i at each layer. Decoder controller enables the functional block designed to compute above variables.

Similar procedure is followed at all other states. Decoder controller enables the functional blocks needed to compute variables at that state and disables the previous state components. After all possible state transitions the decoder controller finds the closest lattice point to the received point. The whole decoder controller system is designed in VHDL and hardware functionality is tested using a test bench at RTL level of abstraction.

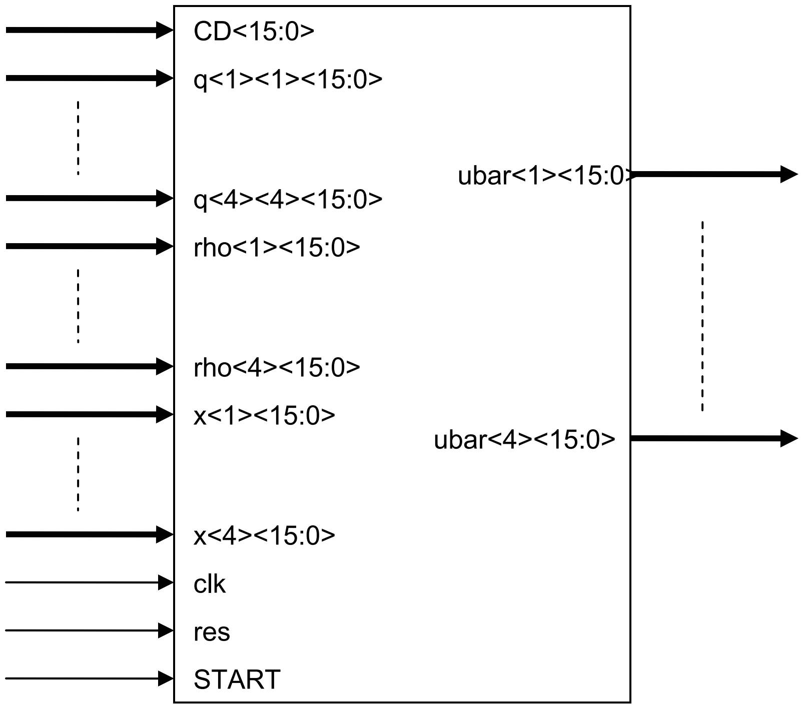


Figure 4.3: Input and Output pins for improved sphere decoder

The pin diagram of the decoder controller of the improved sphere decoding algorithm and its functionality is shown in Figure 4.3 and Table 4.1.

Table 4.1: Pin descriptions for the decoder controller of the improved sphere decoding algorithm

Pin	Width	Type	Description
CD	16	Input	square radius of the sphere
q(1,1) - q(4,4)	16	Input	elements of Cholesky factor matrix
rho(1) - rho(4)	16	Input	coordinates of received point vector with respect to lattice
x(1) - x(4)	16	Input	coordinates of transmitted signal constellation vector
clk	1	Input	clock signal
res	1	Input	reset signal
START	1	Input	control signal to initialize the current state
ubar(1) - ubar(4)	16	Output	coordinates of the closest lattice point being searched

4.5 Hardware-Software Scenario

The complete file structure and planning, of both the simulation and hardware development processes are shown in Figure 4.4. The inputs are generated randomly. The receiver output obtained is noise corrupted. These inputs are preprocessed. Using preprocessed data and necessary inputs, the uncoded receiver signal is decoded by the sphere decoding algorithm (.m file of original or improved version). The decoded outputs and errors are recorded. After the decoder is implemented in hardware, its functionality will be verified with help of same input used for checking the decoding algorithm functionality. In the software simulation i.e., in Matlab, algorithm is tested with 10000 simulations or sets of data at a time, whereas in hardware i.e., in VHDL, algorithm is tested for single data or simulation at a time.

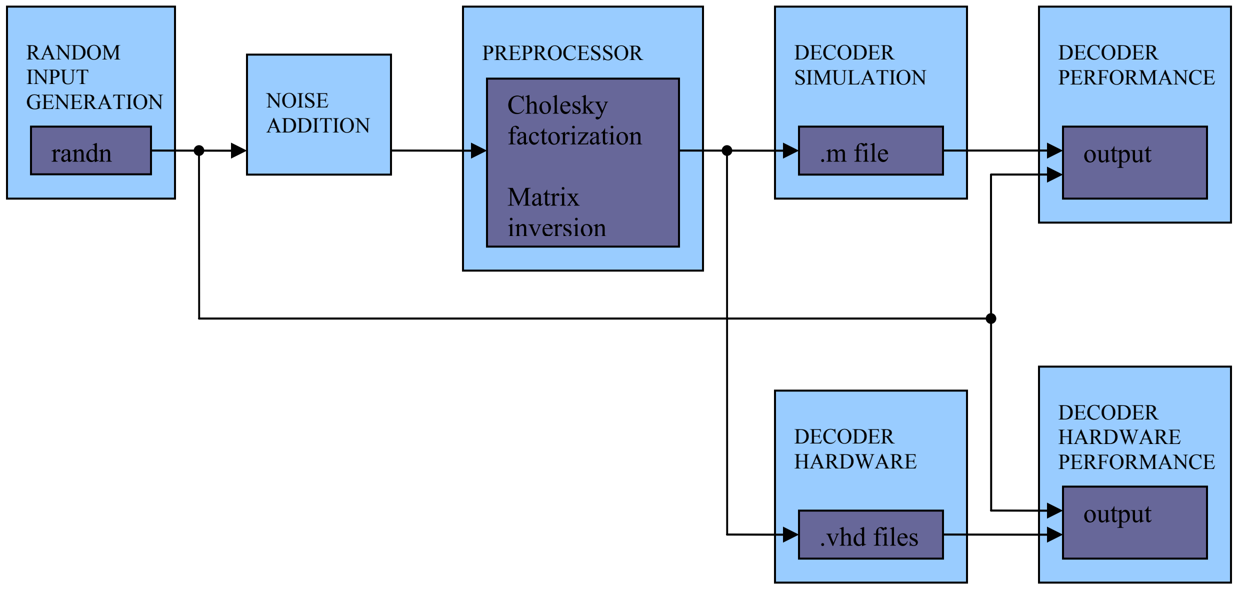


Figure 4.4: Overview of the complete system

4.6 Simulation Results

The decoder core of the improved sphere decoding algorithm is designed in VHDL at register transfer level (RTL). Mentor Graphics' Modelsim SE 5.8 tool is used to create, compile and simulate the VHDL source code of the decoder core. A design library named work is automatically created in the project directory upon cresting a new project and all the necessary design files and test bench are held together in the project directory. The VHDL source code is compiled for its correct syntax and is then executed. Upon successful loading of design, signals are added to the wave and allowed to run until the results are obtained. The waveform gives the details like the processing time of each state, number of time each state is visited and order of states one following the other. The processing time of each state in improved algorithm approximately are shown in Table 4.2.

Table 4.2: Simulation Times of each state in improved algorithm

State	A	B	C	D
Simulation Time in clock cycles	7	7	7	3

The simulation results of the improved sphere decoding algorithm show significant improvement compared to the original algorithm. At State A, number of clock cycles required falls to 7 from 37. This improvement is due to discarding square root in sequence of operations at State A. With approximately equivalent clock cycles at each state, the parallel-pipeline implementation could speed up the search procedure. None of the states need to wait for long time to start or make decisions about next state as it happened in the original algorithm. i.e., when two states are implemented in parallel, they start simultaneously and come to an end approximately at the same time. No latencies are inserted into the system. Thus the improved algorithm is favorable for parallel design implementation.

4.7 Data Dependency

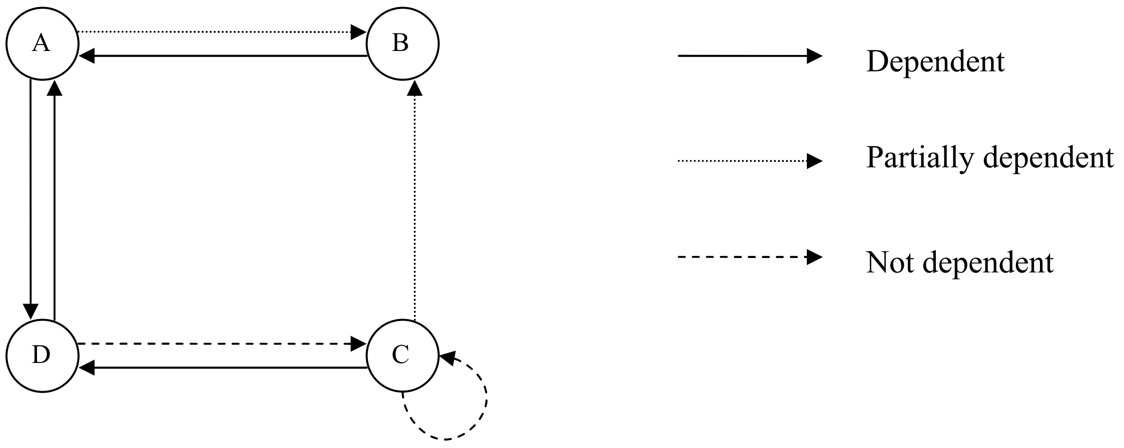


Figure 4.5: Dependency graph of the Sphere decoding algorithm

Unlike other decoding algorithms such as Viterbi and Turbo decoding algorithms, this sphere decoding algorithm has high data dependency between states as depicted in Figure 4.5. State A is flow-dependent on states B and D if the search procedure switches to A from B and D because the parameters S_i and $\hat{d}^2$ calculated in states B and D respectively are used in A when the upper and lower bounds of the value are determined. This means that either state B or state D cannot be implemented in parallel to state A. Similarly states B and D are flow dependent on A, C if the search procedure switches to B or D from A or C because the parameter T_i and u_i, the integer component at i^{th} layer are used in some computations in states A and C. This concludes that states A or C cannot be implemented in parallel to states B or D. Looking at the possibilities of pipelining, it is seen that State A or C can be implemented in pipeline to State B. Considering the case of State A switching to State B, it can be observed that part of operations involved in calculation of variable S_i are independent of index u_i, the output of State A. This means state B is partially dependent on state A. Therefore, State B can begin before the completion of State A or before u_i is computed. Once index u_i is determined, State B continues with other operations.

Thus concept of pipeline evolves between State A and B. The case of State A pipelined to State B also supports partial parallelism or in other words, it can be stated as state B is partially dependent on state A. Therefore, when one state is partially dependent on the other, pipelining could be evolved between them. In case of search procedure switching from C to B, it is seen that computations in state B use the index u_i and not T_i of state C where T_i is computed later than u_i. This means state B can also start before state C ends. Similar is the case of A pipeline to B, it can be stated that State C pipeline to State B.

Both states A and C can neither be executed in parallel nor in pipeline to State D. This is because, squared distance $\hat{d}^2$ computed at State D requires variable vector T which is obtained at the end in both A and C.

Dependency from A to A is not investigated because it is not possible for state A to follow itself in this algorithm. Similar is the case with state B and D. But if we analyze the search procedure in detail, it can be found that state C is not data dependent on state D and itself because it does not use any of the parameters or values calculated during any of the states that could jump to state C.

Based on the data dependency analysis, the possibility of the parallelism and pipelining among the four states is found as follows.

C || C, D || C

A $\nparallel$ B, B $\nparallel$ A, A $\nparallel$ D, D $\nparallel$ A, C $\nparallel$ D, C $\nparallel$ B

A | B, C | B

B $\nmid$ A, A $\nmid$ D, D $\nmid$ A, C $\nmid$ D

Where, D || C means if current state is D and next state is C, these two states can be implemented at the same time, A $\nparallel$ B means if current state is A and next state is B, then these two states cannot be implemented in parallel, C | B means if current state is C and next state is B, then these two states are implemented in pipeline i.e., state B is started before the end of state C is reached, and B $\nmid$ A means if current state is B and next state is A, then these two sates cannot be implemented in pipeline.

5 FPGA Based Architecture Design

The next stage of work involved is the parallel-pipeline implementation of the improved sphere decoding algorithm and therefore designing an efficient architectural model for it. Hence, this chapter discusses in detail the parallel-pipeline architecture for improved lattice decoding algorithm. The design optimization techniques are also illustrated.

5.1 Lattice Decoder Architecture

The hardware architectural model for improved sphere decoding algorithm is shown in Figure 5.1. The decoder controller communicates with the functional blocks at each state. The data transfer and decision about next state are made at the decoder controller. Data buffer unit consists of array of registers to temporarily store data during the decoding process.

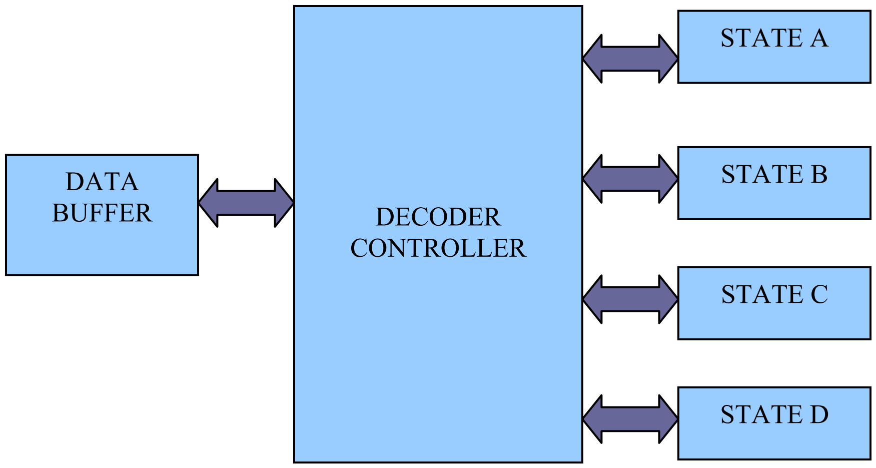


Figure 5.1: The hardware architecture of improved sphere decoding algorithm

The decoder controller is designed using FSM to organize the improved sphere decoding algorithm and to synchronize the operations of functional blocks. The state diagram of this FSM is same as given in Figure 3.3. As the data flow and state transition decisions are same in both original and improved form of the sphere decoding algorithms, the state diagram is similar. Differences between both the algorithmic models lie in the sequences of operations involved at each state. Thus the improved form of the sphere decoding algorithm reduces the individual complexity at each state which is beneficial to the entire decoder system model.

Based on the data dependency analysis, we designed a parallel-pipeline architectural model for improved sphere decoding algorithm. For the parallel architectural model, in addition to the existing four functional blocks, three duplicated functional blocks for sequences of operations at State C are created. This is because State C can be in parallel to another State C. In our case, system being a 4-transmit and 4-receive antenna system i.e., $n = 4$, maximum of four C states can be performed simultaneously in parallel to each other. For a general case with *m*-transmit and *n*-receive antennas, maximum *n* number of C states could be implemented in parallel. Thus, $n-1$ numbers of duplicate functional blocks for State C need to be created.

5.2 Parallel Structure

Based on the data dependency analysis in Chapter 3, a parallel structure is developed to implement the sphere decoding search procedure. Seven hardware modules are created in this structure, with one for each state and three duplicated modules for state C because four continuous C states could be implemented at the same time in parallel. When all four C states are implemented in parallel, it's the end of the algorithm i.e., the closest lattice point is found or no lattice point is reported. The hardware architectural model for parallel-pipeline implementation is shown in the Figure 5.2.

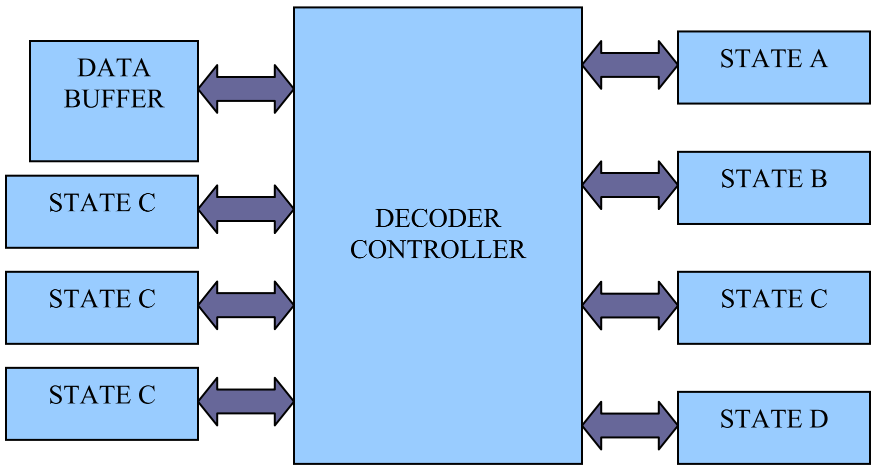


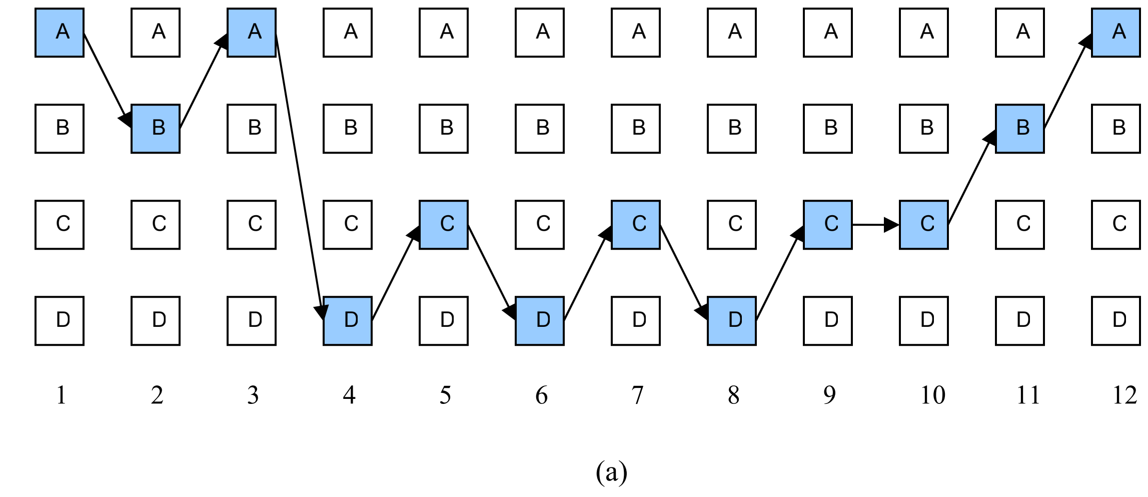
Figure 5.2: The hardware architecture of parallel-pipeline improved sphere decoding algorithm

These seven modules are executed simultaneously to speed up the search procedure as shown in Figure 5.3 (a) and (b) below. For an example of an improved sphere decoding algorithm, the sequence of states captured Eb/No = 6 dB is shown in Table 5.1, to demonstrate the parallel-pipeline implementation.

Table 5.1: Sequence of states for an example of improved algorithm at 6dB SNR

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	....
State	A	B	A	B	A	B	A	D	C	D	C	D	C	C	B	A	....

Figure 5.3 (a) and (b) give the pictorial description of the above sequences of states when implemented in sequential and parallel-pipeline. For making the explanation more simple and clear, the iterations from 5 - 16 are considered. The difference in sequential and parallel-pipeline implementations exists at these iterations.



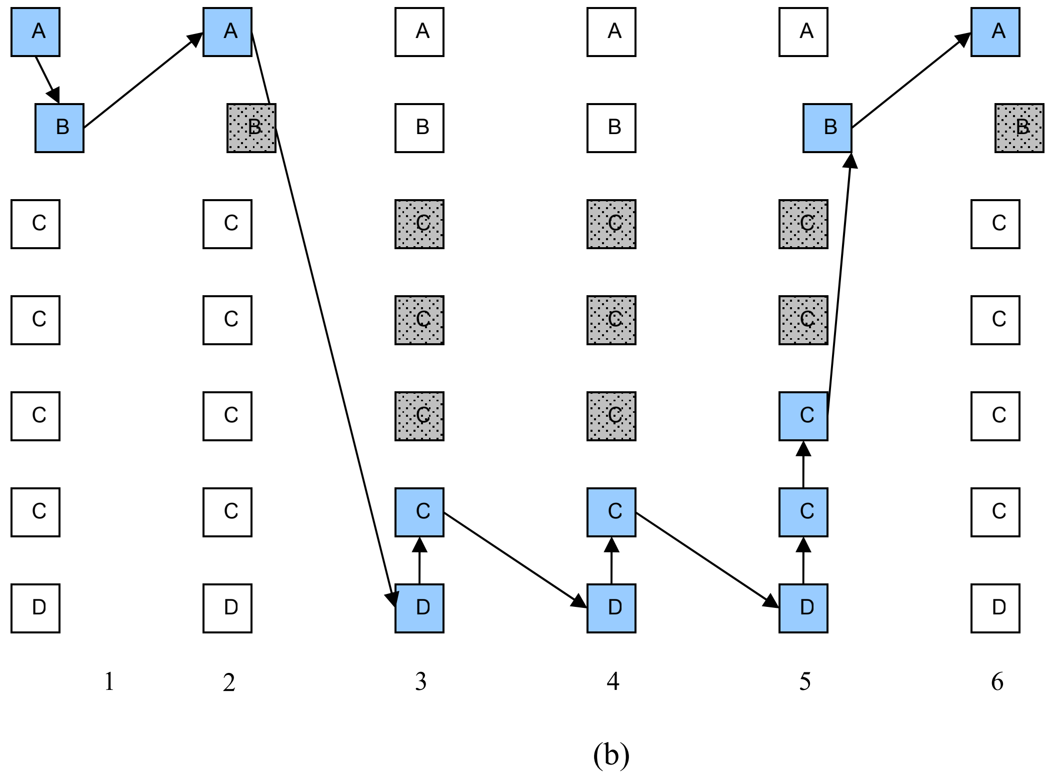


Figure 5.3: An example of improved sphere decoding algorithm (a) Sequential implementation (b) Parallel-Pipeline implementation

The shaded boxes represent the following conditions:

🟦	Module enabled and results accepted.
▨	Module enabled and results ignored.

In the example if the current state is A then module B is enabled with a time delay, making A be implemented in pipeline to B (refer Section 4.7). The results of B are accepted or ignored depending on the state transition conditions as shown in Figure 3.4. In Figure 5.3 (b) at first iteration, results of B are accepted as the state transition conditions lead to B as next state. Therefore the length of iteration is equal to the sum of processing time of module A and the extra time taken by the module B. In the next iteration although module B is enabled, the results are ignored as the state transition conditions lead to D as next state. Length of the iteration in this case is equal to the processing time of module A. In the case of pipelining when the results are accepted, the pipelining state has to be allowed to reach completion. Thus, more time is needed for iterations with pipeline and results being accepted.

If the current state is D then module D and all C modules are enabled, making D be implemented in parallel to all the C's. This is because possible states after next state could be executed in parallel with the next state. The results of either one or multiple C modules is accepted or ignored based on the state transition conditions. In Figure 5.3 (b) at third and fourth iterations, the result of only one C is accepted as the next state is C and possible state after the next state is D. Length of iteration in this case is equal to the processing time of module C (as processing time of module C is higher than D). In fifth iteration along with the module D and all C modules, module B is enabled with a time delay. This is because state transition conditions lead to B as next state after all possible C's and decision about accepting the result of B is already made within the allowed time delay. Based on the state transition conditions the results from two C modules are accepted. Average clock cycles at this iteration are equal to the sum of the processing time of State C (as processing time of State C is more than State D) and extra time needed by State B.

For the purpose of transitions between the states, control signals are generated which enable the modules of next state. In hardware implementation, the decoder controller manipulates these control signals depending on the conditions produced by the data calculated in various states or modules. Separate decoder controller components are developed for each of the sequential structures of the original, improved algorithms and parallel-pipeline structure of improved algorithm. Not only the next states but the possible states after the next states are also enabled if they could be executed with the next state in parallel. This concept is made as the basis in modeling the HDL code for the parallel-pipeline structure.

5.3 VLSI Design Flow

The design flow adopted in this thesis is shown in Figure 5.4

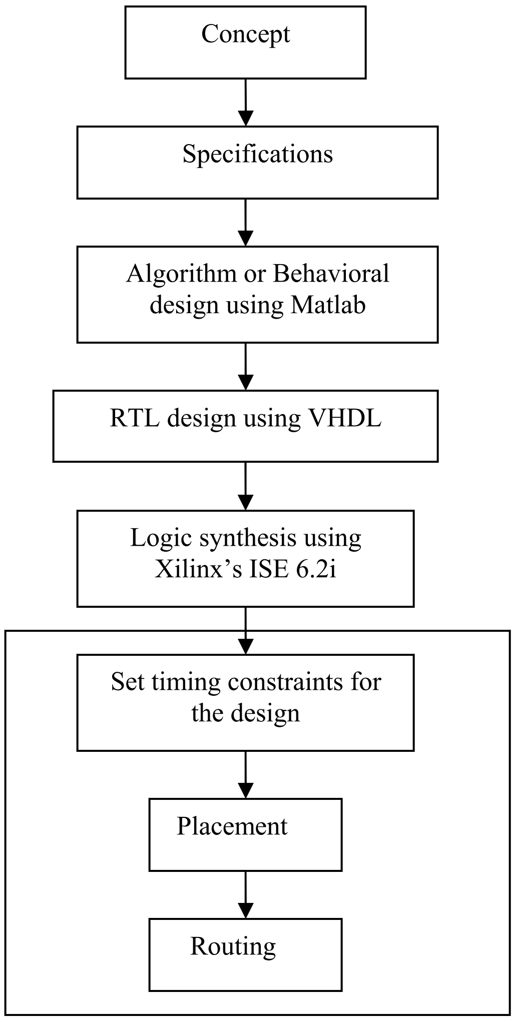
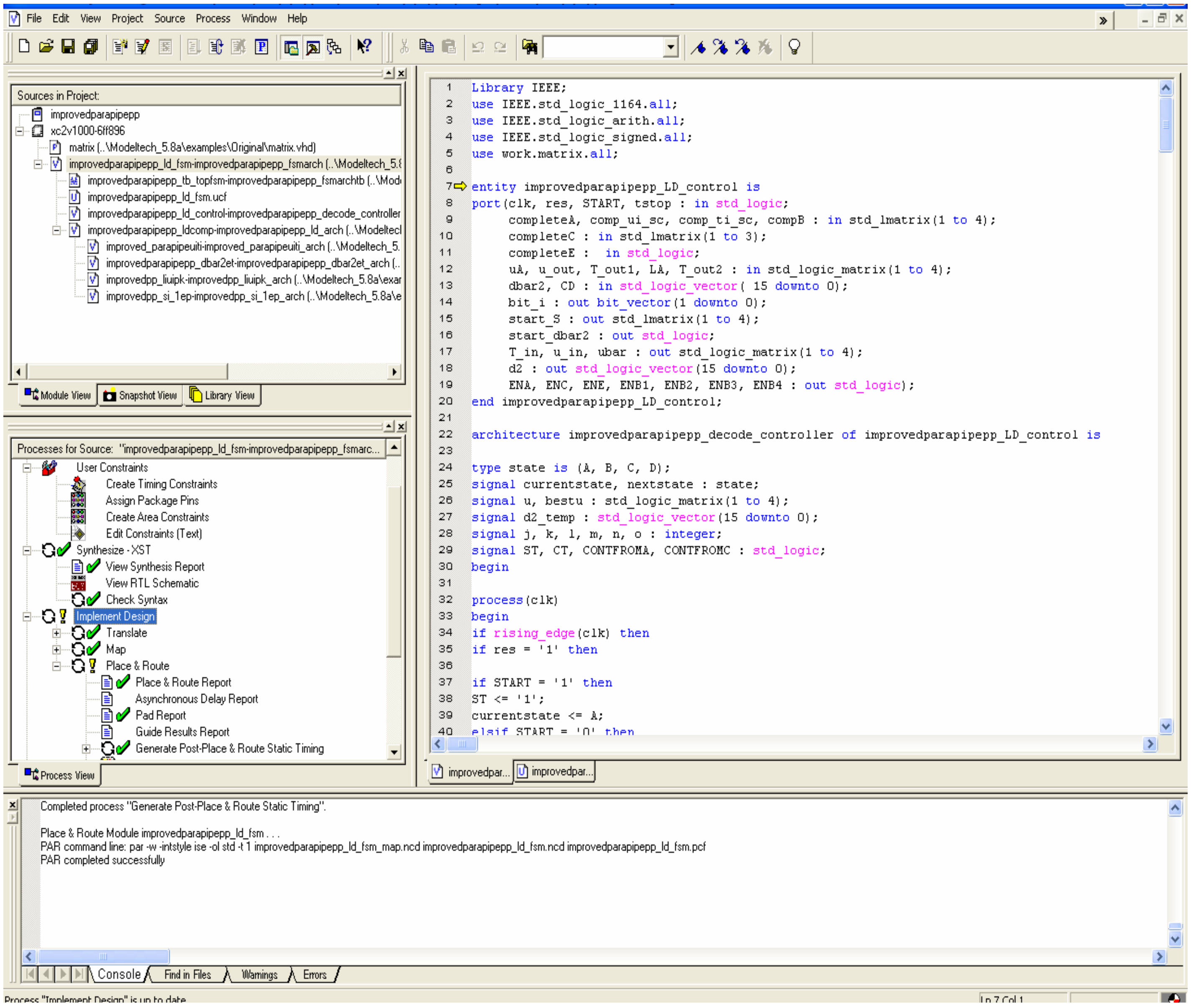

Concept
Specifications
Algorithm or Behavioral design using Matlab
RTL design using VHDL
Logic synthesis using Xilinx's ISE 6.2i
Set timing constraints for the design
Placement
Routing

Figure 5.4: Design flow for an FPGA

After designs are verified using RTL simulations the next most significant step is synthesis process which deals with rendering of a complete design described in VHDL into technology specific circuits. Logic synthesis is a process by which algorithmic descriptions of circuits are turned into a design for electronic hardware of some nature. Common examples of this process include synthesis of HDLs, including VHDL and Verilog. Logic synthesis tools may be used to automatically convert the RTL description of a digital system into a gate level description of the system. In all the implementations in this work, a synthesis tool from Xilinx called ISE 6.2i is used and the target technology being the device XC2V1000-6ff896 from wide range of Virtex-II FPGA family. Project Navigator is the user interface for Xilinx ISE and its work space is presented in Figure 5.5.


File Edit View Project Source Process Window Help
Sources in Project:
improvedparapipepp
xc2v1000-6ff896
matrix (..\Modeltech_5.8a\examples\Original\matrix.vhd)
improvedparapipepp_ld_fsm-improvedparapipepp_fsmarch (..\Modeltech_5.8
improvedparapipepp_tb_topfsm-improvedparapipepp_fsmarchtb (..\Mod
improvedparapipepp_ld_fsm.ucf
improvedparapipepp_ld_control-improvedparapipepp_decode_controller
improvedparapipepp_ldcomp-improvedparapipepp_ld_arch (..\Modeltecl
improved_parapipeuiti-improved_parapipeuiti_arch (..\Modeltech_5.
improvedparapipepp_dbar2et-improvedparapipepp_dbar2et_arch (..
improvedpp_liuipk-improvedpp_liuipk_arch (..\Modeltech_5.8a\exar
improvedpp_si_1ep-improvedpp_si_1ep_arch (..\Modeltech_5.8a\e
Module View
Snapshot View
Library View
Processes for Source: "improvedparapipepp_ld_fsm-improvedparapipepp_fsmarc...
User Constraints
Create Timing Constraints
Assign Package Pins
Create Area Constraints
Edit Constraints (Text)
Synthesize - XST
View Synthesis Report
View RTL Schematic
Check Syntax
Implement Design
Translate
Map
Place & Route
Place & Route Report
Asynchronous Delay Report
Pad Report
Guide Results Report
Generate Post-Place & Route Static Timing
Process View
improvedpar...
improvedpar...
Console
Find in Files
Warnings
Errors
Process "Implement Design" is up to date
Ln 7 Col 1

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use work.matrix.all;

entity improvedparapipepp_LD_control is
port(clk, res, START, tstop : in std_logic;
     completeA, comp_ui_sc, comp_ti_sc, compB : in std_lmatrix(1 to 4);
     completeC : in std_lmatrix(1 to 3);
     completeE :  in std_logic;
     uA, u_out, T_out1, LA, T_out2 : in std_logic_matrix(1 to 4);
     dbar2, CD : in std_logic_vector( 15 downto 0);
     bit_i : out bit_vector(1 downto 0);
     start_S : out std_lmatrix(1 to 4);
     start_dbar2 : out std_logic;
     T_in, u_in, ubar : out std_logic_matrix(1 to 4);
     d2 : out std_logic_vector(15 downto 0);
     ENA, ENC, ENE, ENB1, ENB2, ENB3, ENB4 : out std_logic);
end improvedparapipepp_LD_control;

architecture improvedparapipepp_decode_controller of improvedparapipepp_LD_control is

type state is (A, B, C, D);
signal currentstate, nextstate : state;
signal u, bestu : std_logic_matrix(1 to 4);
signal d2_temp : std_logic_vector(15 downto 0);
signal j, k, l, m, n, o : integer;
signal ST, CT, CONTFROMA, CONTFROMC : std_logic;
begin

process(clk)
begin
if rising_edge(clk) then
if res = '1' then

if START = '1' then
ST <= '1';
currentstate <= A;
elsif START = '0' then
```

```
Completed process "Generate Post-Place & Route Static Timing".

Place & Route Module improvedparapipepp_ld_fsm . . .
PAR command line: par -w -intstyle ise -ol std -t 1 improvedparapipepp_ld_fsm_map.ncd improvedparapipepp_ld_fsm.ncd improvedparapipepp_ld_fsm.pcf
PAR completed successfully
```

Figure 5.5: Workspace of Project Navigator

As seen in the Figure 5.5, all the necessary source files are added to the project which is seen in the sources for project window. Select the top-level source from the sources for project window, set the timing constraints, and then perform the “synthesize” step. This will synthesize the whole project. Then perform the “implement design” step. This step involves three steps to finally achieve the place and route report. The maximum frequency of the digital circuit design prototyped on a FPGA hardware platform can be obtained as the output. The RTL schematic of the decoder controller generated by Xilinx ISE 6.2i synthesis tool is shown in Figure 5.6.

5.4 Design Optimization

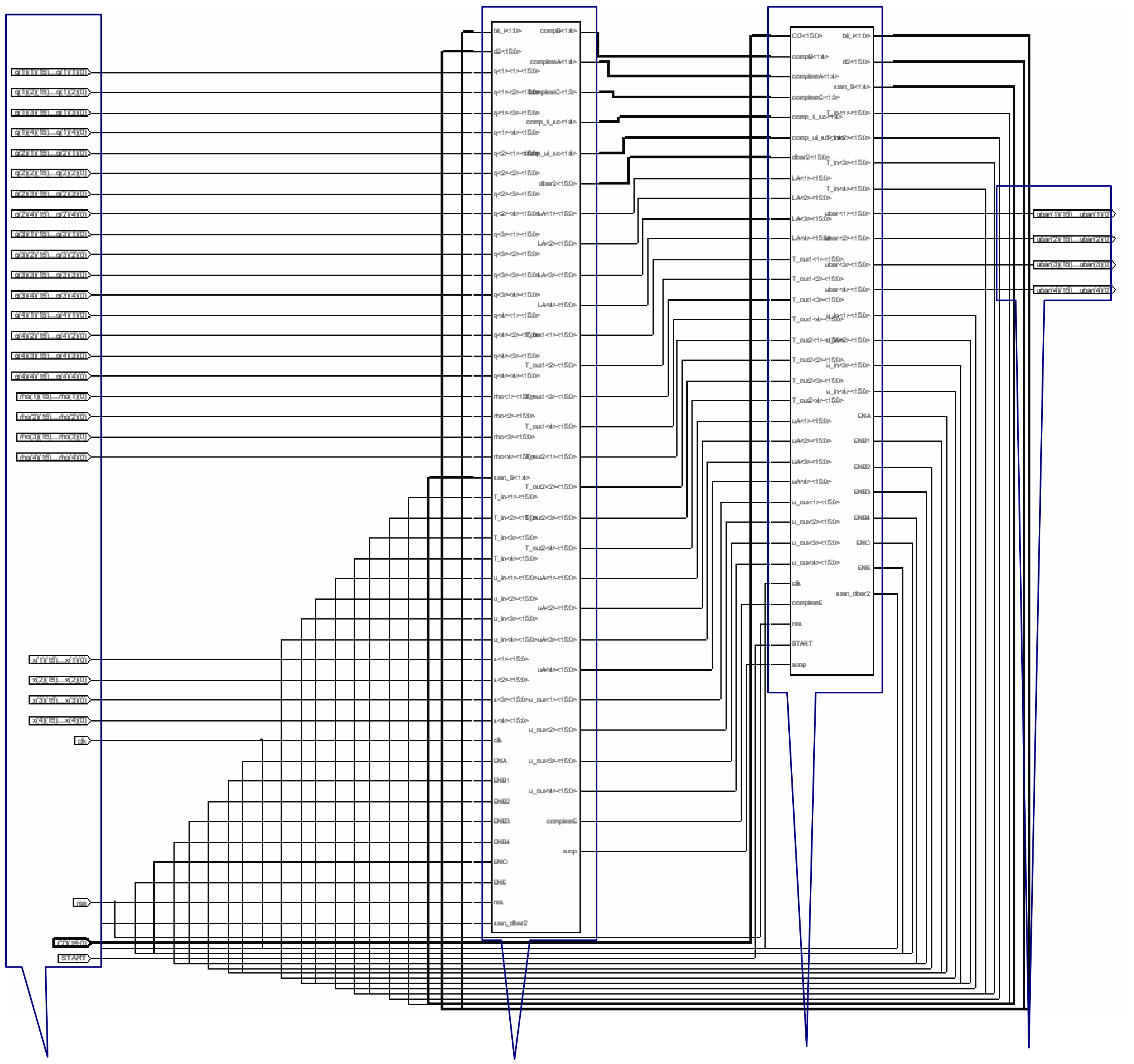
For a given lattice generation matrix *M*, the gram matrix $G = MM^T$ is computed on DSP. The Cholesky factorization of this gram matrix yields an upper triangular matrix *R* which is also performed on DSP. Then

$$Q(\xi) = \xi R^T R \xi^T =\} \| R\xi^T \|^2 = \sum_{i=1}^{n} (r_{ii}\xi_i + \sum_{j=i+1}^{n} r_{ij}\xi_j)^2 \le C$$

Substituting $q_{ii} = r_{ii}^2$ for $i = 1,....,n$ and $q_{ij} = r_{ij} / r_{ii}$ for $i = 1,.....,n,\ \ j = i+1,.....,n,$ from this, it is simplified to

$$Q(\xi) = \sum_{i=i}^{n} q_{ii} (\xi_i + \sum_{j=i+1}^{n} q_{ij}\xi_j)^2 \le C$$

where, *C* is the square radius of sphere centered at the received point and transformed into an ellipsoid centered at origin of the new coordinate system defined by ξ. The matrix Q is also computed on DSP and the results are passed on to FPGA. All the DSP computations are done in the pre-processing stage. The inverse of each of the diagonal element of matrix *Q*, i.e.,$1/q_{ii}$ is also computed on DSP. Thus using DSP to perform the computations of pre-processing stage simplifies the processing in FPGA. In most of the communication applications where decoding the large set of receiver data samples for a single channel matrix is a purpose, the pre-processing stage of the corresponding decoding algorithm has to be performed only once. This shows that pre-processing stage is in imbalance with the decoding stage in terms of number of computations or load of computations. And it is also known that pre-processing stage involves complex computations. Thus partitioning the irregular computation to DSP provides a good balance to entire system performance [Ma 2004].



Inputs to the Universal lattice decoder design.

COMP1: All inputs and outputs from different component or all component instantiations in a .vhd file.

COMP2: Decoder Controller or .vhd file of the decoder controller.

Output (i.e., the closest lattice point) from the Universal lattice decoder design.

6 RESULTS

This chapter gives the experimental results obtained for both the preprocessing and decoding part of the MIMO decoder. Efficient hardware model for the decoding part of the original and improved algorithm are developed and prototyped on to a Xilinx's VirtexII-1000 FPGA. The simulations results and the synthesis results are presented.

6.1 Experimental Setup

A system with 4-transmit and 4-receive antennas i.e., m=n=4 is assumed. The signal constellation linear over the field of real numbers is considered. The symbol set of 4-PAM constellation is ranging from {-3, -1, 1, 3}. The simulation tools used are Matlab 6.5 and Modelsim SE 5.8a to design the decoder at behavioral and RTL levels of abstraction. Xilinx ISE 6.2i is used as synthesis tool. Project Navigator 6.2.03i is the user interface for Xilinx ISE.

6.2 Pre-Processing Results

The pre-processing part involves computations like matrix inversion, transposition and Cholesky decomposition. Of these computations, matrix inversion and Cholesky decomposition are relatively more complicated and time consuming. The transpose operation takes negligible part of the processing time. The whole of pre-processing part was implemented on DSP. TI's TMS320c6711 is a floating point DSP, supports either real or integer arithmetic while TMS320c6201 is a fixed point DSP which supports only integer arithmetic. The maximum frequency on both DSP chips is at 200 MHz. Although floating point calculation is more accurate, it is time consuming and is not supported by VHDL. Therefore we need to calculate the fixed point processing times. The software tool used is Code Composer Studio.

For a 4x4 matrix, the time taken to execute the computations in pre-processing part is 19,645 clock cycles in floating point processing and 26,901 clock cycles in the fixed point processing.

$$T_{float} = 19645/(200*10^6) = 0.1ms$$

$$T_{fix} = 26901/(200*10^6) = 0.13ms$$

For a 8x8 matrix, the time taken to execute the computations in pre-processing part is 19,645 clock cycles in floating point processing and 26,901 clock cycles in the fixed point processing.

$$T_{float} = 98189/(200*10^6) = 0.49ms$$

$$T_{fix} = 141619/(200*10^6) = 0.71ms$$

6.3 Decoding Results

6.3.1 Simulation Results

The processing time taken by the prototyped lattice decoder with original algorithm and the improved algorithm are estimated. Based on the description of the simulation results of original algorithm in Chapter 3, it is observed that State A requires 37 clock cycles in the search process, 7 clock cycles are needed in both States B and D, and 2 clock cycles for State C. At each state, 1 clock cycle is needed for condition check and decision making about next state. For example, State A finds the upper bound and index of the element in the 36th clock cycle and 37th cycle is used in decision making.

The simulation results of improved algorithm as given in Chapter 4 shows that State A requires only 7 clock cycles in the search procedure. 7 clock cycles each for States B and C, and 3 clock cycles are needed for State D. The last clock cycle at each state is used for condition check and decision making. The bar-chart comparing the processing times needed at each state in both the algorithms is presented in Figure 6.1.

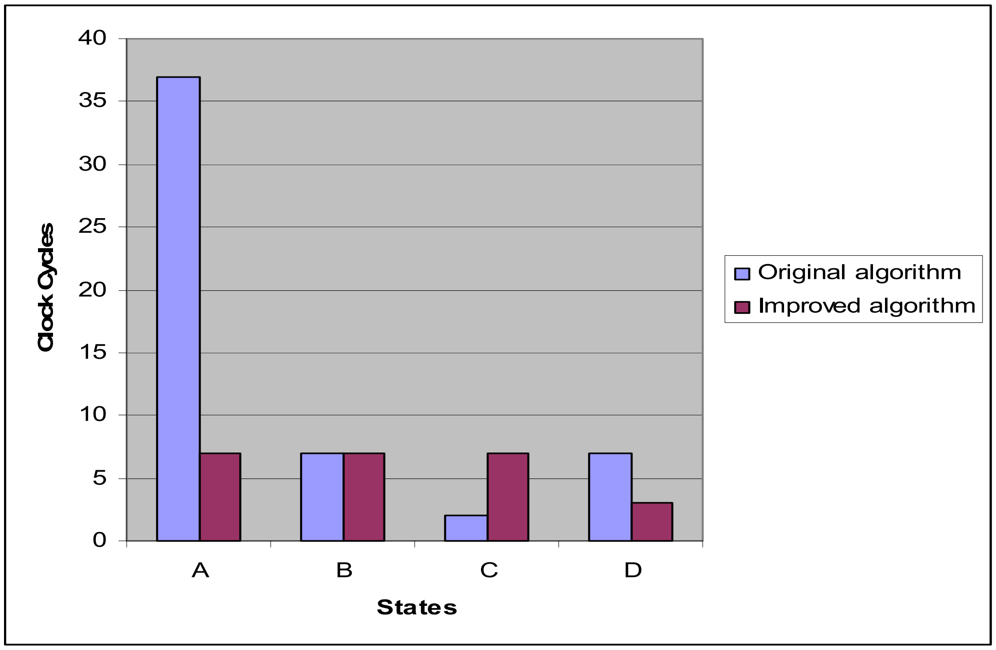


Figure 6.1: Bar chart showing the simulations times of each states in both algorithms

The performance of the sphere decoder is enhanced in the improved algorithm. There is a drastic reduction in the number of clock cycles required by the State A in the search procedure of improved algorithm compared to the original one.

The simulation results from Matlab gives the details about average number of times each state is visited. As the Matlab source code is executed for 10000 simulations or 10000 different sets of received signal vectors, average number of state visits obtained is a result for all 10000 simulations. Here in our thesis, one iteration means a visit to any state. Two or more states operating at the same time also count as one iteration (in case of parallel-pipeline implementation) [Ma 2005]

Table 6.1 is showing the number of state visits for 10000 simulations in each case of original and improved sphere decoding algorithms in their sequential implementation.

Table 6.1: Average number of state visits in sequential implementation at 20 dB

State	A	B	C	D
Original	163,872	132,742	101,389	32,982
Improved	84,328	64,221	43,900	43,494

Table 6.2 is showing the number of state visits for 10000 simulations of improved sphere decoding algorithm in its parallel-pipeline implementation. For more details about A | B, D || C, C | B, refer to Section 4.7 and Section 5.2.

Table 6.2: Average number of state visits in parallel-pipeline implementation at 20 dB

State	A|B B accepted	A|B B ignored	D||C C accepted	D||C C ignored	C|B B accepted	C|B B ignored
Improved-parallel-pipeline	60,816	23,514	13,407	10,107	3,407	19,980

6.3.2 Synthesis Results

After testing the functionality of both the sphere decoding algorithms using Matlab model of simulation, the core decoder function is designed using VHDL, simulated using Mentor Graphic's Modelsim, and prototyped on a device technology XC2V1000-6ff896C of Xilinx Virtex2 FPGA platform [Xilinx 2003]. Figure 6.2 gives the description of the device.

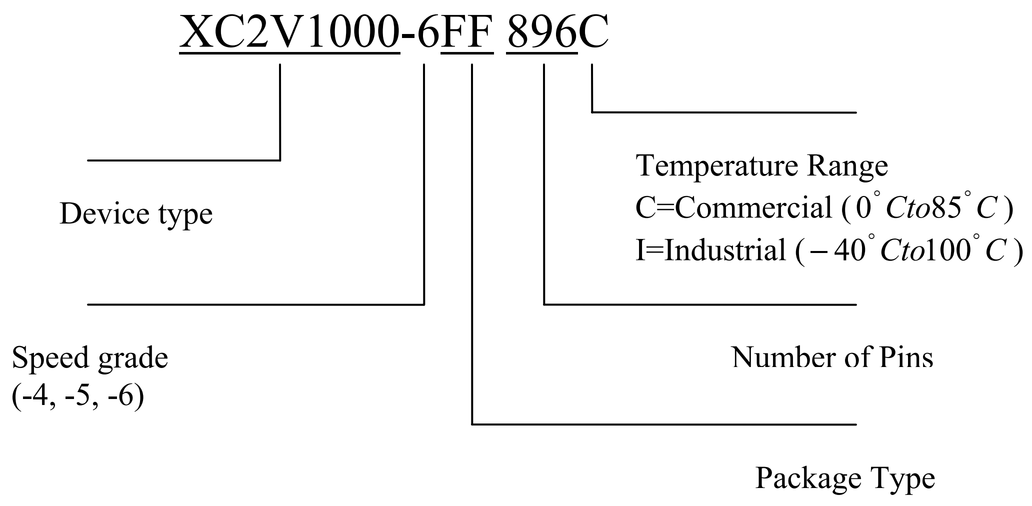


Figure 6.2: Xilinx Virtex-II 1000 FPGA Device Description

The simulation results of Matlab and hardware verify each other. Synthesis results of a sphere decoder with 4-transmit and 4-receive antennas when prototyped on a Xilinx Virtex-II 1000 FPGA using original and improved sphere decoding algorithms are shown in Table 6.3 below. The 18-bit embedded multipliers available on this FPGA are employed in the design to ensure the processing speed.

Table 6.3: Synthesis results of m=n=4 MIMO system

	Original algorithm	**Improved-Sequential**	**Improved-Parallel-Pipeline**
Target FPGA platform	Xc2v1000 -6	Xc2v1000 -6	Xc2v1000 -6
No. of External IOBs	387 out of 432	387 out of 432	387 out of 432
No. of Mult 18X18s	8 out of 40	20 out of 40	26 out of 40
No. of SLICEs	1168 out of 5120	2216 out of 5120	2347 out of 5120
No. of BUFGMUXs	1 out of 16	1 out of 16	1 out of 16
Max. freq	102.8 MHz	80.7 MHz	84.5 MHz

6.3.3 Decoding Rate

The bit rate of decoder is calculated as follows:

Rate = (*frequency* × *bits_per_dimension* × *n*) / (*total number of clock cycles*)

n = 4 for 4 - antenna system

bits_per_dimension = 2

$$Total\ number\ of\ clock\ cycles = \sum_{ni=1}^{tn} CPIT_{ni} * ITC_{ni}$$

where, $CPIT_{ni}$ is the number of cycles per ni^{th} iteration (one iteration here means a visit to any state. Two or more states operating at the same time also count as one iteration). This is obtained from the simulation results of VHDL i.e., from the waveform into which signals are added and allowed to run for some specified time.

ITC_{ni} is the count of the average number of times ni^{th} iteration or a particular state is visited. This is obtained from the Matlab simulations. As we have the data obtained for 10000 simulations, average count for one simulation is calculated and used in the decoding rate computation. Here in our case, this can be obtained by dividing the values in the tables 6.1 and 6.2 by 10000.

tn is the number of possible kinds of iterations or states. For sequential implementations, it is simply equal to number of states in the FSM of the decoder controller, whereas for parallel-pipeline implementation of the improved sphere decoding algorithm this can be obtained from the data dependency analysis. In our case, for sequential *tn* = 4, for parallel-pipeline *tn* = 6 (refer tables 6.1, 6.2)

In order to test both the original and improved form of sphere decoders, the same example as given in Chapter 4 is considered. In this case the lattice generator matrix *M* is some randomly generated matrix with zero mean and unit variance and SNR is set at 20 dB.

$$M = \begin{bmatrix} 0.2944 & -0.6918 & -0.4410 & 0.8156 \\ -1.3362 & 0.8580 & 0.5711 & 0.7119 \\ 0.7143 & 1.2540 & -0.3999 & 1.2902 \\ 1.6236 & -1.5937 & 0.6900 & 0.6686 \end{bmatrix}$$

Then the received signal obtained after scaling and rounding is $\begin{bmatrix} 385 & 119 & -130 & -376 \end{bmatrix}$ when the transmitted signal constellation is $\begin{bmatrix} 3 & 1 & -1 & -3 \end{bmatrix}$.

For above described example, and considering the case of original sphere decoding algorithm, the total number of clock cycles required to complete the search procedure are:

Total number of cycles = 37 × 16.3 (number of iterations with State A) + 7 × 13.2 (number of iterations with State B) + 2 ×10.1 (number of iterations with State C) + 7 × 3.2 (number of iterations with State D) = 738 cycles

Bit rate = (108.2 MHz * 4 * 2) / 738 = 1.17 Mbit/s

For improved algorithm, a parallel-pipeline architectural model is also developed as hardware implementation on FPGA can make use of an additional parallelism feature. In both sequential and parallel-pipeline implementations, the number of iterations, states at each iteration and average clock cycles per iteration vary significantly. This can be explained in detail by looking at the sequence of states in both cases.

The sequential procedure for improved sphere decoding algorithm for above considered example needs 20 iterations, while parallel-pipeline procedure needs only 10 iterations as shown in Table 6.2 and 6.3 respectively.

Table 6.4: Sequence of state in Sequential procedure

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
State	A	B	A	B	A	B	A	D	A	B	A	B	A	B	A	D	C	C	C	C

Table 6.5: Sequence of state in Parallel-Pipeline procedure

Iteration	1	2	3	4	5	6	7	8	9	10
State	A	A	A	A	D	A	A	A	A	D
Parallel state					C's					C's
Pipeline state	B	B	B	B		B	B	B	B	

As we know the states in parallel can begin at the same time, at the iteration 5 from Table 6.5, State D and all possible State C's are executed at the same time but the output obtained from all C states are ignored as the conditions lead to State A after this State D. At the end of State D simulation, parallel C states are interrupted and disabled. Therefore, the average number of cycles at this iteration is equal to the simulation time of State D. Similarly in iteration 10, State D and all C states are executed in parallel and the results are accepted as the conditions satisfy and the search procedure ends. Here State C is allowed to complete as the conditions lead to State C as next state after current State D. Therefore, the average number of cycles at this iteration is equal to the simulation time of State C.

In Table 6.5, we see at iteration 1, State B is implemented in pipeline to State A. And so State B is enabled a little while after State A. This case of pipelining also supports partial parallelism. Here the output from State B is accepted as the conditions of state transitions lead to State B as next state. Therefore, the average number of cycles at this iteration is equal to the sum of simulation time of State A and extra time needed by State D. Similar is the case at iterations 2, 3, 6, 7, 8. At iteration 4, State B is implemented in pipeline to State A and so State B is enabled a little while after State A. But the output of State B is ignored as conditions of state transitions lead to State D as next state. Therefore, the average number of cycles at this iteration is equal to the simulation time of State A alone.

The total number of clock cycles required by the improved algorithm to complete the search procedure at 20 dB in both sequential and parallel-pipeline procedures is as follows:

Sequential:

Total number of cycles = 7 × 8.4 (number of iterations with State A) + 7 × 6.4 (number of iterations with State B) + 7 × 4.4 (number of iterations with State C) + 3 × 4.3 (number of iterations with State D) = 147 cycles

Bit rate = (80.7 MHz * 4 * 2) / 147 = 4.39 Mbit/s

Parallel-Pipeline:

Total number of cycles = 13 × 6.1 (number of iterations with A|B, B accepted) + 7 × 2.3 (number of iterations with A|B, B ignored) + 3 × 1.0 (number of iterations with D||C, C ignored) + 7 × 1.0 (number of iterations with D||C|B, C accepted, B ignored) + 10 × 0.3 (number of iterations with D||C|B, C accepted, B accepted) = 108 cycles

Bit rate = (84.5 MHz * 4 * 2) / 108 = 6.26 Mbit/s

The comparison of decoding rate for original sphere decoding algorithm and improved-sequential, improved-parallel-pipeline algorithms are shown in Table 6.6.

Table 6.6: Comparison of decoding rate at 20 dB

	Original	**Improved-Sequential**	**Improved-Parallel-Pipeline**
Total number of clock cycles	738	147	108
Max Frequency	102.8 MHz	80.7 MHz	84.5 MHz
Decoding Rate	1.17 Mbit/s	4.39 Mbit/s	6.26 Mbit/s

The sequential architecture of the original and improved algorithms offer a decoding rate of 1.17 Mbit/s and 4.39 Mbit/s respectively when implemented on a device technology XC2V1000-6FF896 of Xilinx VirtexII-1000 FPGA platform. From the synthesis results we observe that the maximum frequency of the original sphere decoder is higher compared to the improved form of algorithms. Although this is the case, the decoding rate of the improved sphere decoding algorithm is far better and shows a lot of improvement from the original algorithm.

This is because of better values of number of clock cycles per iteration, $CPIT_{ni}$ and count of average number of times particular iteration or state is visited, ITC_{ni} for the improved sphere decoding algorithm. They contribute to the better decoding rate the decoder.

The bit rate of the decoder with improved algorithm and utilizing the parallelism and pipelining features is 6.26 Mbit/s.

From table 6.6 we observe that the decoding rate of the improved sphere decoding algorithm whose flow chart given in Figure 4.1 in sequential implementation is 3.75 times faster than the original sphere decoding algorithm shown in Figure 3.1. The parallel-pipeline implementation of the improved sphere decoding algorithm is 5.35 times faster than the sequential implementation of the original algorithm when corresponding architectural models of both the algorithms are prototyped on FPGA platform. In case of improved algorithm, the parallel-pipeline architecture speeds up the search procedure by 1.43 times compared with its sequential architecture. Thus the parallel-pipeline architectural model of improved sphere decoder when prototyped on a device XC2V1000-5FF896 of Xilinx's VirtexII-1000 FPGA platform could reach a decoding rate up to 6.26 Mbit/s with a spectral efficiency 2 bits/dimension at SNR of 20 dB.

6.3.4 BER Performance

Using Matlab, BER performance for both original and improved sphere decoding algorithms has been estimated for a particular Gaussian distributed lattice generator matrix and at different SNRs. Figure 6.3 shows BER versus Eb/No (dB) of an uncoded system for $m=n=4$ using original and improved sphere decoding algorithms.

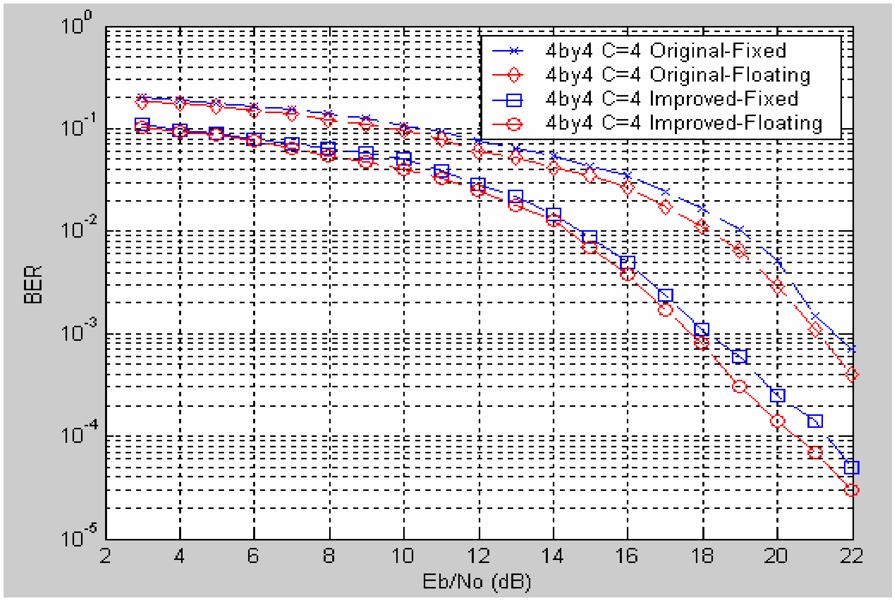


Figure 6.3: BER vs. Eb/No (dB) for an uncoded system using original and improved algorithms

From Figure 6.3, we can observe that improved sphere decoding algorithm shows better BER performance than the original algorithm. This means the number of bit errors reported in the improved algorithm is less compared to that of original algorithm. Both the algorithms are executed for fixed point hardware simulation and floating point software simulation to compare the performance. It is observed from Figure 6.3 that the BER of the fixed point implementation matches the floating point implementation. The Matlab fixed point simulation results also verify with VHDL simulation results.

6.3.5 Comparison between FPGA and DSP Implementations

The comparison of sphere decoding algorithm implementations on DSP and FPGA are as shown in the Table 6.7. The parallel-pipeline architecture of improved sphere decoding algorithm was implemented on FPGA and sequential architecture of original sphere decoding algorithm was implemented on DSP.

Table 6.7: Comparison between FPGA and DSP implementation at 20 dB

Platform	FPGA	DSP
Max Freq	84.5 MHz	200 MHz
Total cycles	108	27,492
Bits/dimension	2	2
Dimension	4	4
Decoding Rate	6.26 Mbit/s	0.06 Mbit/s

From the above comparison we can observe that the decoding rate of parallel-pipeline implementation of an improved sphere decoding algorithm when prototyped on FPGA is approximately 100 times faster than the sequential implementation of the original sphere decoding algorithm prototyped on DSP. Although the frequency on DSP decoder is twice the frequency of the FPGA decoder, we visualize a better performance for the FPGA prototype of the sphere decoder. Total cycles needed to complete the search procedure and obtain a closest lattice using parallel-pipeline implementation of the improved sphere decoding algorithm prototyped on a device technology XC2V1000-6FF896C of Xilinx's VirtexII-1000 FPGA platform is 107. Whereas in case of sequential implementation of the original sphere decoding algorithm prototyped on a TI's TMS320c6201 DSP chip, the total clock cycles consumed are

27,492. The couple of reasons for such a high variation in total number of clock cycles in both the cases are interpreted as follows.

- Algorithmic model is improved.

The improved form of the sphere decoding algorithm does not need to perform square root operation while computing an integer component value or the index value u_i and the its upper bound L_i at each layer in the 4-dimensional (4-transmit 4-receive antenna system) space created at the receiver end. A significant count of clock cycles are saved due to this in the improved sphere decoding algorithm compared to the original one.

- Parallel-pipeline implementation.

An additional feature of FPGA over DSP chip is its support of parallelism. Therefore a parallel-pipeline architectural model is developed for improved sphere decoding algorithm and is prototyped on FPGA as it supports parallelism. Due to possible parallelism and pipelining the FPGA implementation of improved sphere decoding algorithm saves in number of clock cycles essentially.

Hence we observe that improved sphere decoding algorithm in its parallel-pipeline implementation and prototyped on FPGA reaches a decoding rate of 6.26 Mbit/s. And the original sphere decoding algorithm in its sequential implementation and prototyped on DSP chip reaches a decoding rate of 0.06 Mbit/s. The FPGA implementation is 104.33 times faster than the DSP. The DSP implementation is very slow compared to FPGA prototype because the original sphere decoding algorithm performs the iterative search procedure involving square root computation while finding the bounds of the lattice index point at each layer.

6.4 Conclusions

Design and implementation of universal lattice decoder is presented in this thesis. Firstly the functionality of the original sphere decoding algorithm is examined using Matlab simulation. Then a VHDL model is developed for the core decoder function and simulated at RTL level of abstraction using Mentor Graphics' Modelsim SE 5.8a. Because the simulation results show imbalance in the processing time of each individual state, which is not practical for parallel implementation, original algorithm is modified such that square root computation is avoided, as a result an improved universal lattice decoding algorithm is proposed. Functionality testing procedure similar to that of original algorithm is carried out for the improved algorithm. The primary focus in this thesis has been to design an efficient hardware architectural model for the improved sphere decoding algorithm and implement it on FPGA platform.

Based on the data dependency analysis, a parallel-pipeline architectural model is developed for the improved sphere decoding algorithm. Both sequential and parallel-pipeline architectural models are developed in VHDL and are simulated at RTL level of abstraction. All the hardware architectural models are synthesized using Xilinx ISE 6.2i synthesis tool. The device technology XC2V1000-6FF896C of Xilinx VirtexII-1000 FPGA platform is used to prototype the architectural models. BER performance of both original and improved sphere decoding algorithms has also been estimated. When a MIMO system of 4-transmit and 4-receive antennas with 4-PAM modulation is considered, the decoding throughput of 6.32 Mbit/s is achieved for parallel-pipeline implementation of the improved sphere decoding algorithm at 20dB SNR. The parallel-pipeline implementation of improved sphere decoding algorithm is 1.44 times faster than its own sequential implementation and is 5.4 times faster when compared to the sequential implementation of original sphere decoding algorithm when all the hardware architectural models are prototyped on FPGA platform. Comparing the FPGA and DSP implementations, it is concluded that parallel-pipeline implementation of the improved sphere decoding algorithm prototyped on FPGA achieves a decoding throughput of 6.32 Mbit/s, which is about two orders of magnitude faster than the sequential implementation of the original sphere decoding algorithm prototyped on a DSP chip.

REFERENCES

1. [Viterbo 1999] E.Viterbo and J. Boutros, “A universal lattice code decoder for fading channels”, pp 1639-1642, IEEE trans. on Inf. Theory, July.1999.

2. [Ma 2004] Jing Ma and Xinming Huang, “Design of Lattice Decoder for MIMO Systems in FPGA”, *Proceedings of IEEE Workshop on Signal Processing Systems(SiPS’04), Austin, Texas,* pp. 24-29, October 13-15, 2004.

3. [Burg 2004] A. Burg, M. Wenk, M. Zellweger, M. Wegmueller, N. Felber, and W. Fichtner, “VLSI implementation of the Sphere Decoding Algorithm”, pp 303-306, *Proceedings of the 30th European Solid-State Circuits Conference(ESSCIRC’04), Integrated Systems Laboratory, ETH-Zurich, Switzerland,* Sep 21-23, 2004.

4. [Gregory 1999] Gregory C.Ahlquist, Michael Rice, and Brent Nelson, “Error control coding in software radios: An FPGA approach”, *IEEE commun*, pp 35-39, August 1999

5. [Srikanteswara 2003] Srikathyayani Srikanteswara, Ramesh chembil Palat, Jeffrey H.Reed, and Peter Athanas, “ An Overview of Configurable Computing Machines for Software Radio Handsets”, *IEEE commun Mag*, pp 134-141, July 2003

6. [Adjoudani 2003] Ali Adjoudani, Eric C, D Haessig and Salim Manji “Prototype Experience for MIMO Blast Over Third-Generation Wireless System.” Vol. 21, pp 440, *IEEE Journal on Selected areas in communication*, April 2003.

7. [Damen 2000] Damen, A. Chkeif, and J.C. Belfiore, “Lattice code decoder for space-time code”, Vol. 4. No.5, pp161-163, *IEEE Communications Letters*, May 2000

8. [Viterbo 1993] E. Viterbo and E. Biglieri: “A universal decoding algorithm for lattice codes”, *Quatorzieme colloque GRETSI*, pp. 611-614, Juan-les-Pins, September 1993.

9. [Bertrand 2003] Bertrand M. Hochwald and Stephean ten Brink, “Achieving Near-Capacity on a Multiple-Antenna Channel”, *IEEE Transaction on communications,* vol. 51, PP 389-399, March 2003

10. [Cummings 1999] Mark Cummings, and Shinichiro Haruyama, “FPGA in the Software Radio”, pp 108-112, *IEEE Communications Magazine,* Febraury 1999.

11. [Eriksson 2002] Thomas Eriksson, Erik Agrell and Kenneth Zeger, “Closest Point Search in Lattices,” Vol. 48, pp 2201, *IEEE Transaction on Information theory*, August 2002.

12. [Chris] Chris H. Dick, “Design and Implementation of High-Performance FPGA Signal Processing Data paths for Software Defined Radios”, Xilinx Inc.

13. [Garrett 2002] David Garrett, Chris Nicol, “Multipath expands RF bandwidth”, *EE Times*, November 08, 2002.

14. [Jones 2003] V.K. Jones, Greg Raleigh, Richard van Nee, “MIMO answers high-rate WLAN call”, *EE Times*, December 31, 2003.

15. [Xilinx 2003] Virtex II Platform FPGA User Guide, UG001, v1.6.1, Xilinx Inc., August 2003

16. [Love 2004] D. J. Love, R. W. Heath Jr., W. Santipach, and M. L. Honig, “What is the value of limited feedback for MIMO channels?,” *IEEE Commun. Mag.*, vol. 42, no. 10, pp 4-59, October 2004

17. [Smith 1997] Douglas J Smith, “HDL Chip Design” Doone Publication, 1997.

18. [Gesbert 2005] http://heim.ifi.no/~gesbert/mimo_research.htm

19. [Piromsopa 2001] K.Piromsopa, C.Aprontewan, and P.Chogsatitvatana, “An FPGA implementation of a fixed-point square root operation”, *ISCIT* 2001

20. [Pohst 1985] M.Pohst, “On the computation of lattice vectors of minimal length, successive minima and reduced basis with applications,” *ACM SIGSAM Bull.,* vol. 15, pp. 463-471, April 1985.

21. [Damen 2003] Damen, M.O., H.E. Gamal, and G.Caire, “On maximum-likelihood detection and the search for the closest lattice point,” *IEEE Transaction on Information theory*, vol. 49, no.10, 2389-2402, October 2003

22. [Zheng 2003] L.Zheng, David N.C Tse, “Diversity and Multiplexing: A fundamental tradeoff in multiple-antenna channels,” *IEEE Transaction on Information theory*, vol.49, no.5, 1073-1096, May 2003

23. [Lee 2003] Hoo-Jin Lee, Shailesh Patil, and Raghu G. Raj, “Fundamental overview and simulation of MIMO systems for Space-Time coding and Spatial Multiplexing,” *WNGC, Austin, Texas*, May 2003

24. [Ma 2005] Jing Ma and Xinming Huang, “A System-on-programmable chip approach for MIMO sphere decoder”

25. [Kevin 2003] Kevin Morris, “Implementing high-performance DSP Algorithms in FPGA’, FPGA and Programmable Logic Journal”, www.fpgajournal.com, October 2003

26. [Dan 2004] Dan Ganousis, “Top-Down DSP Design Flow to Silicon Implimentation”, www.fpgajournal.com, March 2004

27. [ED 2000] “Design Technology Advances Unleash New FPGA Capabilities” http://www.elecdesign.com/Articles/Index.cfm?AD=1&ArticleID=4405, June 2000

VITA

Swapna Kura was born on the March 15th, 1980, in Hyderabad, India. She aspired to be an Engineer since she was young and her mathematical skills encouraged her to pursue her undergraduate degree in Electrical Engineering. She graduated from Jawaharlal Nehru Technological University, India, in July 2001. Her aspirations to research and learn more about her subject made her get into an International Graduate school. She joined the University of New Orleans in 2002 and majored in Electrical and Computer Engineering. Her research interests are mainly related to FPGA, VHDL and Hardware Design.