Implementation of 2x2 soft-input soft-output sphere decoder

Yan, J.

Award date:
2013

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Implementation of 2×2 Soft-Input Soft-Output Sphere Decoder

Author
Jia Yan
0789055
Embedded Systems

Supervisor
Prof.dr.ir. C.H. van Berkel

Tutor
Dr.ir. Özgün Paker

Date
July 25, 2013

Confidential Statement

This MSc thesis is based on thesis work performed at ST-Ericsson Eindhoven, The Netherlands. The main technical contributions have been made in the context of product architectures. All references to these product architectures, including the product perspectives of the outcomes of the work have been removed from the public version of this thesis.

The complete version of this thesis is confidential. When access to this full version is required for (academic) audit purposes, please contact the supervisor of the thesis work, Prof. C.H. van Berkel (c.h.v.berkel@tue.nl)

Abstract

MIMO (Multiple-input Multiple-output) technique which applies multiple antennas on both transmitter and receiver sides is considered as the most attractive topic in the wireless communication field. In order to suppress the interferences incurred by multiple antennas, various decoders are developed.

This thesis proposes an algorithm of a near-optimal sphere decoder which can cooperate with apriori information coming from turbo decoder. This is known as the Turbo-equalizer. We present the formula of LLR computation of a Soft-in Soft-out sphere decoder, and realize that to solve the formula exactly is impossible since the exact solution leads to exhaustive search which has an exponential complexity with the number of antennas.

CONFIDENTIAL PARAGRAPH

We also present a hardware solution of the complexity reduced sphere decoder algorithm and discuss the possible tradeoff between area and throughput. In Chapter 5, implementation details of each block are presented, and corresponding synthesis results are illustrated in Chapter 6. We make a comparison between the new sphere decoder and the ST-Ericsson's old design and argue the differences between them at the end of this thesis.

Technische Universiteit
Eindhoven
University of Technology

Table of contents

Title
Implementation of 2×2 Soft-Input Soft-Output Sphere Decoder

Technische Universiteit
Eindhoven
University of Technology

Table of contents

Title
Implementation of 2×2 Soft-Input Soft-Output Sphere Decoder

Technische Universiteit
Eindhoven
University of Technology

Table of contents

Title
Implementation of 2×2 Soft-Input
Soft-Output Sphere Decoder

Where innovation starts

1 Introduction

1.1 Sphere Decoder

In order to fulfill the demands of working, living and entertainment, researchers exploit various techniques to provide faster and reliable communication. Among all these techniques, MIMO(Multiple-input Multiple-output) applies multiple antennas on both transmitter and receiver sides. This technique is considered as the most attractive topic since it exploits the degree of freedom in a rich scattering channel. The MIMO technique has become an important part of some of the modern communication standards, for instance WLAN, LTE and WiMax[1].

However, the interferences incurred by multiple antennas limit the channel capacity, and brings new challenge to receiver design. A receiver must be able to suppress this interference and perform low BER(Bit Error Rate) in MIMO communication mode. Various equalizers are developed for the purpose of interference suppression. The well known ones are Zero-Forcing (ZF) and Minimum Mean Square Error (MMSE) detectors. The implementations of linear detector for today's MIMO based air interfaces with a small number of transmission and receiving antennas have low complexity and therefore are attractive. For instance, in [2] authors study the software implementation of a 2×2 MMSE detector and map it on an embedded vector processor. However, linear detectors suffer bad BER performance especially in cases of low SNR and high correlated channels.

On the other end of the spectrum, there are non-linear detectors. E.Viterbo presented a non-linear decoder for arbitrary lattice codes and provided a near-optimal performance[3]. Since the decoder searches points inside a sphere space of a given radius, this technique is called *Sphere Decoding*. The early implementations of the Sphere Decoder consider a conventional receiver architecture where equalization and channel decoding processes are separate[4, 5, 6]. Actually combining equalization and channel decoding provides an even lower BER[7]. This scheme is known as Turbo-equalization. Turbo-equalization requires that the non-linear detector is capable of incorporating a-priori softbits. In this work, we investigate how we can address this challenge.

1.2 Problem Description

In the early designs of the Sphere Decoder, apriori information is not involved. Authors in [4] addressed a transformation from a minimum distance search procedure to a depth-first tree search together with a branch pruning mechanism, continually shrinking the search

space. In order to execute the tree search efficiently, it is required to sort the candidates. For instance, during the forward search, the child symbol that is preferred is the one that minimizes the distance increment; At each level which corresponds to one spatial layer, the candidates are ordered in ascending order according to the Partial Euclidean distance matric (Section 2.4). This scheme is known as Schnorr-Euchner ordering[8]. When a backtrack happens during tree search procedure, the order simply helps to find the next candidate. Hence for implementation of both cases, finding the preferred child and sorting, existing literature makes use of only the geometrical relation between the received signal and the signal constellation to save complexity.

We try to incorporate apriori information into Sphere Decoder which can be used in Turbo-equalization in order to pursue lower BER performance. However, when the apriori information is given as a constraint, geometrical order of QAM symbols is distorted by the apriori contribution of each symbol. Equation (2.9) shows that the LLR computation requires not only the Euclidean distance information but also the contribution of apriori information. Hence to solve (2.9), exhaustive search is inevitable due to the fact that for QAM symbols there exists no relationship between their Euclidean distances and their apriori information. E.g. the closest candidate based on Euclidean distance and the most probable candidate based on apriori information may not be the same symbol. Our challenge is to approximate the LLR by finding the "appropriate" symbols for the forward-traversing of the tree search.

The contents of this thesis can be divided into two parts. The first part includes Chapter 2 and Chapter 3. In these chapters, we address the principle of Sphere Decoder and the investigation of low-complexity algorithm. At the end of Chapter 3, a new forward traversing search mechanism is proposed. Compared with conventional algorithm, the new mechanism reduces the complexity significantly with little performance penalty. The discussions in Chapter 5 present the concept of High Level Synthesis and the implementation details of HLS model designed by using Cataplt C.

1.3 Contributions

The sphere decoder algorithm proposed in this thesis is based on the work of Özgün Paker. The thesis also studies the impacts of different parameters on the pruning method and simplifies the pruning method according to the simulations conducted in Chapter 3. The main contribution of this thesis is to address hardware implementation of a 2×2 soft-input sphere decoder. The detail implementation aspects of each block are presented in Chapter 5. Moreover, the thesis explores the tradeoff between area and throughput of the hardware implementation.

2 MIMO transmission and Detection

In this chapter, we will show the principle of a soft-input soft-output sphere decoder. Section 2.2 introduces the MIMO transmission model. Subsequently, Section 2.3 shows that with max-log approximation LLR computation can be converted to an Euclidean distance computation. After the introduction of Max-Log approximation, we introduce a pre-processing stage. The pre-processing stage enables a depth-first tree search scheme to solve the minimum distance search, which is presented in Section 2.4.

2.1 Notation

The notations used in this chapter are defined as follows. Matrices are written in bold capital letter, e.g. channel matrix $\mathbf{H}$. $\mathbf{A}_j$ is the jth column of the matrix, and a_{ij} represents a element in the ith row and jth column of matrix. Vectors are shown with bold lowercase letters, for example $\mathbf{s} = [s_1\, s_2\, \cdots\, s_n]^T$ denotes a vector, where s_i is the element at ith position.

2.2 MIMO Transmission

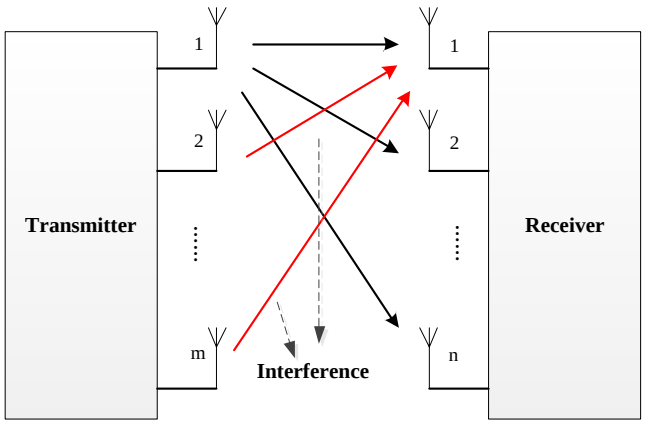


Figure 2.1: MIMO transmission model with m transmitters and n receivers

Figure 2.1 shows a MIMO transmission example, with m transmit antennas and n receive antennas. The MIMO transmission is represented in matrix form:

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n} \tag{2.1}$$

where $\mathbf{H} \in \mathbb{C}^{n\times m}$ is the channel matrix, $\mathbf{y} \in \mathbb{C}^n$, $\mathbf{s} \in \mathbb{C}^m$ and $\mathbf{n} \sim \mathcal{C}N(0, \sigma_0\mathbf{I}_n)$ denote received signal, transmitted signal and white Gaussian noise respectively. Each transmitted symbol s_i is chosen from a complex constellation set $\mathcal{O}$. Every constellation point in this set is assigned with a binary sequence of length of q, such that the transmitted signal vector $\mathbf{s}$ is corresponding to a binary vector with mq-dimension. We denote this sequence as $\mathbf{x} \in \mathbb{R}^{mq}$. An entry $x_{i,j}$ is the bit value of jth position corresponding to symbol s_i, where $j = 1\cdots q$ and $i = 1\cdots m$, $x_{i,j} \in \{0, 1\}$. A detector with given channel knowledge $\mathbf{H}$ finally derive an estimation of vector $\mathbf{x}$ for a received signal vector $\mathbf{y}$. One can observe that there exists 2^{mq} possible solutions.

2.3 LLR computation using Maximum-log Approximation

Compared with hard-output decoder, soft-in soft-output decoder does not give the bit value, but the probability of bit value. This probability is represented in Log-Likelihood Ratio (LLR) defined as:

$$L_{i,j} = \log \frac{P(x_{i,j} = 1|\mathbf{y}, \mathbf{H})}{P(x_{i,j} = 0|\mathbf{y}, \mathbf{H})} \tag{2.2}$$

Apply Bayes' theorem to the numerator and denominator of (2.2), the probabilities are:

$$\begin{aligned} P(x_{i,j} = 1) &= \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} P(\mathbf{s}|\mathbf{y}, \mathbf{H}) \\ &= \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} \frac{p(\mathbf{y}|\mathbf{s}, \mathbf{H})P(\mathbf{s})}{p(\mathbf{y})} \end{aligned} \tag{2.3}$$

and

$$\begin{aligned} P(x_{i,j} = 0) &= \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} P(\mathbf{s}|\mathbf{y}, \mathbf{H}) \\ &= \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} \frac{p(\mathbf{y}|\mathbf{s}, \mathbf{H})P(\mathbf{s})}{p(\mathbf{y})} \end{aligned} \tag{2.4}$$

Substitute (2.3) and (2.4), (2.2) is rewritten in:

$$L_{i,j} = log \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} p(\mathbf{y}|\mathbf{s}, \mathbf{H})P(\mathbf{s}) - log \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} p(\mathbf{y}|\mathbf{s}, \mathbf{H})P(\mathbf{s}) \tag{2.5}$$

According to the transmission model in section 2.2, the conditional distribution of received signal vector $\mathbf{y}$ follows Gaussian distribution that:

$$p(\mathbf{y}|\mathbf{s}, \mathbf{H}) = \left(\frac{1}{\sigma^2\pi}\right)^n \exp\left(-\frac{\|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2}{\sigma^2}\right) \tag{2.6}$$

Therefore, the LLR of corresponding bit position is :

$$L_{i,j} = log \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} \exp\left(-\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2}\right) P(\mathbf{s}) - log \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} \exp\left(-\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2}\right) P(\mathbf{s}) \tag{2.7}$$

However, (2.7) is not a feasible algorithm for LLR computation, since the logarithm and exponent function have high complexity. Fortunately, [9] gives a simple solution as known as the max-log approximation which corresponds to :

$$log \sum_{i=1\cdots n} \exp(x_i) \approx \max_{i=1\cdots n} x_i \tag{2.8}$$

With this method, the approximation of LLR follows that:

$$\begin{aligned} L_{i,j} &\approx \max_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} \left(-\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2} + \log P(\mathbf{s})\right) - \max_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} \left(-\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2} + \log P(\mathbf{s})\right) \\ &= \min_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} \left(\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2} - \sum_{i=1}^{m} \log P(s_i)\right) - \min_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} \left(\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2} - \sum_{i=1}^{m} \log P(s_i)\right) \end{aligned} \tag{2.9}$$

When the apriori probabilities of all symbols are the same, (2.7) and (2.9) are simplified as:

$$L_{i,j} = log \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} \exp\left(-\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2}\right) - log \sum_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} \exp\left(-\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2}\right) \tag{2.10}$$

and

$$L_{i,j} = \min_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} \left(\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2}\right) - \min_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} \left(\frac{\|\mathbf{y}-\mathbf{Hs}\|^2}{\sigma^2}\right) \tag{2.11}$$

One of the these two minimum term in (2.9) corresponds the ML estimation. We denote this minimum term as

$$\mu_{i,j}^{ML} = \frac{1}{\sigma^2}\|\mathbf{y}-\mathbf{Hs}^{ML}\|^2 - \log P(\mathbf{s}^{ML}) \tag{2.12}$$

The second term is given by:

$$\mu_{i,j}^{\overline{ML}} = \min_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=\overline{x_{i,j}^{ML}}\}} \left(\frac{1}{\sigma^2}\|\mathbf{y}-\mathbf{Hs}\|^2 - \log P(\mathbf{s})\right) \tag{2.13}$$

where $x_{i,j}^{ML}$ is the value of the jth bit position of ith symbol in the ML solution $\mathbf{s}^{ML}$. $\overline{x_{i,j}^{ML}}$ is the opposite component of $x_{i,j}^{ML}$, and it is defined as the *counter hypothesis*. Therefore (2.9) is given in a simpler form in (2.14):

$$L_{i,j} = \begin{cases} \mu_{i,j}^{\overline{ML}} - \mu_{i,j}^{ML} & x_{i,j}^{ML} = 1 \\ \mu_{i,j}^{ML} - \mu_{i,j}^{\overline{ML}} & x_{i,j}^{ML} = 0 \end{cases} \tag{2.14}$$

(2.14) indicates that to calculate the LLR we need firstly find its ML estimation which is the closest vector in the search space and then find the close vectors in the counter hypothesis set. The formula converts the computation of LLR to a problem of searching two minimum *overall distances*. The overall distance is the combination of the Euclidean distance between transmitted signal and received signal and a bias introduced by the a-priori information of transmitted signal, $P(\mathbf{s})$.

2.4 Sphere Decoder in MIMO detection

In the section 2.3, we have shown that the problem simplified to minimum distance searching. One trivial solution is to enumerate all possible vectors, such that the size of the search space is equal to 2^{mq} as well as the complexity of optimal ML decoder. The complexity increases exponentially, and is infeasible to implement.

In this section, we introduce a pre-processing step to enable a structured tree-search method to deal with (2.9). Otherwise (2.9) can only be solved by exhaustive searching.

2.4.1 QR decomposition as pre-processing stage

We write the channel matrix H in its QR decomposition:

$$\mathbf{H} = \mathbf{QR}$$

where $\mathbf{Q}$ is a $n \times m$ isometry matrix, and $\mathbf{R}$ is a upper triangular matrix. If we define

$$\mathbf{y}^* = \mathbf{Q}^H \mathbf{y}$$

$$\mathbf{n}^* = \mathbf{Q}^H \mathbf{n}$$

and multiple $\mathbf{Q}^H$ on both sides of (2.1), then we can obtain that:

$$\mathbf{y}^* = \mathbf{Rs} + \mathbf{n}^* \tag{2.15}$$

With the knowledge of stochastic theory, we can conclude that $\mathbf{n}^* \sim \mathcal{C}N(0, \sigma_0 \mathbf{I}_n)$, thus, we can apply (2.9) on this model:

$$L_{i,j} \approx \min_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=0\}} \left(\frac{\|\mathbf{y}^* - \mathbf{Rs}\|^2}{\sigma^2} - \sum_{i=1}^{m} \log P(s_i) \right) - \min_{\mathbf{s}\in\{\mathbf{s}|x_{i,j}=1\}} \left(\frac{\|\mathbf{y}^* - \mathbf{Rs}\|^2}{\sigma^2} - \sum_{i=1}^{m} \log P(s_i) \right) \tag{2.16}$$

Expand the term $\|\mathbf{y}^* - \mathbf{Rs}\|^2$, then we represent the 'distance' between $\mathbf{y}$ and $\mathbf{s}$ as:

$$\frac{1}{\sigma^2}\|\mathbf{y}^* - \mathbf{Rs}\|^2 - \log P(\mathbf{s}) = \sum_{i=1}^{m} \frac{1}{\sigma^2} |y_i^* - \sum_{j=i}^{m} R_{ij} s_j|^2 - \sum_{i=1}^{m} \log P(s_i) \tag{2.17}$$

We define the distance between the vector $\begin{bmatrix} y_i^* & y_{i+1}^* & \cdots & y_m^* \end{bmatrix}^T$ and $\begin{bmatrix} s_i & s_{i+1} & \cdots & s_m \end{bmatrix}^T$ as the *Partial Distance*(*PD*):

$$PD_i = \sum_{g=i}^{m} \frac{1}{\sigma^2} |y_g - \sum_{h=g}^{m} R_{gh} s_h|^2 - \sum_{g=i}^{m} \log P(s_g) \tag{2.18}$$

Note that when $i = 1$, the Partial Distance is equal to distance between received signal and transmitted signal:

$$PD_1 = \frac{1}{\sigma^2}\|\mathbf{y}^* - \mathbf{Rs}\|^2 - \log P(\mathbf{s})$$

When the apriori information is not involved, the *Partial Distance* coincides the concept of *Partial Euclidean Distance*, since (2.18) only calculates the Euclidean distance. The partial distance can be obtain by a recursive way shown in (2.19)

$$PD_i = PD_{i+1} + \left(\frac{1}{\sigma^2} |y_i - \sum_{j=i}^{m} R_{ij} s_j|^2 - \log P(s_i) \right) \tag{2.19}$$

The second term in the right is defined as *Distance Increment*(DI_i).Based on (2.19), we can compute the total Euclidean distance by following the steps:

1. Set $PD_{m+1} = 0$ and $i = m$
2. Calculate corresponding DI_i
3. Compute PD_i using (2.19)
4. Decrease i and go back to 2

This iteration ends when $i = 0$, and we get $PD_1 = \frac{1}{\sigma^2} \|\mathbf{y}^* - \mathbf{Rs}\|^2 - \log P(\mathbf{s})$

2.4.2 Tree construction

In this section, we present a depth-first tree searching to calculate the overall distance in (2.17). The forward traverse of tree search accompanies with a threshold shrinking mechanism. We only examine the vectors which have a distance less than the threshold. The limited search space enables us to solve (2.16) without exhaustive searching.

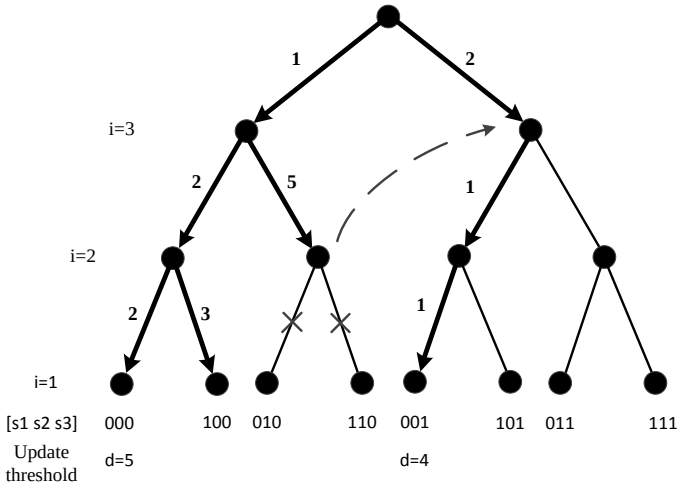


Figure 2.2: Depth-first tree search

Figure 2.2 takes BPSK as an example where there are three transmission antennas, and the corresponding transmitted vector is $[s_1\ s_2\ s_3]^T$. Each node except the root in this tree is

associated with one constellation point. Each layer corresponds to one spatial symbol. Thus symbol s_3 locates on the top layer, and the leaves are denoted with symbol s_1. The branches are labeled with the corresponding DI. For each node, its children are ordered by Schnorr-Euchner enumeration which arranges the QAM symbols on the same circle in a zig-zag way based on their locations to a certain start point[8, 4]. To solve (2.16), exhaustive method searches all branches, but we can use a modified depth-first tree search to reduce the search complexity. The depth-first tree search starts at root node while the partial distance and threshold are initialized as zero and infinity respectively. At each node, we visit the most preferred child which is determined by SE enumeration and update the partial distance by using (2.19). Once the partial distance exceeds the threshold, the branch is pruned. We backtrack to the upper layer and examine the adjacent child since all children are ordered in ascending order of their partial distances. When we reach one of the leaves, the threshold shrinks to the current partial distance and a backtrack happens. Therefore, the threshold continually reduces and shrinks the search space.

2.5 Summary

In this chapter, we presented the mathematical background in the sphere decoding. Especially, (2.9) is an important formula as it gives the LLR computation when apriori information is involved. Subsequently, in the later sections, we introduced the QR pre-processing step. This pre-processing finally enables us to solve the (2.9) through a depth-first tree search process.

3 Proposal for a Soft-in Soft-output Sphere Decoder algorithm

This chapter addresses the challenge of a 2×2 soft-input soft-output sphere decoder, in particular the difficulty of finding the best possible forward path when apriori information is involved. Section 3.1 demonstrates the overall algorithm that we follow, and in Section 3.2 we present the investigation of the problem given in Section 1.2.

3.1 Structure of the 2x2 Sphere Decoder

CONFIDENTIAL PARAGRAPHS

3.2 *REMOVED SECTION*

CONFIDENTIAL PARAGRAPHS.

3.3 Simulation results of Sphere Decoder

So far we addressed the structure of a 2×2 sphere decoder which can cooperate apriori information from the Turbo decoder. In order to limit the search space, we introduced an efficient pruning mechanism which can eliminate multiple candidates simultaneously.

In this section, we discuss the performed experiments and demonstrate the corresponding simulation results. Firstly we briefly explain the simulation environment and than present simulation results in different configurations and situations, and summarize the important lessons we have learned.

CONFIDENTIAL PARAGRAPHS

3.4 Abstraction of Sphere Decoder

CONFIDENTIAL PARAGRAPHS

3.5 Summary

CONFIDENTIAL PARAGRAPHS.

4 Simulation of Fixed-point Sphere Decoder

In Chapter 3, we have conducted several simulations to validate our algorithm. Floating-point data are used in those simulations. Although floating-point simulation gives us a result with high precision, fixed-point implementation is preferred due to the factor of component area and processing speed. It is crucial to investigate to what extent the precision-limited data can affect the performance of the algorithm and choose the most economical bitwidth for fixed-point implementation.

Simulation results of fixed-point sphere decoder algorithm are demonstrated in this chapter. Conclusions on internal fixed-point representation are given at the end of the chapter.

4.1 Truncated Sphere Decoder Algorithm

CONFIDENTIAL PARAGRAPHS.

4.2 Study of Quantization Steps

In order to investigate the quantization effects of different signals, the simulation results of quantization are presented in this section step by step. At the end of this section, a short conclusion will be given.

CONFIDENTIAL PARAGRAPHS.

4.3 Summary

CONFIDENTIAL PARAGRAPHS.

5 Implementation

So far we have discussed the sphere decoder algorithm, in this chapter, our topic is the implementation details of the sphere decoder. The design requirements are addressed at the very beginning of this chapter, and the implementation of each block is discussed explicitly.

5.1 Motivation – Why Hardware

There are many options to implement an algorithm. Sphere Decoder can be mapped to a dedicated hardware design or a software solution running on certain DSP systems. Before make a decision, it is crucial to analyse application requirements.

The sphere decoder kernel is designed for LTE application which supports QPSK, 16-QAM and 64-QAM modulations. The data rate is specified as 1200 carriers per OFDM symbol and 13 OFDM symbols per millisecond. Mapping sphere decoder kernel to a DSP system (for example EVP_{16} in [2]) causes extremely high memory access and arithmetic computation pressure.

CONFIDENTIAL PARAGRAPHS.

To conclude, the DSP based sphere decoder is an inefficient solution due to the frequent memory accesses and the huge computation loads. We must find a dedicate hardware solution of the sphere decoder.

5.2 Design Space Exploration

A certain algorithm can be mapped to various hardware architectures. Since the performances of different architectures are various in throughput, area, power and so on, analysis of design space becomes crucial. A good design space analysis provides a guideline of hardware design and help engineers make tradeoff decisions. The design space of the sphere decoder involves in parallelism exploration. The parallelism of sphere decoder exists in two scopes.

CONFIDENTIAL PARAGRAPHS.

The detail aspects of implementation of each block are discussed in the following sections. In Chapter 6, we map each block to different solutions which have different throughputs. The relationships between block area and throughput are presented explicitly by synthesis reports.

5.3 High Level Synthesis and Catapult C

Section 5.2 shows that the same algorithm can be mapped to different architecture solutions. Different solutions differ with each other in the performances of throughput, area and so on. In conventional design flow, engineers must code all possible solutions manually to explore the performance differences and make a proper tradeoff decision. However, exploration process accompanies with endless cross checking with original algorithm model and frustrating debugging.

In order to avoid inefficient manually coding and debugging, EDA tool vendors promote the development of High Level Synthesis (HLS) which is a automatic process that generates hardware that performs desired behaviour of the target algorithm. The synthesis process is referred as high-level since it begins with untimed behaviour model where no timing information exists. The goal of HLS tools is to release hardware engineers from debugging, let them concentrate on architecture exploration from higher level.After 20 years research, HLS tools are widely used nowadays. The Catapult C from Calypto Design Systems is chosen in our design. The source codes(C/C++) are analysed, constrained, scheduled to generate Register Transfer Level (RTL) model which is subsequently synthesised to gate level by a logic synthesis tool. Among those process steps, architecture constraints especially loop constraints play a crucial role for final results. This section presents two aspects of loop constraints known as the loop pipelining and unrolling, and their influences on the final architecture via a simple 4-element accumulator example.

5.3.1 Loop pipelining

Listing 5.1 describes a simple C++ accumulator. Although there is no explicit loop in the codes, a HLS model design always results in a implied loop as known as the *main loop*, shown in the Figure 5.1. The existence of implied loop lies on the fact that in a real hardware design, a hardware block is always activated.

Listing 5.1: Simple Accumulator

```
void accum(int &a, int &b, int &c, int &d, int &dout){

    int temp1, temp2;

    temp1 = a+b;
    temp2 = temp1+c;
    dout = temp2+d;
}
```

When the loop is unconstrained in default situation, the accumulator reads new input data after finishing whole processing loop. Figure 5.2(a) shows this default scheduling. In the default scheduling, each clock cycle (Ci in the figure) is assigned to one addition operation. The default design costs three cycles to accomplish the accumulation process. Clock cycle C4 is reserved to output the result. Therefore the default constrained design costs 4 cycles to finish one loop iteration. At the fifth clock cycle C5, the accumulator block read new data and starts a new accumulation process. Since only one addition is executed in every clock

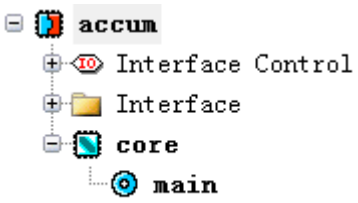
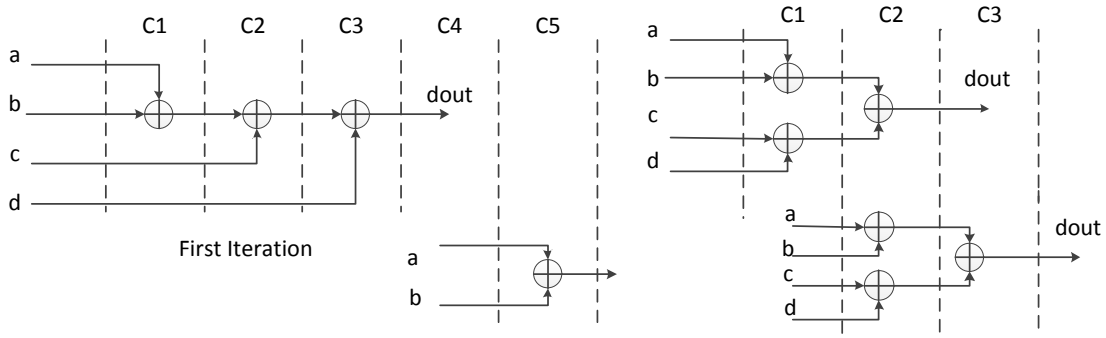


Figure 5.1: Implied Loop

cycle, one adder is shared during the whole processing. Figure 5.4(a) is the RTL schematic of default setting accumulator which is automatically generated by Catapult. There is only one adder in the final RTL result as the same as what we expect. The input of the adder is connected to a MUX which is responsible for selecting proper input data in different clock cycles.



(a) Default Design (b) Loop Pipelined Design, $II = 1$

Figure 5.2: The scheduling of *accum*

One the other hand, loop pipelining allows loop iterations to be overlapped. It is specified by the parameter *Initiation Interval (II)* (Figure 5.3). The Initiation Interval is how many clock cycles are taken before a new loop takes place, also determines the number of pipeline stages. Figure 5.2(b) demonstrates the schedule of a pipelined accumulator where the Initiation Interval is set as $II = 1$. Two addition operations are scheduled at the first clock cycle. At the second clock cycle, the accumulator processes the third addition, meanwhile a new accumulation process starts. New input data are read and two additions of the new loop are executed at clock cycle C2, since the Initiation Interval is specified as 1.

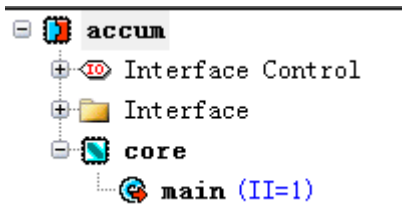
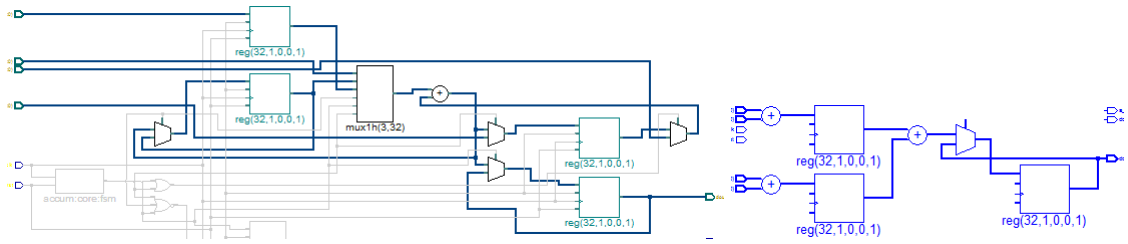


Figure 5.3: Loop Pipelining Example



(a) Default Design (b) Loop Pipelined Design, $II = 1$

Figure 5.4: RTL schematics of *accum*

Compared with the default scheduling, the pipelined design executes three additions simultaneously. This difference is reflected in the final RTL model where three adders are created by Catapult as shown in Figure 5.4(b). Moreover, the registers in Figure 5.4(b) consist a 2-stage pipeline.

5.3.2 Loop unrolling

Loop unrolling is another import aspect of loop constraint. The codes in Listing 5.2 describe a 4-element accumulator using an explicit ***for*** statement.

Listing 5.2: Left-rolled Accumulator

```
void accum2(int a[4], int &dout){
    int temp=0;
    ACCUM:for(int i=0; i<4; i++){
        temp = temp+a[i]
    }
    dout = temp;
}
```

As shown in Figure 5.5, the accumulator implies two loops. One is the implied main loop, the other one is the explicit loop–*ACCUM*. Loop unrolling has close relationship with resource allocation, it is usually associate with loop pipelining to achieve a desire throughput. We here only show the impact of inner loop unrolling, and the main loop is left-rolled.

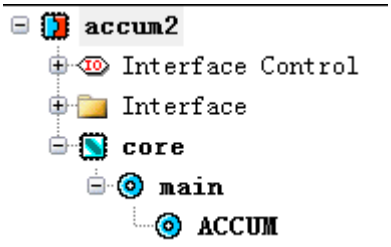
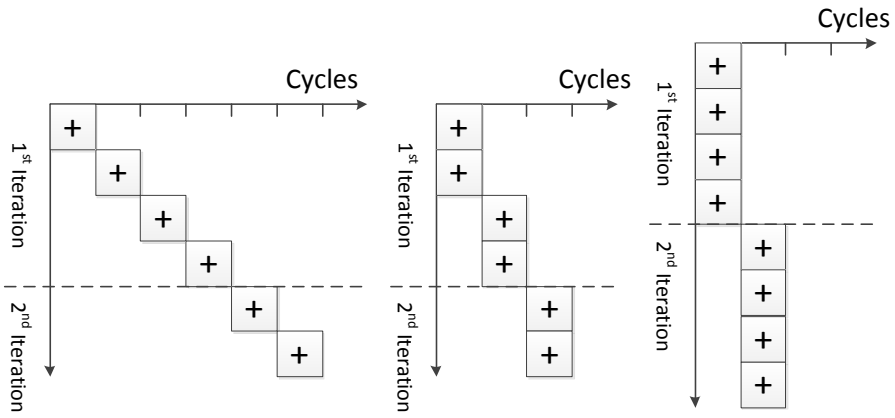


Figure 5.5: Main loop and explicit loop

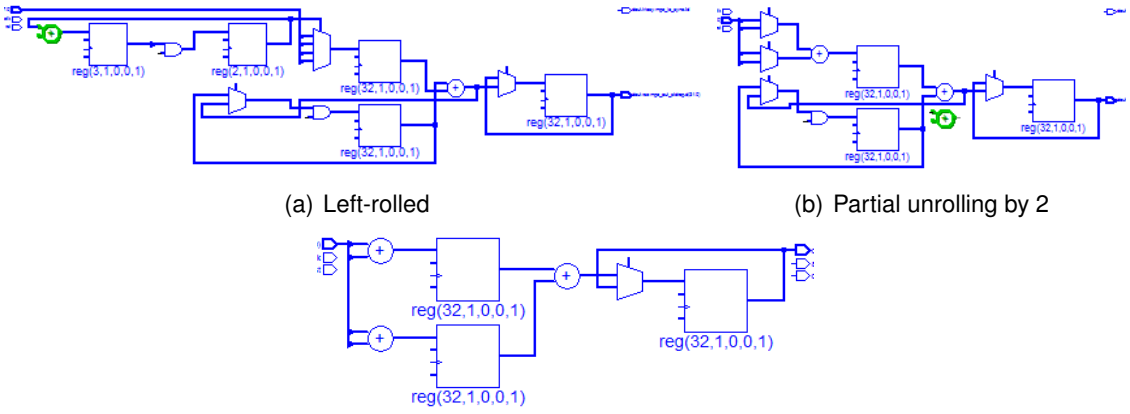
Figure 5.6(a) shows the scheduling of two main loop iterations while the *ACCUM* loop is left-rolled. Four additions are mapped into four cycles, such that only one adder is generated for



(a) Left-rolled (b) Partial unrolling by 2 (c) Fully unrolling

Figure 5.6: Scheduling of *accum2*

the accumulation (see Figure 5.7(a)). Notice that the green adder in the Figure 5.7(a) is loop control overhead implied by the statement (*for(. . . ,. . . ,**i++**)*). We partially unroll the inner loop, and the scheduling is shown in Figure 5.6(b). The first iteration costs two clock cycles. Two additions are executed in one clock cycles, which results in two adders in Figure 5.7(b). Notice that there is also an extra adder in Figure 5.7(b) (the green one) due to the loop control overhead. Figure 5.6(c) describes the scheduling of a fully unrolling design. Under the circumstance, all addition operations are scheduled in the same clock cycle and imply three adders in the final RTL model as shown in Figure 5.7(c). Since the inner loop is totally flattened, there is no control overhead in the RTL model.



(a) Left-rolled (b) Partial unrolling by 2

(c) Fully unrolling

Figure 5.7: RTL schematics of *accum2*

5.4 Implementation aspect of Euclidean Distance Computation

CONFIDENTIAL PARAGRAPHS.

5.5 Conversion between bit patterns and constellation coordinates

CONFIDENTIAL PARAGRAPHS.

5.6 Determining the Constellation Point

CONFIDENTIAL PARAGRAPHS.

5.7 Implementation aspect of Log-probability Computation

Clearly, Euclidean distance computation is not the whole story. The second term on the right of formula (2.18)

$$PD_i = \sum_{g=i}^{m} \frac{1}{\sigma^2} |y_g - \sum_{h=g}^{m} R_{gh} s_h|^2 - \sum_{g=i}^{m} \log P(s_g) \tag{5.1}$$

shows a requirement of log-probabilities of symbols. Assuming that the bit-pattern of a 16-QAM symbol is $b_3 b_2 b_1 b_0$, its log-probability is:

$$\log P(s) = \sum_{i=0}^{3} \log P(b_i)$$

CONFIDENTIAL PARAGRAPHS.

5.8 Design of Distance Table

CONFIDENTIAL PARAGRAPHS.

5.9 Implementation aspect of Softbits Calculation

Block *Softbis Calculation* is the last stage of the sphere decoder.

CONFIDENTIAL PARAGRAPHS.

6 Synthesis Results

As mentioned in Section 5.2 and Section 5.3, each hardware block in Figure **??** can be mapped to different solutions. In this chapter, the tradeoffs between area and initiate interval of each block are addressed. At the end of this chapter, we also compare the difference of the SISO sphere decoder with old sphere decoder [1] where the apriori information is not involved.

CONFIDENTIAL PARAGRAPHS.

The tools used in this section to constrain and synthesize our design are Catapult 7.0a and Design Compiler 2012.06. All the estimated area presented in this chapter are based on ST-Microelectronic's $28nm$ technology.

6.1 Euclidean Distance Computation

CONFIDENTIAL PARAGRAPHS.

6.2 *REMOVED SECTION*

CONFIDENTIAL PARAGRAPHS.

6.3 Apriori Distance Computation

CONFIDENTIAL PARAGRAPHS.

6.4 Update Distance Table

CONFIDENTIAL PARAGRAPHS.

6.5 Softbits Computation

CONFIDENTIAL PARAGRAPHS.

6.6 Search Thread

So far, all the sections of this chapter only present the synthesis result of an individual block. The total area of a single-branch *SearchThread* are illustrated.

CONFIDENTIAL PARAGRAPHS.

6.7 Comparison with Old Sphere Decoder

CONFIDENTIAL PARAGRAPHS.

6.8 Further Optimizations

In this section we identify the further optimizations in the design. They are:

1. *CONFIDENTIAL PARAGRAPHS*.
2. Bundling input data. One feature of the Catapult is that it creates synchronization logic for every input and output interface. Since each input data of the sphere decoder has its own interface, there exists lots of redundant synchronization logic in the current sphere decoder. The optimization method is to bundle associated input data together. For instance, the elements of received vector $\mathbf{y}$, y_1 and y_2, each of them has its own interface. However, since these two input are strongly associated, we can bundle them into a structure, such that these two input can share the same synchronization logic.
3. Recoding *Softbits Computation* block. As mentioned in Section 6.5, Catapult generate 12 subtractors to fulfill the scheduling. According to our analysis, two subtractors are enough to fulfill the throughput requirement. The Catapult scheduling is sensitive with coding style, therefore by dedicated coding it is possible to reduce the number of subtractors from 12 to 2.

7 Conclusions

To conclude this thesis, in the first chapter, we briefly review the previous works on MIMO detector and address the challenge of Turbo-equalization which requires that a decoder can cooperate with apriori information.

Chapter 2 discusses mathematic problem behind a soft-in soft-out sphere decoder. LLR computation formula (2.9) is addressed in Chapter 2. Note that to solve for the LLR in (2.9) exactly, exhaustive search is inevitable. Since exhaustive search leads to exponential complexity with the number of antennas, we need to approximate the LLR to achieve a low complexity solution. Later in this chapter, a depth-first tree searching is proposed to reduce the search space in (2.9) for an exact solution.

CONFIDENTIAL PARAGRAPHS.

After the proposal of sphere decoder algorithm, we focus on the implementation of a 2×2 SISO sphere decoder. Chapter 4 transforms the floating point algorithm to a fixed-point algorithm. *CONFIDENTIAL PARAGRAPHS*.

Chapter 5 addresses that a DSP based software implementation is impossible. We have to pursue a dedicate hardware solution. *CONFIDENTIAL PARAGRAPHS*.

Implementation detail of each block is discussed in Chapter 5. And the synthesis results are shown in Chapter 6. *CONFIDENTIAL PARAGRAPHS*.

Bibliography

[1] Özgün Paker, Sebastian Eckert, and Andreas Bury, "A Low Cost Multi-standard Near-Optimal Soft-output Sphere Decoder: Algorithm and Architecture", *Design, Automation & Test in Europe Conference & Exhibition*, 2010.

[2] Özgün Paker, Kees van Berkel, and Kees Moerman, "Hardware and Software Implementations of an MMSE equalizer for MOMO-OFDM based WLAN", *Signal Processing Systems Design and Implementation*, 2005.

[3] Emanuele Viterbo and Joseph Boutros, "A Universal Lattice Decoder for Fading Channels", *IEEE Transactions on Information Theory, VOL.45, NO.5*, July, 1999.

[4] Andreas Burg, Moritz Borgmann, Markus Wenk, Martin Zellweger, Wolfgang Fichtner, and Helmut Bolcskei, "VLSI Implementation of MIMO Detection Using the Sphere Decoding Algorithm", *IEEE Journal of Solid-State Circuits, VOL.40, No.7*, July, 2005.

[5] Ceoff knagge, Mark Bickerstaff, Brett Ninness, Steven R. Weller, and Graeme Woodward, "A VLSI 8×8 MIMO Near-ML Decoder Engine", *Signal Processing Systems Design and Implementation*, 2006.

[6] Zhan Guo and Peter Nilsson, "Algorithm and Implementation of the K-Best Sphere Decoding for MIMO Detection", *IEEE Journal on Selected Areas in Communications*, 2006.

[7] Bertrand M. Hochwald and Stephen ten Brink, "Achieving Near-Capacity on a Multiple-Antenna Channel", *IEEE Transactions on Communications, VOL.51, NO.3*, March, 2003.

[8] C.P. Schnorr and M. Euchner, "Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems", *Mathematical Programming*, 1994.

[9] E. Villebrun and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", *IEEE International Conference on Communications*, 1995.