Diss. ETH No 13970

Decision Feedback Equalization for Powerline and HIPERLAN

A dissertation submitted to the

SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Technical Sciences

presented by

Thomas Michael Sailer
Dipl. El.-Ing. ETH
born April 29th, 1971
citizen of Winterthur, ZH

accepted on the recommendation of

Prof. Dr. Gerhard Tröster, examiner
Prof. Dr. Hans-Andrea Loeliger, co-examiner
Dr.-Ing. Johannes Kneip, co-examiner

2001

Acknowledgement

I would like to express my gratitude to my advisor Prof. Dr. G. Tröster for giving me a very high degree of freedom in my research. I would like to thank my associate advisors Prof. Dr. H.-A. Loeliger and Dr.-Ing. J. Kneip for co-examining and for their valuable inputs to this thesis.

I would also like to thank James Aldis of Ascom Applicable Research & Technology for suggesting and collaborating in the KTI project that ultimately led to this dissertation. Ascom is now applying for a patent.

Last, but not least, I would like to thank Alexander Kurpiers of the Technical University Darmstadt for the fruitful discussions and for making me read [1].

Zürich, February 2001 Thomas Sailer

Contents

I Introduction 1

1. Trends in Communication Systems 3
- 1.1. Network Access Technologies . . . 3
- 1.2. Receiver Computational Demand . . . 4
  - 1.2.1. Orthogonal Frequency Division Multiplex . . . 5
  - 1.2.2. Serial Tone Modulation . . . 5
- 1.3. Receiver Decision Strategies . . . 5
- 1.4. Outline of the Thesis . . . 8

II Algorithm 11

2. Decision Feedback Equalization 13
- 2.1. The System Model . . . 14
- 2.2. The Stochastic Channel Model . . . 16
  - 2.2.1. Truncating the Channel Impulse Response . . . 16
  - 2.2.2. Channel Model Parameters for Power Line Communications . . . 16
  - 2.2.3. Channel Model Parameters for the Indoor Radio Channel/HIPERLAN . . . 17
- 2.3. The Decision Feedback Equalizer . . . 17
  - 2.3.1. DFE Key Equations . . . 20
  - 2.3.2. Computing the Feedback Filter Coefficients and the Bias . . . 26
  - 2.3.3. Choosing N_f . . . 27
  - 2.3.4. Choosing N_b . . . 31
  - 2.3.5. The Optimal Equalizer for 1-Dimensional Signal Constellations . . . 31

3. DFE Matrix Factorization 37
- 3.1. Problem Statement . . . 38
- 3.2. Direct Methods . . . 38
  - 3.2.1. Generic LU Factorization . . . 38
  - 3.2.2. QR Factorization . . . 38
  - 3.2.3. Cholesky Factorization . . . 39

3.2.4. Displacement Structure Theory 40
3.2.5. Avoiding the Back Substitution 43
3.2.6. Bounds for the Diagonal of the Cholesky Factor **L** . 46
3.3. Iterative Methods 48
3.3.1. Gauss-Seidel 48
3.4. DFE Solution Algorithm Comparison 48
3.4.1. Cholesky Factorization 50
3.4.2. Displacement Structure Algorithm with Back Substitution 50
3.4.3. Displacement Structure Algorithm without Back Substitution 54
3.4.4. Gauss-Seidel Iteration 55
3.4.5. Discussion 55

III Implementation **59**

4. Fast VLSI Architectures for the Displacement Structure Algorithms **61**
4.1. State of the Art 62
4.1.1. Systolic Arrays for Cholesky Factorization 62
4.1.2. Systolic Arrays for **QR** Factorization 62
4.1.3. Linear Equalizers 63
4.1.4. Computing the Feedback Coefficients using **QR** Decomposition 64
4.1.5. Summary 64
4.2. General Architecture 64
4.2.1. Reduced Hardware Complexity 67
4.3. The Processing Elements 68
4.3.1. Working with Real Numbers 68
4.3.2. Working with Complex Numbers 69
4.3.3. The CORDIC Blocks 70
4.3.4. Processing Element Examples 73
4.3.5. Reducing Hardware Complexity further 74
4.3.6. Speeding up the Computation 77
4.4. Architecture for HIPERLAN 77

5. FPGA DSP Core **83**
5.1. Motivation 83
5.2. Designing a DSP Core for FPGA 84

5.2.1. Execution Unit 85
5.2.2. Instruction Decoder 88
5.2.3. Development Tools 91
5.2.4. Reciprocal Square Root 91
5.3. Results 94
5.3.1. Comparison to other FPGA Processors 94
5.3.2. Comparison of Different DFE Coefficient Computation Algorithms 95
5.3.3. Comparison to Dedicated DFE Coefficient Computation FPGA Hardware 99

6. Outlook **103**
6.1. OFDM Systems over Higly Dispersive Channels 103
6.2. Multiuser Detection for Wideband CDMA 104
6.3. Continuous Systems 104

A. Utilized Parameters and their Symbols **105**

B. Abbreviations **109**

List of Figures

2.1 System model 15
2.2 Decision feedback equalizer 18
2.3 Computation of the feedback filter coefficients and the bias . 27
2.4 Decision point error energy vs. N_f for $N_0 = -5dB$ and 2-dim. eq. ($N_D = 1, N_O = 1, d_i \in \mathbb{C}$) 28
2.5 Decision point error energy vs. N_f for $N_0 = -10dB$ and 2-dim. eq. ($N_D = 1, N_O = 1, d_i \in \mathbb{C}$) 28
2.6 Decision point error energy vs. N_f for $N_0 = -20dB$ and 2-dim. eq. ($N_D = 1, N_O = 1, d_i \in \mathbb{C}$) 29
2.7 Decision point error energy vs. N_f for $N_0 = -5dB$ and 1-dim. eq. ($N_D = 1, N_O = 2, d_i \in \mathbb{R}$) 29
2.8 Decision point error energy vs. N_f for $N_0 = -10dB$ and 1-dim. eq. ($N_D = 1, N_O = 2, d_i \in \mathbb{R}$) 30
2.9 Decision point error energy vs. N_f for $N_0 = -20dB$ and 1-dim. eq. ($N_D = 1, N_O = 2, d_i \in \mathbb{R}$) 30
2.10 Decision point error energy vs. N_b for $N_0 = -20dB$ and 2-dim. eq. ($N_D = 1, N_O = 1, d_i \in \mathbb{C}$) 32
2.11 Decision point error energy vs. N_b for $N_0 = -20dB$ and 1-dim. eq. ($N_D = 1, N_O = 2, d_i \in \mathbb{R}$) 32
2.12 Decision feedback equalizer for $N_D = 2$ 33
2.13 Decision feedback equalizer for 1-dimensional constellation, $N_D = 1$ 33

3.1 Decision point error energy vs. number of iterations for $N_0 = -10dB$ and 2-dim. eq. ($N_D = 1, N_O = 1, d_i \in \mathbb{C}$) 49
3.2 Decision point error energy vs. number of iterations for $N_0 = -10dB$ and 1-dim. eq. ($N_D = 1, N_O = 2, d_i \in \mathbb{R}$) 49
3.3 Number of multiplications of 2-d equalizer algorithms . . . 56
3.4 Storage requirements of 2-d equalizer algorithms 56
3.5 Number of multiplications of 1-d equalizer algorithms . . . 57
3.6 Storage requirements of 1-d equalizer algorithms 57

4.1 CORDIC based systolic array for **QR** factorization 63
4.2 Architecture for displacement structure factorization 65
4.3 Architecture for displacement structure solution 66
4.4 Modified architecture for displacement structure solution . . 67

4.5 Recycling processing elements 68
4.6 CORDIC blocks 71
4.7 CORDIC microrotation slice 72
4.8 Processing element for $N_D = 1$, $N_O = 2$, real numbers, white noise 73
4.9 Processing element for $N_D = 1$, $N_O = 2$, real numbers, coloured noise 74
4.10 Processing element for $N_D = 1$, $N_O = 1$, complex numbers, white noise 75
4.11 Processing element for $N_D = 1$, $N_O = 1$, complex numbers, coloured noise 75
4.12 Detailed diagram of the processing element for $N_D = 1$, $N_O = 1$, complex numbers, white noise 76
4.13 Simplified processing element for $N_D = 1$, $N_O = 2$, real numbers, white noise, two passes per iteration 77
4.14 Computer simulation of residual ISI vs. number of microrotations 78
4.15 Computer simulation of residual ISI vs. wordlength 79
4.16 Minimum Latency Architecture for DFE coefficient computation 80
4.17 Single PE Architecture for DFE coefficient computation . . 80

5.1 Simplified Xilinx Virtex CLB slice 85
5.2 DSP core block diagram 86
5.3 DSP execution unit architecture 89
5.4 DSP core pipeline 90
5.5 Reciprocal square root seed table 92
5.6 Reciprocal square root, below 1 92
5.7 Reciprocal square root, below 0.2 93
5.8 Execution time (number of cycles) for different algorithms on FPGA DSP core 97
5.9 Dedicated DFE computation hardware 100
5.10 FPGA die plot of DSP core 101
5.11 FPGA die plot of dedicated hardware 102

List of Tables

3.1 Real operations required for cholesky factorization 51
3.2 Real operations required for Θ computation 52
3.3 Real operations required for displacement structure factorization 53
3.4 Real operations required for displacement structure solution 54
3.5 Real operations required for Gauss-Seidel iteration 55

4.1 Hardware architecture comparison for 2-d equalizer coefficient computation 64
4.2 Summary of processing elements 79
4.3 AMS 0.35μm standard cell library data 81
4.4 Area and power consumption of datapath components using the AMS 0.35μm standard cell process 82
4.5 Summary of three HIPERLAN coefficient computation architectures 82

5.1 FPGA DSP core implementation results 95
5.2 FPGA RISC/DSP cores 95
5.3 Code size 96
5.4 Execution time (number of cycles) of cholesky factorization 96
5.5 Execution time (number of cycles) of displacement structure factorization 96
5.6 Execution time (number of cycles) of displacement structure solution 97
5.7 Number of datapath multiplications versus number of clocks 98
5.8 FPGA DSP core versus dedicated FPGA hardware 99
5.9 FPGA DSP clock cycles and time versus dedicated hardware architecture 100

Abstract

High-speed packet based communication systems such as HIPERLAN or Powerline Communications (PLC) are increasingly popular. The channels these systems operate on introduce severe Intersymbol Interference (ISI), which has to be mitigated. The Decision Feedback Equalizer (DFE) is an attractive method for high speed systems, as it performs well at moderate implementation cost. The DFE is most often implemented using transversal Finite Impluse Response (FIR) filters. The DFE consists of a Feedforward Filter (FFF) and a Feedback Filter (FBF).

Packet based systems usually employ a preamble for synchronisation purposes and to estimate the channel impulse response (CIR). The minimum mean square error criterion leads to a system of linear equations, the DFE key equations, for computing the optimal DFE filter coefficients from the CIR. While the DFE itself is easy to map onto parallel hardware, the fast computation of the filter coefficients is more difficult. Because it is part of the critical path of packet decoding, quick computation of the coefficients is essential. Therefore this contribution focuses on the efficient computation of the DFE coefficients.

A novel algorithm based on Displacement Structure Theory well suited to VLSI implementation is developed. A hardware architecture implementing the algorithm is proposed. It consists of a linear chain of processing elements. The processing elements mainly consist of CORDIC blocks. The architecture has desirable properties for VLSI technology, namely local communication only and a highly regular and aggressively pipelineable datapath, making fast clock frequencies possible. The proposed architecture computes the FFF coefficients of a 12 tap Decision Feedback Equalizer suitable for HIPERLAN I in 221 clock cycles using an area of 1.4mm^2 on a 0.35μm standard cell process and consuming 1.5μJ per computation. In contrast, the so called QR factorization previously proposed in literature for the same problem requires approximately 576 clock cycles, an area of 30.90mm^2 and 68μJ per computation.

For systems that can tolerate longer packet decode latency but require more flexibility, such as Powerline communication (PLC) systems, a Digital Signal Processor is a suitable platform for computing the equalizer coefficients. In order to make real-time prototyping possible, a high speed DSP core that can be implemented on Field Programmable Gate Arrays (FPGAs)

has been developed. The DSP core can operate at up to 80 MHz/80 MIPS on a Xilinx Virtex XCV400-6 device, uses approximately 100k gate equivalents and features a single cycle throughput multiplier/accumulator and a 16bit datapath. It outperforms all competing commercial designs known to the author by more than 50%. It can compute the FFF coefficients of a 10 tap symbol spaced single output real equalizer in 9923 clock cycles. A dedicated hardware architecture requires 692 clock cycles and 24k gates for the same problem.

Kurzfassung

Paketbasierte schnelle Kommunikationssysteme wie HIPERLAN oder Powerline Modems (PLC) werden immer populärer. Die Kanäle, über die diese Systeme kommunizieren, verursachen starke Intersymbolinterferenz. Der entscheidungsrückgekoppelte Entzerrer (Decision Feedback Equalizer, DFE) ist eine attraktive Methode, die Intersymbolinterferenz zu kompensieren. Der DFE wird meist mittels Transversalfiltern realisiert. Er besteht aus einem Vorwärts- und einem Rückkoppelungsfilter.

Paketbasierte Systeme verwenden oft eine Präambel, welche zur Synchronisation und zur Schätzung der Kanalimpulsantwort (CIR) verwendet wird. Die Verwendung des minimalen Fehlerquadrat-Kriteriums führt zu einem System linearer Gleichungen, mit denen die optimalen DFE Filterkoeffizienten aus der Kanalimpulsantwort berechnet werden können. Der DFE selber kann einfach parallelisiert werden, die schnelle Berechnung der Koeffizienten jedoch ist schwieriger. Weil die Koeffizientenberechnung im kritischen Pfad des Paketempfangs liegt, ist eine schnelle Berechnung dieser Koeffizienten essenziell. Der Schwerpunkt der vorliegenden Arbeit liegt daher in der Koeffizientenberechnung.

Ein neuer Algorithmus basierend auf Displacement Structure Theory wurde entwickelt, und eine dazu passende Hardwarearchitektur wird vorgeschlagen. Die Architektur besteht aus einer linearen Kette von Prozessorelementen. Die Prozessorelemente selber enthalten vor allem CORDIC-Blöcke. Diese Architektur hat vorteilhafte Eigenschaften für VLSI Technologieen, insbesondere ausschliesslich lokale Kommunikation, ein sehr regelmässiger Datenpfad, der aggressives Pipelining ermöglicht. Hohe Taktraten sind so möglich. Zur Berechnung der Vorwärtsfilterkoeffizienten eines für HIPERLAN geeigneten 12 Tap DFE (DFE der Ordnung 12) benötigt diese Architektur 221 Taktzyklen, 1.4mm^2 Siliziumfläche und 1.5μJ Energie pro Berechnung auf einem 0.35μm Standardzellenprozess. Im Gegensatz dazu benötigt die in der Literatur vorgeschlagene sogenannte QR Faktorisierung für dasselbe Problem 576 Taktzyklen, 30.90mm^2 Fläche und 68μJ Energie pro Berechnung.

Ein schneller DSP ist für Systeme wie zum Beispiel die Kommunikation über Stromleitungen (Powerline Communication, PLC), welche grössere Paketempfangslatenzzeiten tolerieren können, aber grössere Flexibilität benötigen, die geeignete Plattform. Um Echtzeit-Prototypen zu

ermöglichen, wurde ein schneller DSP-Kern entwickelt, der auf einem feldprogrammierbaren Logikbaustein (FPGA) implementiert werden kann. Der DSP-Kern arbeitet auf einem Xilinx Virtex XCV400-6 mit bis zu 80 MHz/80 MIPS und benötigt ungefähr 100k Gatteräquivalente. Der Datenpfad ist 16 Bit breit, und der Durchsatz beträgt eine Multiplikation/Akkumulation pro Taktzyklus. Der DSP-Kern übertrifft die Geschwindigkeit aller dem Autor bekannten konkurrierender kommerzieller FPGA-Prozessoren um mehr als 50%. Er kann die Vorwärtsfilterkoeffizienten eines 10 tap Entzerrers für reelle Werte in 9923 Taktzyklen berechnen. Eine dedizierte Hardwarearchitektur für dasselbe Problem benötigt 692 Taktzyklen und 24k Gatteräquivalente.

Part I

Introduction

1

Trends in Communication Systems

1.1. Network Access Technologies

The recent liberalisation of the European telecommunications market has spurred heavy competition on the long distance market, that made the long distance communication costs plummet. In the local access market, however, especially residential and small business users still face a defacto monopoly of the former state monopoly telecommunication companies [2]. The main reason for this situation is that with the currently prevalent access technology, the wired copper local loop, incumbent telecom operators would have to face huge investments in money and time to match the former state monopolies existing copper plant.

Therefore, it is attractive to search for alternative "last mile" access technologies. Three technologies are currently promising:

1. Every home or office has a power outlet. It is therefore tempting to use the power wiring not only for transferring electrical energy but also for communication. This is called powerline communication (PLC).

2. Recently, huge bandwidths in the microwave bands have been auctioned off in several European countries to be used as wireless local loop (WLL). Futhermore, recent studies [3] have shown that the microwave outdoor channel is similar to the microwave indoor channel when directive antennas are employed. It is therefore tempting to use technologies introduced for indoor communications also for WLL.

3. Since most homes are already connected to the cable television (CATV) network, it is also attractive to use the CATV infrastructure to provide local access.

To achieve the data rate users expect from contemporary systems, terminals must employ computationally demanding algorithms in their receiving section.

1.2. Receiver Computational Demand

> Shannon asks for more than Moore can deliver
>
> — Heinrich Meyr
>
> (the inofficial title to his International Zurich Seminar on Broadband Communications IZS 2000 presentation [4])

This quote illustrates that the complexity of communication systems is growing faster than the process technology is improving. This gap needs to be filled with algorithm simplification and innovative VLSI architectures, which is the subject of this contribution and of [5].

The goal of this section is to underline the need for efficient algorithms to implement the receivers tasks.

Both the powerline and the wireless channel are significantly dispersive at the transmission rates needed for broadband access. Therefore, high computational demand is placed on the receiver to combat the Intersymbol Interference (ISI).

Two techniques exist to combat the ISI, namely OFDM and serial tone modulation with equalization.

1.2.1. Orthogonal Frequency Division Multiplex

Orthogonal Frequency Division Multiplex (OFDM) divides the transmission bandwidth into many subchannels with a small bandwidth that are ideally orthogonal. The Fast Fourier Transform (FFT) is often used to perform the orthogonal decomposition. The frequency response of the channel then becomes approximately flat over the small bandwith of one subchannel. The time dispersion of the channel is mitigated by inserting a guard interval between subsequent modulation symbols.

Since the fading of the channel is flat over the bandwidth of a subchannel, all the receiver has to do is to estimate and compensate for the amplitude and phase variance introduced by the channel for every subchannel.

With OFDM, it is easy to exclude frequency bands because of regulatory or interference issues.

The OFDM signal, however, has a high peak to average power ratio (PAPR) and therefore requires a highly linear transmitter. This directly translates into expensive transistors, especially for microwave radio transmitters, and a low power efficiency of the transmitter. Furthermore, since noise pulses are spread in time over the duration of one subchannel symbol and over all subchannels by the FFT, OFDM is less desirable for channels with impulsive noise.

1.2.2. Serial Tone Modulation

Unlike OFDM, serial tone modulation uses short symbols that occupy the full channel bandwidth and encode only few bits. The distortion seen by these symbols is therefore frequency selective. It is usually modelled by a linear finite impulse response filter. The receiver now has to estimate all FIR filter taps and compensate for the distortion introduced by this filter.

Serial tone modulation methods exist that exhibit a constant envelope. These signals allow the use of a highly nonlinear C-class power amplifier.

In the next section, receiver strategies for coping with frequency selective fading are reviewed.

1.3. Receiver Decision Strategies

It is the task of the receiver to decide, upon the received signal, what was the most likely message sent by the transmitter. The signal is corrupted by the

nonflat channel response and by additive noise.

$$\hat{d}_{\mathrm{MAP}} := \arg\max_{d} P(d|r) \tag{1.1}$$

gives the optimal decision rule, called Maximum Aposteriory Probability (MAP). d is the transmitted message, $\hat{d}$ is the receivers decision about the transmitted message, and r is the received signal. There are usually additional unwanted parameters, unwanted in the sense that the receiver is not interested in their values, however they must be estimated to decode the message. Examples include the channel impulse response, carrier frequency offset, symbol timing, etc.

$$\hat{d}_{\mathrm{MAP}} := \arg\max_{d} \int_{e} P(d|r,e) \tag{1.2}$$

is the MAP decision rule with unwanted parameters. If all possible messages are equally likely, the MAP rule reduces to the Maximum Likelihood (ML) rule

$$\hat{d}_{\mathrm{ML}} := \arg\max_{d} \int_{e} P(r|d,e). \tag{1.3}$$

Clearly, the ML decision rule is complicated a lot by the unwanted parameters. An often used strategy to deal with the unwanted parameters is for the transmitter to send sequences of known training symbols, called pre-, post- or midambles, which allow the receiver to separate the unwanted parameter estimation problem from the message decision problem. However, even the simplified ML decision rule

$$\hat{d}_{\mathrm{ML}} := \arg\max_{d} P(r|d,\hat{e}) \tag{1.4}$$

is still too complex for direct implementation. The direct implementation complexity grows exponentially with the message size.

The optimal detection strategy for uncoded transmission and finite channel impulse response length in accordance with (1.4) is called Maximum Likelihood Sequence Detection (MLSD) [6]. The best known algorithm is called the Viterbi Equalizer (VE) and its complexity is proportional to the message length and exponential to the channel length. For moderate channel lengths, the Viterbi Equalizer is expensive to implement. Reducing the VE complexity by truncating the channel impulse response does not work well; performance degrades rapidly. A number of suboptimal algorithms also exist that only keep the M most likely surviving paths through the trellis instead of all surviving paths. They are seldom used in practice, because the need to sort

the surviving paths at every symbol to select the most likely ones outweighs the computational gain of not having to process all of them [7].

For coded interleaved transmission, the combined coding/interleaving/channel trellis gets very large, therefore optimal detection using a Viterbi Equalizer becomes unfeasible again. A suboptimal scheme is to use an equalizer that outputs probabilities instead of final decisions. These probabilities are then deinterleaved and fed to the decoder. The decoder also outputs probabilities, which are interleaved again and fed back to the equalizer. This cycle is repeated several times. This scheme is called "Turbo Detection" and has received a lot of interest recently, as it has been demonstrated that the Turbo transmission scheme can operate near the Shannon capacity bound. Two algorithms are used to compute the probabilities. One is an enhancement to the Viterbi Algorithm (VA), which is called Soft Output Viterbi Algorithm (SOVA) [8]. The SOVA enhancement is based on the assumption that the most likely error dominates all other errors. This is not always the case, therefore SOVA often underestimates the probability of error. The optimal algorithm is called the BCJR [9] algorithm. It requires two passes through the trellis, one in forward direction and one in backward direction, therefore it is often also called the Forward/Backward Algorithm. Its complexity is roughly twice that of the SOVA algorithm. The complexity of the Turbo scheme is still prohibitively high for high speed systems, confining the Turbo scheme to relatively low speed deep space and HF communication systems.

The simplest decision rule is the symbol by symbol rule

$$\hat{d}_{i,\mathrm{ML}} := \arg\max_{d_i} P(r_i|d_i, \hat{e}). \tag{1.5}$$

Because it treats the intersymbol interference (ISI) introduced by the channel impulse response (CIR) as noise, it performs badly even on lightly spread channels.

A popular method to improve the performance of the symbol by symbol rule is to insert an FIR filter between the received signal and the decision device, and to subtract the influence of the previously decided symbols. This is called the Decision Feedback Equalizer (DFE). Its complexity is linear with respect to the message length and only linear with respect to the length of the channel impulse response. The DFE is an attractive trade-off between computational complexity and performance for high speed systems.

The difficulty with the DFE is the computation of the optimal filter coefficients from the channel impulse response estimate. This computation is usually in the critical packet decoding path. Therefore, this contribution focuses on algorithms and VLSI architectures for solving this problem at low latency and area cost. In [10], the authors conjectured that the DFE might

be too complex for HIPERLAN. It is a goal of this contribution to prove the opposite by construction.

1.4. Outline of the Thesis

In chapter 2, the system model and the stochastic channel model used throughout the thesis are presented. The Decision Feedback Equalizer key equations are derived in the most general Multiple Input Multiple Output (MIMO) form, and guidelines for choosing the number of feedforward and feedback taps are given. Furthermore, the optimal Decision Feedback Equalizer for a real constellation is compared to the suboptimal approach of using a complex DFE for a real constellation, both in terms of complexity and signal to error energy loss.

Chapter 3 reviews algorithms for solving the DFE key equations. Emphasis is placed on the Cholesky factorization, Displacement Structure Factorization and Displacement Structure Solution. Cholesky factorization is the traditional method for computing the DFE coefficients. Its complexity is $O(n^3)$. Displacement Structure Factorization employs the Displacement Structure Theory framework to use the inherent structure in the DFE equations to reduce the number of operations for computing the Cholesky factors to $O(n^2)$. Displacement Structure Solution is a novel algorithm for directly computing the DFE feedforward coefficients without the need for back substitution. Its virtue is the high regularity of the computation and dataflow, albeit at the expense of an increased number of operations compared to Displacement Structure Factorization. Furthermore, the exact number of additions, multiplications and other operations are given for the different algorithms for the practically important cases of the symbol spaced equalizers with a single output for real and complex constellations.

Chapter 4 discusses fast VLSI architectures and implementation issues of the Displacement Structure Solution algorithm. To illustrate the power of the proposed family of architectures, a case study concerning an equalizer suitable for HIPERLAN compares two circuits employing the proposed architecture to an architecture previously proposed in literature.

Chapter 5 discusses the design and the implementation of a high performance Digital Signal Processor (DSP) implemented on a Field Programmable Gate Array (FPGA). The design outperforms all commercial FPGA DSP and RISC processors known to the author. The three aforementioned DFE coefficient computation algorithms have been implemented on this FPGA DSP core, and the results are discussed. Finally, the FPGA DSP core is compared to dedicated hardware for computing the DFE coefficients.

Chapter 6 suggests possible extension of this work.

Part II

Algorithm

2

Decision Feedback Equalization

Section 2.1 introduces the stochastic system model that will be used throughout this work. Section 2.2 describes the channel model that is used in the simulations, along with the parameters for the Power Line Communications channel and the 5 GHz HIPERLAN indoor channel.

Section 2.3 introduces the Decision Feedback Equalizer. Equations for the optimum DFE filter coefficients are derived using the Minimum Mean Square Error criterion.

Sections 2.3.3 and 2.3.4 provide guidelines for choosing the number of feedforward and feedback taps.

Section 2.3.5 compares the optimal Decision Feedback Equalizer for a real constellation to the suboptimal approach of using a complex DFE for a real constellation, both in terms of complexity and signal to error energy loss.

Notation The following notational conventions are used throughout this dissertation. i denotes the discrete time index. Lowercase (uppercase) bold symbols denote column vectors (matrices). $(\cdot)^*$ denotes elementwise com-

plex conjugation, $(\cdot)^T$ transposition, and $(\cdot)^H$ hermitian transposition. $E[]$ denotes the expectation operator. $(\cdot)^{\Re}$ and $(\cdot)^{\Im}$ denote the real part and the imaginary part of a complex number, respectively. $:=$ denotes definition. $x_i^{(j)}$ denotes the j-th element of the column vector $\mathbf{x}_i$. $X_i^{(j,k)}$ denotes the j,k-th element of the matrix $\mathbf{X}_i$. Indices (j and k) start at zero. Most of the time, j runs from 0 to $N_O - 1$ and k runs from 0 to $N_D - 1$. N_D denotes the number of symbols the transmitter generates per time step and N_O denotes the number of channel outputs per time step.

2.1. The System Model

Figure 2.1 shows the system model. For the sake of generality, multiple input multiple output channels are considered. Fractionally spaced ($\frac{N_D}{N_O T}$) equalizers as well as equalizers for one dimensional constellations may be expressed in this framework. The multiple input multiple output DFE may also be useful for asynchronous CDMA systems, where the signature waveforms of the different users are not orthogonal, or for OFDM systems which use a guard period that is shorter than the channel spread, and therefore intersymbol interference and interchannel interference occurs. The transmitter generates N_D symbols $\mathbf{d}_i$ at every time step i. The symbol is transmitted through the channel $\mathbf{C}_i$. For notational convenience, the i-th taps of all subchannels are collected into the $N_O \times N_D$ matrix $\mathbf{C}_i$. $\mathbf{C}_i^{(j,k)}$ denotes the i-th tap of subchannel j,k.

The complex baseband representation of the channel is used. The channel considered here is the convolution of the transmitter filter and the radio wave propagation environment as seen by the receiver.

The channel is modelled as a linear time invariant system. This assumption only holds over a relatively short period of time, i.e. one single packet. Therefore, the channel impulse response has to be estimated for every packet.

Stationary Gaussian noise is added to the output of the channel. Most of the time the noise is assumed to be white, but the algorithms described in this thesis also work for coloured noise.

$$\mathbf{r}_i := \sum_{j=-\infty}^{\infty} \mathbf{C}_j \mathbf{d}_{i-j} + \mathbf{n}_i = \sum_{j=-\infty}^{\infty} \mathbf{C}_{i-j} \mathbf{d}_j + \mathbf{n}_i \tag{2.1}$$

gives the channel output. $\mathbf{r}_i$ are the channel outputs, $\mathbf{d}_i$ the transmitted symbols, $\mathbf{C}_i$ the channel impulse response (CIR) and $\mathbf{n}_i$ the noise samples.

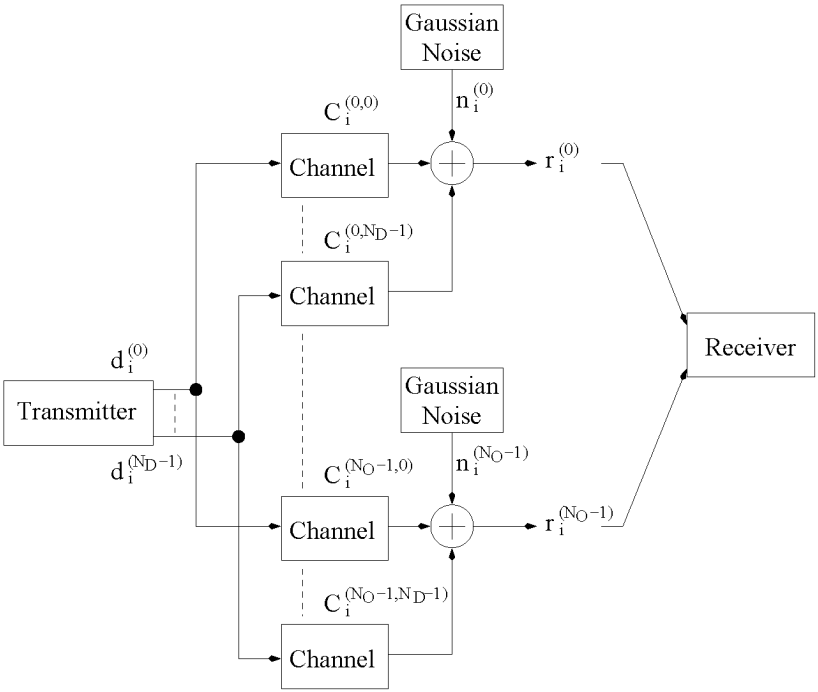

Figure 2.1: System model
The block labelled "receiver" contains the decision feedback equalizer (figure 2.2)

2.2. The Stochastic Channel Model

Whereever simulation results are presented, the channel model of this section is used for the individual subchannels ([11, Section 1.3] and the references therein).

$$c_i := \begin{cases} 0 & i < 0 \\ (c_i^{\Re} + j c_i^{\Im}) \sqrt{\frac{1}{2}(1 - e^{-\frac{1}{\tau}})} e^{-\frac{i}{2\tau}} & i \geq 0 \end{cases} \tag{2.2}$$

shows the channel model. $c_i^{\Re}$ and $c_i^{\Im}$ are independent gaussian random variables with zero mean and unit variance, τ is the delay spread normalized to the symbol rate, and the square root term normalizes the CIR to unit energy on average.

Channel models with an exponentially decaying delay profile have been proposed by Jakes [12], COST 207, ETSI and IEEE 802, among others.

2.2.1. Truncating the Channel Impulse Response

It is computationally advantageous to truncate the channel impulse response for $i \geq N_f$. To justify the truncation, the average CIR energy lost is given by

$$E_{cl} := \sum_{i=N_f}^{\infty} E[c_i^* c_i] = (1 - e^{-\frac{1}{\tau}}) e^{-\frac{N_f}{\tau}} \frac{1}{1 - e^{-\frac{1}{\tau}}} = e^{-\frac{N_f}{\tau}}. \tag{2.3}$$

Therefore if $N_f \gtrsim 7\tau$, the energy lost by the truncation is less than $-30dB$. The 99.9% energy rule is applied widely, for example in [13] or COST 207.

2.2.2. Channel Model Parameters for Power Line Communications

Power cables were not made for transmitting high frequency signals. The attenuation of the cables limit the frequency range usable for outdoor communication to about 10 MHz [14]. Furthermore, the shielding of the cables is imperfect. In order to protect colocated services like Shortwave Broadcast and Amateur Radio from mutual interference with Powerline Communications (PLC), PLC may not use the frequency bands allocated to the former services [15, 16]. This limits the contiguous bandwidth to $\approx 1.5\mathrm{MHz}$, which in turn limits the maximum symbol rate to $\approx 2\mathrm{MSymbols/s}$.

From the measurements in [14], a delay spread normalized to the symbol rate of about $\tau = 0.6$ can be expected.

2.2.3. Channel Model Parameters for the Indoor Radio Channel/HIPERLAN

Various propagation measurements for the indoor radio channel have been performed [17, 18]. If the base station is in an aisle and the terminals are in the same aisle or adjacent rooms, then the delay spread is virtually always below 50ns [17]. Normalized to the HIPERLAN bitrate of 20MBit/s, this results in a delay spread of $\tau = 1$. For multiple-aisle coverage, the delay spread increases. It is $\leq 150ns$ for 90% of the time.

[18] showed that there is little difference in delay spread for different UHF frequency bands.

In the subsequent Equalizer discussion, two HIPERLAN receivers will be discussed, the "simple" one for $\tau_{\text{simple}} = 1$ and the "robust" one for $\tau_{\text{robust}} = 3$.

The delay spread values are used subsequently to determine the filter lengths of a decision feedback equalizer (DFE), which is introduced in the next section.

2.3. The Decision Feedback Equalizer

The Decision Feedback Equalizer (DFE) [19] (see Figure 2.2) consists of Feedforward Filters, Feedback Filters, and Decision Devices. Both the Feedforward and the Feedback Filters are usually realized as transversal finite impulse response (FIR) filters. The Feedforward and the Feedback filters as well as the Decision Devices operate once per received symbol vector.

The early literature [20, 21, 22] operated the DFE in Decision-Directed (DD) mode and adjusted the filter coefficients iteratively from the error signal $\tilde{\mathbf{d}}_i - \hat{\mathbf{d}}_i$. This method is unsuited to packet transmission systems, due to the large number of training symbols required for the equalizer to reach its steady state solution (around 1000 symbols).

Fechtel and Meyr [23] and Tidestav [24] made a case for directly computing the optimal filter coefficients from the channel impulse response.

In order to compute the optimal filter coefficients, the Channel Impulse Response (CIR) first needs to be estimated. Packet communication systems usually utilize pre-, post- or midambles consisting of known symbols to aid synchronisation and CIR estimation. The exact procedure to obtain the CIR estimate is outside the scope of this work, the reader is referred to eg. [25, 26].

The block labelled Coefficient Calculator computes the optimal filter coefficients from the estimated CIR by solving a system of equations. These equations will be derived in section 2.3.1. Efficient algorithms for computing

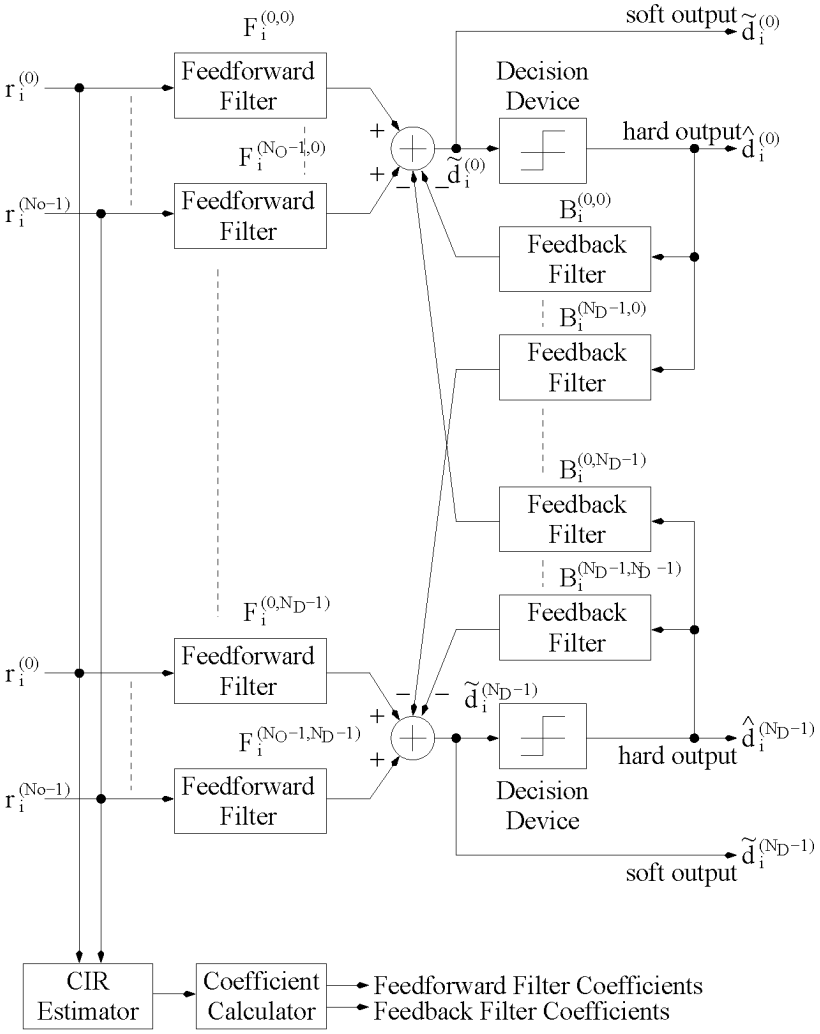

Figure 2.2: Decision feedback equalizer
The decision devices operate on the symbol estimate $\tilde{d}_i$. The feedback filters remove the postcursor of the intersymbol interference, i.e. the influence of the past already decided symbols, while the feedforward filters minimize the effect of the precursor of the ISI, i.e. the future symbols.

the filter coefficients and architectures for implementing them are developed in this work.

Two optimality criteria have been used, the Zero Forcing (ZF) criterion and the Minimum Mean Square Error (MMSE) criterion. The ZF equalizer tries to invert the channel impulse response without taking noise into account. Notches will therefore be compensated by high gain, which leads to untolerable noise enhancement. The ZF equalizer can therefore be used only on relatively flat channels with high Signal to Noise Ratios (SNR). The MMSE criterion minimizes the energy of the error at the decision point $\tilde{\mathbf{d}}_i$.

Al-Dhahir and Cioffi [27] derived equations for the filter coefficients by writing the equalization problem of a whole block in matrix form. They used a finite CIR put into a fully windowed Toeplitz matrix. Their factorization procedure effectively computes the optimal feedback filter coefficients for every possible decision delay Δ. They then choose the Δ that results in the least decision point mean squared error (MSE) by back substituting the chosen feedback coefficients into a triangular system.

It is however advantageous to first compute the feedforward coefficients, as the feedback coefficients can then be computed using the feedforward filter, which shall be shown later. Therefore, the derivation will follow the one of Proakis [28, 29].

The derivation below assumes that there are no decision errors, i.e. $\hat{\mathbf{d}}_i = \mathbf{d}_i$. At high symbol error rates, error propagation in the feedback filter becomes a problem. The often-suggested solution of moving the feedback filter into the transmitter (Tomlinson-Harashima precoding, [30]) is impractical, as the channel and therefore the optimal feedback filter coefficients are only known after the packet is transmitted. A more practical approach is to move the feedback filter into the decoder for an error correcting code. If a convolutional code is used, the feedback filter can be moved into the branch processing unit. This is called Per-Survivor Processing [31].

Sections 2.3.1 to 2.3.2 derive equations for the optimal feedforward and feedback filter coefficients and a few additional parameters. Similar derivations can be found in the literature, for example [28, 29, 23, 24]. Sections 2.3.3 to 2.3.5.1 present new results derived from monte carlo computer simulations.

2.3.1. DFE Key Equations

The decision point signal $\tilde{\mathbf{d}}_i$ is

$$\tilde{\mathbf{d}}_{i-\Delta} := \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \mathbf{r}_{i-j} - \sum_{j=1}^{N_b} \mathbf{B}_j \hat{\mathbf{d}}_{i-\Delta-j}. \tag{2.4}$$

$\mathbf{r}_i$ are the received signal samples, $\mathbf{F}_i$ the feedforward filter coefficients, N_f the number of feedforward coefficients, $\mathbf{B}_i$ the feedback filter coefficients, N_b the number of feedback coefficients, $\hat{\mathbf{d}}_i$ are the decided symbols and Δ is the decision delay. The error signal $\mathbf{e}_i$

$$\mathbf{e}_i := \tilde{\mathbf{d}}_i - \mathbf{d}_i = \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \mathbf{r}_{i+\Delta-j} - \sum_{j=1}^{N_b} \mathbf{B}_j \hat{\mathbf{d}}_{i-j} - \mathbf{d}_i \tag{2.5}$$

at the decision point shall be minimized. Invoking the orthogonality principle [32]:

$$E[\mathbf{e}_i \mathbf{r}_{i+\Delta-k}^H] = \mathbf{0} \qquad 0 \le k < N_f \tag{2.6a}$$

$$E[\mathbf{e}_i \hat{\mathbf{d}}_{i-k}^H] = \mathbf{0} \qquad 1 \le k \le N_b \tag{2.6b}$$

For the following derivation, it is assumed that the data symbols are independent and have unit energy on average

$$E[\mathbf{d}_i \mathbf{d}_j^H] = \begin{cases} \mathbf{I} & i = j \\ \mathbf{0} & i \neq j \end{cases}, \tag{2.7a}$$

and that noise and data are independent

$$E[\mathbf{d}_i \mathbf{n}_j^H] = \mathbf{0}. \tag{2.7b}$$

The two covariances

$$E[\mathbf{d}_k \mathbf{r}_i^H] = \mathbf{C}_{i-k}^H \tag{2.8a}$$

$$\begin{aligned} E[\mathbf{r}_j \mathbf{r}_i^H] &= E[\sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \mathbf{C}_{j-k} \mathbf{d}_k \mathbf{d}_l^H \mathbf{C}_{i-l}^H] + E[\mathbf{n}_j \mathbf{n}_i^H] \\ &= \sum_{k=-\infty}^{\infty} \mathbf{C}_{j-k} \mathbf{C}_{i-k}^H + E[\mathbf{n}_j \mathbf{n}_i^H] \end{aligned} \tag{2.8b}$$

are needed later on.

Equations for the feedback filter coefficients Now (2.6b) is used to compute the feedback filter coefficients $\mathbf{B}_k$

$$E[\mathbf{e}_i \mathbf{d}_{i-k}^H] = \sum_{j=0}^{N_f-1} E[\mathbf{F}_j^T \mathbf{r}_{i+\Delta-j}] \mathbf{d}_{i-k}^H - \sum_{j=1}^{N_b} \mathbf{B}_j E[\mathbf{d}_{i-j} \mathbf{d}_{i-k}^H] - E[\mathbf{d}_i \mathbf{d}_{i-k}^H] \tag{2.9}$$

Moving $\mathbf{B}_j$ of (2.9) to the lefthand side leads to equations for the feedback filter coefficients

$$\mathbf{B}_k = \sum_{j=0}^{N_f-1} \mathbf{F}_j^T E[\mathbf{d}_{i-k} \mathbf{r}_{i+\Delta-j}^H]^H = \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \mathbf{C}_{k+\Delta-j}. \tag{2.10}$$

Inserting (2.10) into (2.6a) and simplifying yields the equations for the DFE feedforward coefficients

$$\begin{aligned}
E[\mathbf{e}_i\mathbf{r}_{i+\Delta-k}^H] &= \sum_{j=0}^{N_f-1} \mathbf{F}_j^T E[\mathbf{r}_{i+\Delta-j}\mathbf{r}_{i+\Delta-k}^H] - \\
&\quad \sum_{j=1}^{N_b}\sum_{l=0}^{N_f-1} \mathbf{F}_l^T \mathbf{C}_{j+\Delta-l} E[\mathbf{d}_{i-j}\mathbf{r}_{i+\Delta-k}^H] - E[\mathbf{d}_i\mathbf{r}_{i+\Delta-k}^H] \\
&= \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \Big(\sum_{l=-\infty}^{\infty} \mathbf{C}_{i+\Delta-j-l}\mathbf{C}_{i+\Delta-k-l}^H + E[\mathbf{n}_{i+\Delta-j}\mathbf{n}_{i+\Delta-k}^H]\Big) - \\
&\quad \sum_{j=1}^{N_b}\sum_{l=0}^{N_f-1} \mathbf{F}_l^T \mathbf{C}_{j+\Delta-l}\mathbf{C}_{j+\Delta-k}^H - \mathbf{C}_{\Delta-k}^H \\
&= \sum_{j=0}^{N_f-1}\sum_{l=-\infty}^{\infty} \mathbf{F}_j^T \mathbf{C}_{l+\Delta-j}\mathbf{C}_{l+\Delta-k}^H + \\
&\quad \sum_{j=0}^{N_f-1} \mathbf{F}_j^T E[\mathbf{n}_{i+\Delta-j}\mathbf{n}_{i+\Delta-k}^H] - \\
&\quad \sum_{j=0}^{N_f-1}\sum_{l=1}^{N_b} \mathbf{F}_j^T \mathbf{C}_{l+\Delta-j}\mathbf{C}_{l+\Delta-k}^H - \mathbf{C}_{\Delta-k}^H \\
&= \sum_{j=0}^{N_f-1} \sum_{l\in\mathbb{Z}\setminus\{1\ldots N_b\}} \mathbf{F}_j^T \mathbf{C}_{l+\Delta-j}\mathbf{C}_{l+\Delta-k}^H + \\
&\quad \sum_{j=0}^{N_f-1} \mathbf{F}_j^T E[\mathbf{n}_{i+\Delta-j}\mathbf{n}_{i+\Delta-k}^H] - \mathbf{C}_{\Delta-k}^H \\
&= \mathbf{0}.
\end{aligned} \tag{2.11}$$

Equations for the feedforward filter coefficients Rearranging (2.11) and assuming that the noise is stationary leads to a system of $N_f \times N_O$ linear equations with N_D different right hand sides

$$\sum_{j=0}^{N_f-1} \Big(\sum_{k\in\mathbb{Z}\setminus\{1\ldots N_b\}} \mathbf{C}_{k+\Delta-i}^* \mathbf{C}_{k+\Delta-j}^T + E[\mathbf{n}_{-i}^* \mathbf{n}_{-j}^T]\Big)\mathbf{F}_j = \mathbf{C}_{\Delta-i}^*. \tag{2.12}$$

Writing these linear systems in block matrix form is more convenient

$$(\breve{\mathbf{C}} + \breve{\mathbf{N}})\bar{\mathbf{F}} = \bar{\mathbf{C}}. \tag{2.13}$$

$\breve{\mathbf{N}}$ is the noise covariance block matrix. Its i, j-th element is

$$\breve{\mathbf{N}}_{(i,j)} = E[\mathbf{n}_i^* \mathbf{n}_j^T]. \tag{2.14a}$$

Since the noise is stationary, $\breve{\mathbf{N}}$ is a hermitian symmetric block Toeplitz matrix. A Toeplitz matrix is a matrix whose elements satisfy $\breve{\mathbf{N}}_{(i,j)} = \breve{\mathbf{N}}_{(i-j)}$ [33]. If the noise is white and the noise contributions to different channel outputs independent and of equal energy, $\breve{\mathbf{N}} = N_0\mathbf{I}$ reduces to a scaled $N_f N_O \times N_f N_O$ identity matrix, where $N_0 = E[\mathbf{n}_i^{(j)} \mathbf{n}_i^{(j)H}]$ is the noise energy. $\breve{\mathbf{C}}$ is a hermitian block matrix that depends only on the channel impulse response. $\breve{\mathbf{C}}$ is an $N_f \times N_f$ matrix whose i, j-th element is the $N_O \times N_O$ matrix

$$\breve{\mathbf{C}}_{(i,j)} = \sum_{k \in \mathbb{Z} \setminus \{1 \ldots N_b\}} \mathbf{C}^*_{k+i+\Delta-(N_f-1)} \mathbf{C}^T_{k+j+\Delta-(N_f-1)}. \tag{2.14b}$$

$\breve{\mathbf{C}}$ is also hermitian symmetric, but not Toeplitz, due to the postcursor ISI cancellation done by the feedback filter. This results in the "hole" from $1 \ldots N_b$ in the CIR covariance sum of (2.14b). For convenience with further derivations, the row and column indices have been reversed. They run from 0 to $N_f - 1$. The right hand side of the equation system is

$$\bar{\mathbf{C}} = \begin{pmatrix} \mathbf{C}^*_{\Delta-N_f+1} \\ \mathbf{C}^*_{\Delta-N_f+2} \\ \vdots \\ \mathbf{C}^*_{\Delta} \end{pmatrix}, \tag{2.14c}$$

and the vector of unknowns is

$$\bar{\mathbf{F}} = \begin{pmatrix} \mathbf{F}_{N_f-1} \\ \mathbf{F}_{N_f-2} \\ \vdots \\ \mathbf{F}_0 \end{pmatrix}. \tag{2.14d}$$

In order to simplify the above expressions, the decision delay is set to $\Delta := N_f - 1$ and the CIR $\mathbf{C}_i$ is truncated outside the interval $0 \leq i < N_f$. The justification for truncating the CIR can be found in section 2.2.1. The

same equation for the decision delay Δ has also been found by Al-Dhahir and Cioffi [34]. In practice, the synchronisation circuitry would search for the CIR window of length N_f containing the most energy.

To gain more insight into the structure of the matrix $\mathbf{A} := \breve{\mathbf{C}} + \breve{\mathbf{N}}$, it is written down explicitly

$$\mathbf{A} = \begin{pmatrix} \mathbf{C}_0^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(0,0)} & \mathbf{C}_0^*\mathbf{C}_1^T+\breve{\mathbf{N}}_{(0,1)} & \mathbf{C}_0^*\mathbf{C}_2^T+\breve{\mathbf{N}}_{(0,2)} & \cdots \\ \mathbf{C}_1^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(1,0)} & \mathbf{C}_1^*\mathbf{C}_1^T+\mathbf{C}_0^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(1,1)} & \mathbf{C}_1^*\mathbf{C}_2^T+\mathbf{C}_0^*\mathbf{C}_1^T+\breve{\mathbf{N}}_{(1,2)} & \cdots \\ \mathbf{C}_2^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(2,0)} & \mathbf{C}_2^*\mathbf{C}_1^T+\mathbf{C}_1^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(2,1)} & \mathbf{C}_2^*\mathbf{C}_2^T+\mathbf{C}_1^*\mathbf{C}_1^T+\mathbf{C}_0^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(2,2)} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{2.15}$$

It is a remarkable fact that each element of this matrix is the sum of its north west element plus an additional term.

$\breve{\mathbf{C}}$ can now be represented as the product of two lower antitriangular block Hankel matrices

$$\breve{\mathbf{C}} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{C}_0^* \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{C}_0^* & \mathbf{C}_1^* \\ \mathbf{0} & \mathbf{0} & \iddots & \iddots & \iddots \\ \mathbf{0} & \mathbf{C}_0^* & \cdots & \mathbf{C}_{N_f-3}^* & \mathbf{C}_{N_f-2}^* \\ \mathbf{C}_0^* & \mathbf{C}_1^* & \cdots & \mathbf{C}_{N_f-2}^* & \mathbf{C}_{N_f-1}^* \end{pmatrix} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & 0 & \mathbf{C}_0^T \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{C}_0^T & \mathbf{C}_1^T \\ \mathbf{0} & \mathbf{0} & \iddots & \iddots & \iddots \\ \mathbf{0} & \mathbf{C}_0^T & \cdots & \mathbf{C}_{N_f-3}^T & \mathbf{C}_{N_f-2}^T \\ \mathbf{C}_0^T & \mathbf{C}_1^T & \cdots & \mathbf{C}_{N_f-2}^T & \mathbf{C}_{N_f-1}^T \end{pmatrix}. \tag{2.16}$$

This structure can be used to simplify the computation of the matrix $\breve{\mathbf{C}}$.

Decision Point MSE and Bias The Decision Feedback Equalizer is a biased receiver, i.e. there is self-interference. To compute the bias,

$$\begin{aligned} \tilde{\mathbf{d}}_{i-\Delta} &= \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \Big(\sum_{k=-\infty}^{\infty} \mathbf{C}_{i-j-k}\mathbf{d}_k + \mathbf{n}_{i-j} \Big) - \\ &\quad \sum_{k=1}^{N_b} \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \mathbf{C}_{k+\Delta-j}\mathbf{d}_{i-\Delta-k} \\ &= \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \Big(\sum_{k\in\mathbb{Z}\setminus\{1\ldots N_b\}} \mathbf{C}_{k+\Delta-j}\mathbf{d}_{i-\Delta-k} + \mathbf{n}_{i-j} \Big) \end{aligned} \tag{2.17}$$

is needed. The bias $\boldsymbol{\alpha}$ can be computed as

$$E[\tilde{\mathbf{d}}_{i-\Delta}|\mathbf{d}_{i-\Delta}] = \Big(\sum_{j=0}^{N_f-1} \mathbf{F}_j^T \mathbf{C}_{\Delta-j} \Big)\mathbf{d}_{i-\Delta} =: \boldsymbol{\alpha}\mathbf{d}_{i-\Delta}, \tag{2.18}$$

therefore

$$\boldsymbol{\alpha} = \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \mathbf{C}_{\Delta-j}. \tag{2.19}$$

Even though $\boldsymbol{\alpha}$ is an $N_D \times N_D$ matrix, the lowercase greek letter $\boldsymbol{\alpha}$ has been used, mainly because previous DFE literature used this symbol and the uppercase $\boldsymbol{\alpha}$ has the same shape as the latin letter A. The decision point error signal is

$$\begin{aligned} \mathbf{e}_{i-\Delta} &= \tilde{\mathbf{d}}_{i-\Delta} - \mathbf{d}_{i-\Delta} \\ &= \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \Big(\sum_{k \in \mathbb{Z} \setminus \{1 \ldots N_b\}} \mathbf{C}_{k+\Delta-j} \mathbf{d}_{i-\Delta-k} + \mathbf{n}_{i-j} \Big) - \mathbf{d}_{i-\Delta}, \end{aligned} \tag{2.20}$$

and the decision point error energy is

$$\begin{aligned} E[\mathbf{e}_{i-\Delta}\mathbf{e}_{i-\Delta}^H] = & \sum_{j=0}^{N_f-1} \sum_{l=0}^{N_f-1} \mathbf{F}_j^T E[\Big(\sum_{k \in \mathbb{Z} \setminus \{1 \ldots N_b\}} \mathbf{C}_{k+\Delta-j} \hat{\mathbf{d}}_{i-\Delta-k} + \mathbf{n}_{i-j} \Big) \\ & \Big(\sum_{m \in \mathbb{Z} \setminus \{1 \ldots N_b\}} \hat{\mathbf{d}}_{i-\Delta-m}^H \mathbf{C}_{m+\Delta-l}^H + \mathbf{n}_{i-l}^H \Big)] \mathbf{F}_l^* \\ & - \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \mathbf{C}_{\Delta-j} - \sum_{j=0}^{N_f-1} \mathbf{C}_{\Delta-j}^H \mathbf{F}_j^* + \mathbf{I} \\ = & \sum_{j=0}^{N_f-1} \sum_{k=0}^{N_f-1} \mathbf{F}_j^T \Big(\sum_{l \in \mathbb{Z} \setminus \{1 \ldots N_b\}} \mathbf{C}_{l+\Delta-j} \mathbf{C}_{l+\Delta-k}^H + E[\mathbf{n}_{i-j}\mathbf{n}_{i-l}^H] \Big) \mathbf{F}_k^* \\ & - \boldsymbol{\alpha} - \boldsymbol{\alpha}^H + \mathbf{I} \\ = & \sum_{k=0}^{N_f-1} \mathbf{C}_{\Delta-k}^H \mathbf{F}_k^* - \boldsymbol{\alpha} - \boldsymbol{\alpha}^H + \mathbf{I} \\ = & \boldsymbol{\alpha}^H - \boldsymbol{\alpha} - \boldsymbol{\alpha}^H + \mathbf{I} = \mathbf{I} - \boldsymbol{\alpha}. \end{aligned} \tag{2.21}$$

The SNR of the biased decision feedback equalizer is therefore

$$SNR_{biased} = (\mathbf{I} - \boldsymbol{\alpha})^{-1}, \tag{2.22}$$

assuming $\mathbf{I} - \boldsymbol{\alpha}$ is invertible.

Cioffi, Dudevoir, Eyuboglu and Forney have shown [35, 36] that the optimal unbiased MMSE DFE is the optimal biased MMSE DFE with the bias

removed. While the unbiased MMSE DFE has a lower decision point SNR, it has a smaller decision error probability. Therefore, the simulation results of the decision point error energy printed in this work will always be those of the unbiased DFE.

The bias can be removed by multiplying the decision point signal with $\boldsymbol{\alpha}^{-1}$ (assuming $\boldsymbol{\alpha}$ is invertible) or equivalently by scaling the decision regions with $\boldsymbol{\alpha}$. For the popular binary antipodal signal constellation, the decision threshold is zero and therefore bias scaling invariant.

The decision point error signal of the unbiased DFE is

$$\mathbf{e}_{i-\Delta}^{u} = \boldsymbol{\alpha}^{-1} \sum_{j=0}^{N_f-1} \mathbf{F}_j^T \Big(\sum_{k \in \mathbb{Z} \setminus \{1 \ldots N_b\}} \mathbf{C}_{k+\Delta-j} \mathbf{d}_{i-\Delta-k} + \mathbf{n}_{i-j} \Big) - \mathbf{d}_{i-\Delta} \tag{2.23}$$

and the decision point error energy is

$$\begin{aligned} E[\mathbf{e}_{i-\Delta}^{u} \mathbf{e}_{i-\Delta}^{uH}] &= \boldsymbol{\alpha}^{-H} \boldsymbol{\alpha}^{H} \boldsymbol{\alpha}^{-1} - \boldsymbol{\alpha}^{-1} \boldsymbol{\alpha} - \boldsymbol{\alpha}^{H} \boldsymbol{\alpha}^{-H} + \mathbf{I} \\ &= \boldsymbol{\alpha}^{-1} - \mathbf{I} = \boldsymbol{\alpha}^{-1} (\mathbf{I} - \boldsymbol{\alpha}). \end{aligned} \tag{2.24}$$

This result is only meaningful for the single output ($N_D = 1$) case, because in the multiple output case, the decision point error signals would be scaled by the inverse of the diagonal elements of $\boldsymbol{\alpha}$ only.

The SNR of the unbiased DFE is

$$SNR_{unbiased} = \boldsymbol{\alpha} (\mathbf{I} - \boldsymbol{\alpha})^{-1}, \tag{2.25}$$

and the difference of the SNR of the biased and the unbiased DFE is

$$\begin{aligned} SNR_{biased} - SNR_{unbiased} &= (\mathbf{I} - \boldsymbol{\alpha})^{-1} - \boldsymbol{\alpha} (\mathbf{I} - \boldsymbol{\alpha})^{-1} \\ &= (\mathbf{I} - \boldsymbol{\alpha})(\mathbf{I} - \boldsymbol{\alpha})^{-1} = \mathbf{I}. \end{aligned} \tag{2.26}$$

This result has been found by Cioffi, Dudevoir, Eyuboglu and Forney [35].

2.3.2. Computing the Feedback Filter Coefficients and the Bias

Both the bias $\boldsymbol{\alpha}$ (2.19) and the feedback filter coefficients $\mathbf{B}_k$ (2.10) are just different time offsets of the convolution of the CIR $\mathbf{C}_i$ and the feedforward filter coefficients $\mathbf{F}_j$. Thus the feedforward filter section of the DFE may be used to compute the bias and the feedback filter coefficients. This is illustrated in Figure 2.3. First, the N_f CIR coefficients are clocked in. The bias $\boldsymbol{\alpha}$ appears at the output. After the next clock, $\mathbf{B}_1$ appears at the output and so forth. If the CIR is truncated ($\mathbf{C}_i = \mathbf{0}$ for $i \geq N_f$), then $\mathbf{B}_k = 0$ for $k \geq N_f$.

When using the truncated CIR, the number of feedback filter coefficients is best set to $N_b := N_f - 1$. When the untruncated CIR is used, the number of feedback coefficients may be enlarged, but the gain is negligible.

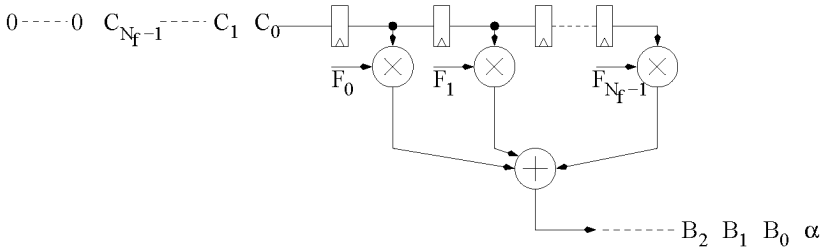

Figure 2.3: Computation of the feedback filter coefficients and the bias

2.3.3. Choosing N_f

It is clear that the optimal choice of the Equalizer feedforward filter length N_f depends on the channel model and its parameters.

Computer simulations with the channel model of section 2.2 have been performed and the results are plotted in Figures 2.4 to 2.9. Both the channel delay spread parameter τ and the white noise energy N_0 for the computation of the feedforward filter coefficients have been varied. Every simulation point is the average of 100'000 channel realizations. Both the residual ISI energy part of the decision point error signal and the noise energy proportionality factor have been plotted separately versus the number of feedforward taps N_f. The number of feedback taps N_b has been set to 64 to make sure that postcursor ISI is fully cancelled.

The Figures 2.4 to 2.6 for the 2-dimensional Equalizer plot the energies per real dimension to make the results comparable to the Figures 2.7 to 2.9 for the 1-dimensional Equalizer.

Several conclusions can be drawn from these plots.

- When the number of feedforward taps is chosen according to the 99.9% channel energy criterion (section 2.2.1) $N_f \approx 7\tau$, the performance is very near to the infinite length DFE performance. It is however often possible to achieve satisfactory performance with a significantly smaller N_f.

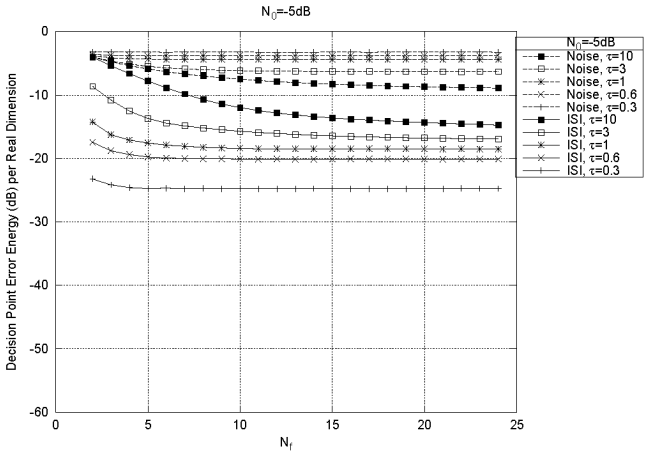

Figure 2.4: Decision point error energy vs. N_f for $N_0 = -5dB$ and 2-dim. eq. ($N_D = 1$, $N_O = 1$, $d_i \in \mathbb{C}$)

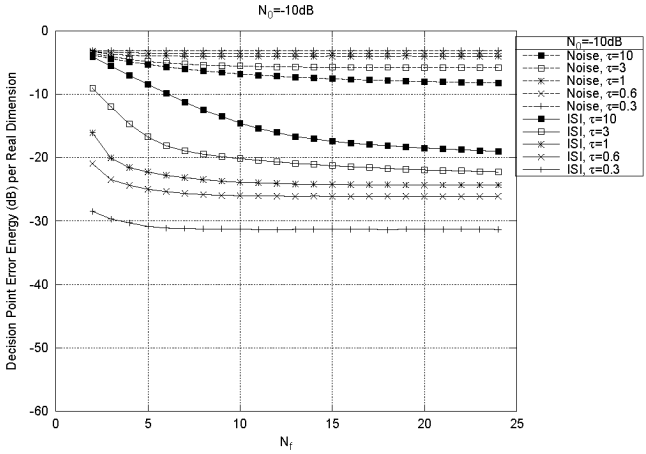

Figure 2.5: Decision point error energy vs. N_f for $N_0 = -10dB$ and 2-dim. eq. ($N_D = 1$, $N_O = 1$, $d_i \in \mathbb{C}$)

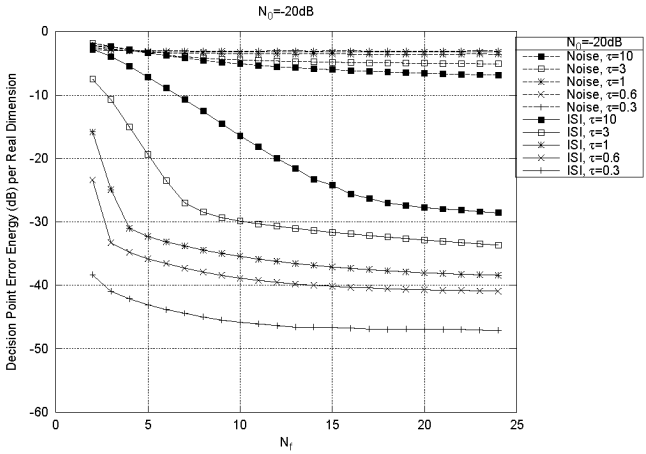

Figure 2.6: Decision point error energy vs. N_f for $N_0 = -20dB$ and 2-dim. eq. ($N_D = 1$, $N_O = 1$, $d_i \in \mathbb{C}$)

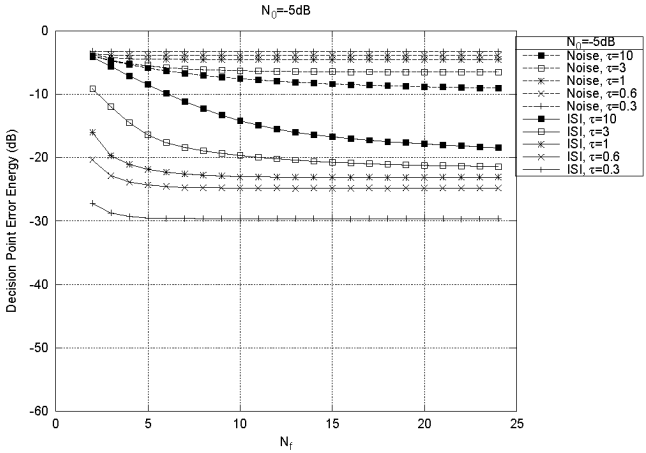

Figure 2.7: Decision point error energy vs. N_f for $N_0 = -5dB$ and 1-dim. eq. ($N_D = 1$, $N_O = 2$, $d_i \in \mathbb{R}$)

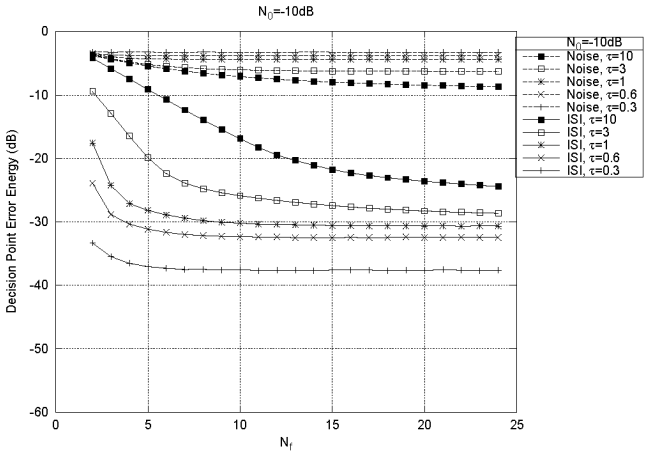

Figure 2.8: Decision point error energy vs. N_f for $N_0 = -10dB$ and 1-dim. eq. ($N_D = 1$, $N_O = 2$, $d_i \in \mathbb{R}$)

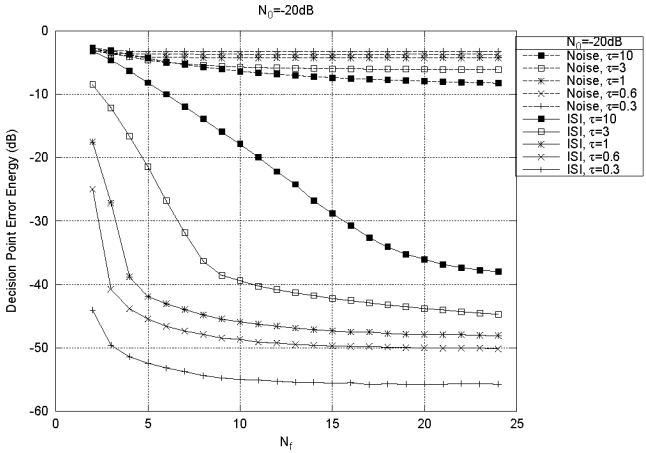

Figure 2.9: Decision point error energy vs. N_f for $N_0 = -20dB$ and 1-dim. eq. ($N_D = 1$, $N_O = 2$, $d_i \in \mathbb{R}$)

- The noise proportionality constant depends little on N_f. The residual ISI has a distinct knee. N_f should be chosen near the residual ISI knee.
- The noise proportionality constant depends little on the assumed noise energy N_0 during the calculation of the feedforward filter coefficients, while the residual ISI depends significantly on N_0. In a well designed system, bit errors induced by noise dominate. Therefore N_0 does not need to be estimated for the computation of the feedforward filter coefficients. It is sufficient to choose an N_0 which results in a tolerable residual ISI.
- The noise proportionality constant is nearly the same for the 2-dimensional and the 1-dimensional equalizer. There is a significant difference in the residual ISI energy for the two equalizers.

2.3.4. Choosing N_b

The feedback filter multiplies the decisions $\hat{\mathbf{d}}_i$ with the feedback filter coefficients $\mathbf{B}_i$. Since $\hat{\mathbf{d}}_i$ can only take a few distinct values, the multiplications in the feedback filter section are significantly cheaper than general variable $\times$ variable multiplications. For the practically important binary antipodal constellation case ($\mathbf{d}_i, \hat{\mathbf{d}}_i \in \{+1, -1\}^{N_D \times 1}$), these multiplications reduce to add/subtracts. Computation of the feedback filter coefficients is only linearly dependent on N_b, therefore it is much less important to choose the minimum possible N_b. On the other hand, the feedback loop limits the possibilities for pipelining the feedback filter.

As can be seen in (2.10) and section 2.3.2, when working with the truncated channel impulse response, the feedback filter coefficients $\mathbf{B}_k$ are zero for $k \geq N_f$. It does not make sense to choose N_b larger than $N_f - 1$. If, however, N_f was chosen significanly smaller than 7τ, then residual ISI performance may be improved significantly by working with the untruncated channel impulse response and choosing N_b larger than N_f. This is illustrated in Figures 2.10 and 2.11.

2.3.5. The Optimal Equalizer for 1-Dimensional Signal Constellations

All signals of the Equalizer in Figure 2.2 are complex. The case where the signal constellation is real $\mathbf{d}_i \in \mathbb{R}^{N_D \times 1}$ is of great practical interest however, especially the binary antipodal signal constellation $\mathbf{d}_i \in \{+1, -1\}^{N_D \times 1}$. Binary Phase Shift Keying (BPSK) and Minimum Shift Keying (MSK) fall into this category. Here it is desirable to minimize only the real part of the error at the decision point $\tilde{\mathbf{d}}_i$ [37].

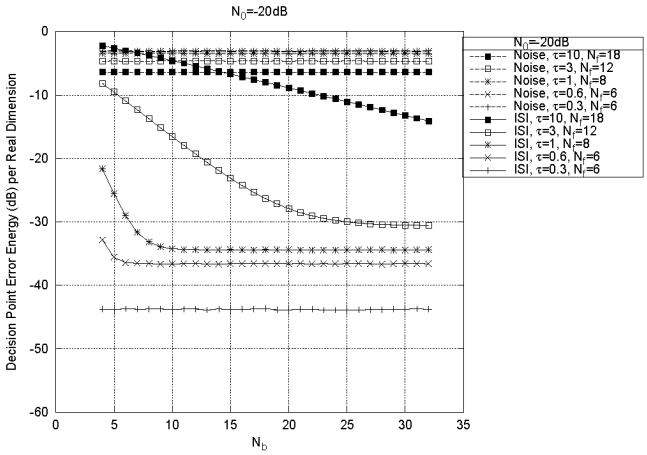

Figure 2.10: Decision point error energy vs. N_b for $N_0 = -20dB$ and 2-dim. eq. ($N_D = 1$, $N_O = 1$, $d_i \in \mathbb{C}$)

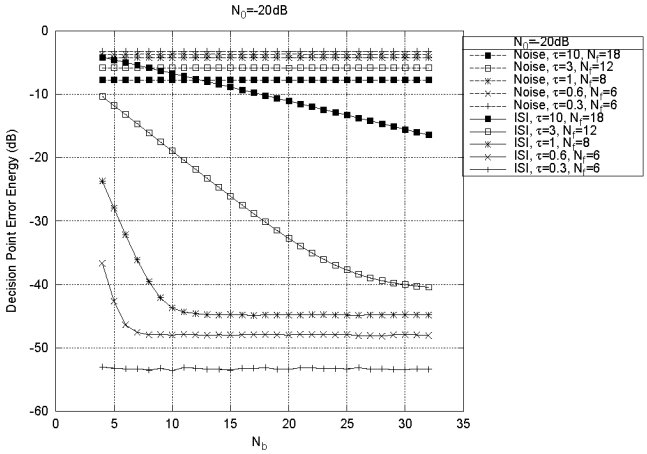

Figure 2.11: Decision point error energy vs. N_b for $N_0 = -20dB$ and 1-dim. eq. ($N_D = 1$, $N_O = 2$, $d_i \in \mathbb{R}$)

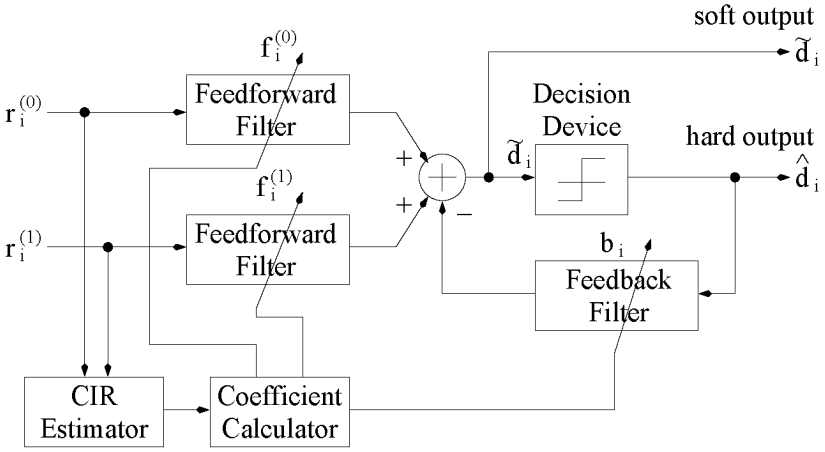

Figure 2.12: Decision feedback equalizer for $N_D = 2$

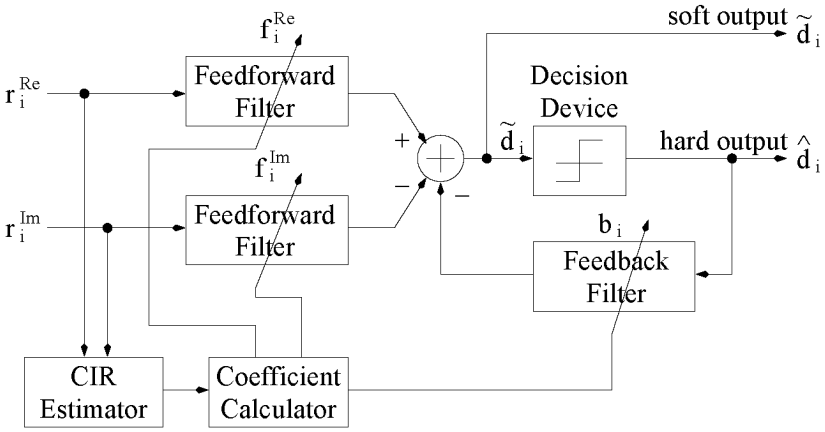

Figure 2.13: Decision feedback equalizer for 1-dimensional constellation, $N_D = 1$

It is clear that the general 2-dimensional (complex) Equalizer can also be used for a 1-dimensional signal constellation. This setup will be suboptimal, however, because the equalizer minimizes both the real and the imaginary error energy at the decision point $\tilde{d}_i$, while the decision device only takes the real part of the decision point signal into account.

Optimizing only the real part of the error energy fits well into the presented framework.

$$\Re(\mathbf{F}_j^T \mathbf{C}_i) = \Re\{(\mathbf{F}_j^{\Re T} + i\mathbf{F}_j^{\Im T})(\mathbf{C}_i^{\Re} + i\mathbf{C}_i^{\Im})\} = \mathbf{F}_j^{\Re T}\mathbf{C}_i^{\Re} - \mathbf{F}_j^{\Im T}\mathbf{C}_i^{\Im} \quad (2.27)$$

illustrates that optimizing the real part of a complex DFE can be treated, apart from the minus sign, as optimizing a real DFE with twice the number of channel outputs, the "real" channel output and the "imaginary" channel output. Indeed, Figure 2.13 illustrating the 1-dimensional equalizer looks very similar to Figure 2.12.

$$\mathbf{F}_i = \begin{pmatrix} \mathbf{F}_i^{\Re} \\ \mathbf{F}_i^{\Im} \end{pmatrix} \quad (2.28a)$$

and

$$\mathbf{C}_i = \begin{pmatrix} \mathbf{C}_i^{\Re} \\ -\mathbf{C}_i^{\Im} \end{pmatrix} \quad (2.28b)$$

show the feedforward filter coefficient vectors and the channel tap vectors for the 1-dimensional equalizer. These equations look similar to those of the complex fractionally $T/(2N_O)$ spaced equalizer, except that the formers elements are real, while the latters elements are complex.

2.3.5.1. Benefit versus Cost of the 1-Dimensional Equalizer

The 1-dimensional Equalizer requires the solution of a system of $2N_fN_O$ real linear equations while the 2-dimensional Equalizer requires the solution of N_fN_O complex linear equations. Inversion algorithms for matrices such as cholesky factorization are $O(n^3)$. The 1-dimensional equalizer therefore requires 8 times the number of operations of the 2-dimensional equalizer. Since a complex multiplication requires four real multiplications and two real additions, the 1-dimensional equalizer requires approximately twice the number of real multiplications and four times the number of real additions than the 2-dimensional equalizer to compute the optimal filter coefficients.

The feedforward and the feedback filter have the same number of taps and the same structure for both the 2-dimensional and the 1-dimensional equalizers.

As has been observed in section 2.3.3, in a well designed system noise dominates the decision point error energy. Also, the noise proportionality constant is only slightly smaller for the 1-dimensional equalizer than for the 2-dimensional equalizer. The difference is approximately $0.4dB$ for the symbol spaced equalizer.

To conclude this section, a gain of $0.4dB$ has to be traded versus a doubling in computational complexity.

3

DFE Matrix Factorization

In this chapter, conventional direct and iterative methods for solving the DFE equations are discussed. The direct methods either factor the matrix into a product of two easy to invert triangular or unitary matrices, and then compute the result by solving two easier systems, or use matrix bordering techniques to directly compute the result. The iterative methods start with an initial estimate of the solution and improve the estimate at each iteration. Emphasis is put onto the Cholesky Factorization, as an intuitive comprehension of the Cholesky Factorization is required to understand Section 3.2.4.

3.1. Problem Statement

In this chapter, methods for solving the system of linear equations

$$\mathbf{A}\begin{pmatrix}\mathbf{F}_{N_f-1}\\ \mathbf{F}_{N_f-2}\\ \vdots\\ \mathbf{F}_0\end{pmatrix}=\begin{pmatrix}\mathbf{C}_0^*\\ \mathbf{C}_1^*\\ \vdots\\ \mathbf{C}_{N_f-1}^*\end{pmatrix}, \tag{3.1}$$

the DFE key equations, are discussed. The matrix

$$\mathbf{A}=\begin{pmatrix}\mathbf{C}_0^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(0,0)} & \mathbf{C}_0^*\mathbf{C}_1^T+\breve{\mathbf{N}}_{(0,1)} & \mathbf{C}_0^*\mathbf{C}_2^T+\breve{\mathbf{N}}_{(0,2)} & \cdots\\ \mathbf{C}_1^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(1,0)} & \mathbf{C}_1^*\mathbf{C}_1^T+\mathbf{C}_0^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(1,1)} & \mathbf{C}_1^*\mathbf{C}_2^T+\mathbf{C}_0^*\mathbf{C}_1^T+\breve{\mathbf{N}}_{(1,2)} & \cdots\\ \mathbf{C}_2^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(2,0)} & \mathbf{C}_2^*\mathbf{C}_1^T+\mathbf{C}_1^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(2,1)} & \mathbf{C}_2^*\mathbf{C}_2^T+\mathbf{C}_1^*\mathbf{C}_1^T+\mathbf{C}_0^*\mathbf{C}_0^T+\breve{\mathbf{N}}_{(2,2)} & \cdots\\ \vdots & \vdots & \vdots & \ddots\end{pmatrix} \tag{3.2}$$

is highly structured; each element is the sum of its north west neighbor plus a channel dependent term and a noise dependent term. This structure can be exploited either to simplify the computation of $\mathbf{A}$ or the solution of (3.1), as shall be shown in Section 3.2.4.

3.2. Direct Methods

The direct methods for solving systems of linear equations discussed here are order recursive. They transform a problem of size N into a problem of size $N-1$. Complexity is given in terms of the problem size (number of variables) N – which is $N_f N_O$.

3.2.1. Generic LU Factorization

The general method for solving linear systems is Gaussian Elimination, also called LU-Factorization. In order to ensure stability, Pivoting (row permutations) has to be used. Pivoting introduces data dependent decisions and is therefore undesirable. $O(N^3)$ arithmetic operations are required.

3.2.2. QR Factorization

Any matrix $\mathbf{A} =: \mathbf{QR}$ can be factored into the product of a unitary matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$. Since $\mathbf{R}$ is upper triangular and the inverse

of $\mathbf{Q}$ is easy to compute (since $\mathbf{Q}$ is unitary, $\mathbf{Q}\mathbf{Q}^H = \mathbf{I} \rightarrow \mathbf{Q}^{-1} = \mathbf{Q}^H$), the system $\mathbf{Q}\mathbf{R}\bar{\mathbf{F}} = \bar{\mathbf{C}}$ can be solved easily by back substitution and matrix-vector multiplication.

$\mathbf{Q}$ is often decomposed into $N(N-1)/2$ Givens rotation matrices. The decomposition of $\mathbf{Q}$ is beneficial for numeric stability and also for VLSI integration.

Complexity of the $\mathbf{QR}$ factorization is $O(N^3)$.

3.2.3. Cholesky Factorization

The matrix to be inverted is hermitian symmetric. Furthermore, since it is the result of a least squares problem, the matrix is also positive definite. Therefore the matrix can be factored into a product of $\mathbf{A} = \mathbf{L}\mathbf{L}^H$ or $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^H$. $\mathbf{L}$ is a lower triangular matrix and $\mathbf{D}$ a diagonal matrix. The $\mathbf{L}\mathbf{L}^H$ factorization requires the computation of N inverse square roots, while the $\mathbf{L}\mathbf{D}\mathbf{L}^H$ factorization requires the computation of N divisions. For fixed point computation, the inverse square root $\frac{1}{\sqrt{x}}$ is preferrable over the division, since its output has a smaller dynamic range. Precise results at a low number of iterations have been achieved with a Newton Raphson Iteration for the inverse square root with an initial seed table [38, Chapter 21.5].

$$\mathbf{A}_N = \begin{pmatrix} a_{11} & a_{12} & \cdots \\ a_{21} & a_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} = \mathbf{L}_N \mathbf{L}_N^* + \mathbf{A}_{N-1} \tag{3.3}$$

illustrates the Cholesky recursion step of the $\mathbf{L}\mathbf{L}^H$ factorization.

$$\mathbf{L}_N = \begin{pmatrix} \frac{a_{11}}{\sqrt{a_{11}}} & 0 & \cdots \\ \frac{a_{21}}{\sqrt{a_{11}}} & 0 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \tag{3.4}$$

denotes the partial Cholesky factor computed at recursion step N. The first column or row of $\mathbf{A}$ is scaled by $\frac{1}{\sqrt{a_{11}}}$ and stored into the first column of $\mathbf{L}_N$. The rest of $\mathbf{L}_N$ is set to zero. Now $\mathbf{A}_N - \mathbf{L}_N \mathbf{L}_N^H$ results in a matrix whose first row and first column is identically zero. The problem has therefore been reduced to a problem of size $N-1$.

After $\mathbf{A}$ has been factored into the product $\mathbf{L}\mathbf{L}^H$, $\bar{\mathbf{F}}$ can be found by back substitution. This involves solving the systems $\mathbf{L}\mathbf{y} = \bar{\mathbf{C}}$ and $\mathbf{L}^H\mathbf{x} = \mathbf{y}$. These systems are "easy" because $\mathbf{L}$ and therefore $\mathbf{L}^H$ are triangular.

The complexity of the cholesky factorization is $O(N^3)$, while the complexity of the back substitution is $O(N^2)$.

Implementation remarks Since $\mathbf{A}$ is hermitian symmetric, the cholesky factorization can be performed in place. $\mathbf{L}$ is stored into one triangular part of the memory while $\mathbf{A}$ is taken out of the other triangular part.

Instead of storing $\frac{a_{ii}}{\sqrt{a_{ii}}} = \sqrt{a_{ii}}$ into the diagonal elements of $\mathbf{L}$, $\frac{1}{\sqrt{a_{ii}}}$ may be stored, turning the divisions in the back substitution into multiplications. $\frac{1}{\sqrt{a_{ii}}}$ has to be computed anyway for the row scaling.

Both the Cholesky factorization as well as the back substitution can not easily be parallelized to achieve low latency result computation. The order recursion steps cannot be overlapped, there is inherent serialization at each recursion step. Furthermore, the work to be done at each recursion step varies widely.

3.2.4. Displacement Structure Theory

(3.2) clearly shows that there is more structure to the problem than just hermitian symmetry and positive definiteness. It should be possible to exploit this structure to simplify the problem.

It turns out that the recently developed Displacement Structure Theory [39, 40] is a powerful tool to simplify structured matrix inversion problems.

$$\nabla_{\{\mathbf{Z},\mathbf{Z}\}}\mathbf{A} := \mathbf{A} - \mathbf{Z}\mathbf{A}\mathbf{Z}^H \tag{3.5}$$

is called the displacement representation of $\mathbf{A}$. $\mathbf{Z}$ is called the displacement operator. $\mathbf{Z}$ is an arbitrary strictly lower triangular (i.e. with identically zero diagonal) matrix. Since $\mathbf{Z}$ is strictly lower triangular, the first row and the first column of $\mathbf{Z}\mathbf{A}\mathbf{Z}^H$ are identically zero. Therefore, the first row and the first column of $\mathbf{A}$ and $\nabla_{\{\mathbf{Z},\mathbf{Z}\}}\mathbf{A}$ are identical.

$$\mathbf{Z} := \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots \end{pmatrix} \tag{3.6}$$

is called the lower shift matrix. $\mathbf{I}$ denotes the $N_O \times N_O$ identity matrix, and $\mathbf{0}$ denotes the $N_O \times N_O$ all zero matrix. Premultiplying a matrix with $\mathbf{Z}_n$ deletes the bottom N_O rows of the matrix and inserts N_O all zero rows at the top. Postmultiplying with $\mathbf{Z}^H$ or $\mathbf{Z}^T$ deletes the rightmost N_O columns and

inserts N_O all zero column at the left edge. It is easy to see that

$$\nabla_{\{\mathbf{Z},\mathbf{Z}\}}\mathbf{A} = \begin{pmatrix} \mathbf{C}_0^*\mathbf{C}_0^T + \breve{\mathbf{N}}_{(0,0)} & \mathbf{C}_0^*\mathbf{C}_1^T + \breve{\mathbf{N}}_{(0,1)} & \mathbf{C}_0^*\mathbf{C}_2^T + \breve{\mathbf{N}}_{(0,2)} & \cdots \\ \mathbf{C}_1^*\mathbf{C}_0^T + \breve{\mathbf{N}}_{(1,0)} & \mathbf{C}_1^*\mathbf{C}_1^T & \mathbf{C}_1^*\mathbf{C}_2^T & \cdots \\ \mathbf{C}_2^*\mathbf{C}_0^T + \breve{\mathbf{N}}_{(2,0)} & \mathbf{C}_2^*\mathbf{C}_1^T & \mathbf{C}_2^*\mathbf{C}_2^T & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{3.7}$$

can be represented as a product $\mathbf{G}\mathbf{J}\mathbf{G}^H$. $\mathbf{J} = \mathrm{diag}\{\pm 1, \pm 1, \cdots, \pm 1\}$ is called the signature matrix. $\mathbf{G}$ is called a generator of $\nabla_{\{\mathbf{Z},\mathbf{Z}\}}\mathbf{A}$, and its columns are denoted with $\mathbf{g}_0 \cdots \mathbf{g}_{r-1}$. r, the number of columns of $\mathbf{G}$, is called the displacement rank of $\mathbf{A}$.

In the white noise case, $\breve{\mathbf{N}}_{(i,j)}$ vanishes for $i \neq j$. Therefore, the only noise contribution is to the top left corner of $\nabla_{\{\mathbf{Z},\mathbf{Z}\}}\mathbf{A}$. Furthermore, $\breve{\mathbf{N}}_{(0,0)} = N_0\mathbf{I}$ is a scaled identity matrix. A suitable $\mathbf{G}$ is

$$\mathbf{G} = \begin{pmatrix} \mathbf{C}_0^* & \sqrt{N_0}\mathbf{I} \\ \mathbf{C}_1^* & \mathbf{0} \\ \mathbf{C}_2^* & \mathbf{0} \\ \vdots & \vdots \end{pmatrix} := \begin{pmatrix} \mathbf{g}_0 & \cdots & \mathbf{g}_{r-1} \end{pmatrix}, \tag{3.8}$$

and the corresponding $\mathbf{J}$ is simply the $(N_O + N_D) \times (N_O + N_D)$ identity matrix. The displacement rank is $r = N_O + N_D$.

In the case where the noise is coloured, but the noise contributions to different channel outputs are uncorrelated and have the same energy, the elements of $\breve{\mathbf{N}}$ are also scaled identity matrices, i.e. $\breve{\mathbf{N}}_{(i,j)} = N_{i-j}\mathbf{I}$. A suitable $\mathbf{G}$ is

$$\mathbf{G} = \begin{pmatrix} \mathbf{C}_0^* & \sqrt{N_0}\mathbf{I} & \mathbf{0} \\ \mathbf{C}_1^* & \frac{N_1}{\sqrt{N_0}}\mathbf{I} & \frac{N_1}{\sqrt{N_0}}\mathbf{I} \\ \mathbf{C}_2^* & \frac{N_2}{\sqrt{N_0}}\mathbf{I} & \frac{N_2}{\sqrt{N_0}}\mathbf{I} \\ \vdots & \vdots & \vdots \end{pmatrix} := \begin{pmatrix} \mathbf{g}_0 & \cdots & \mathbf{g}_{r-1} \end{pmatrix}, \tag{3.9a}$$

and the first $N_O + N_D$ elements of $\mathbf{J}$ are $+1$, while the last N_O elements of $\mathbf{J}$ are -1;

$$\mathbf{J} = \begin{pmatrix} \mathbf{I} & 0 & 0 \\ 0 & \mathbf{I} & 0 \\ 0 & 0 & -\mathbf{I} \end{pmatrix}. \tag{3.9b}$$

The displacement rank is $r = 2N_O + N_D$.

Generators are not unique. In fact, if $\mathbf{G}$ is a generator, then $\mathbf{G\Theta}$ is also a generator, provided that $\mathbf{\Theta}$ is $\mathbf{J}$-unitary, i.e. $\mathbf{\Theta J \Theta}^H = \mathbf{J}$, since $\mathbf{G\Theta J \Theta}^H \mathbf{G}^H = \mathbf{GJG}^H$.

The freedom of choosing a generator shall now be used to develop an order recursion that transforms the displacement representation of a problem of size N into the displacement representation of a problem of size $N-1$. From now on, $\mathbf{A}$ shall no longer be treated as an $N_f \times N_f$ block matrix of $N_O \times N_O$ elements, but as an $N_f N_O \times N_f N_O$ matrix of scalar elements.

First, $\mathbf{\Theta}$ shall be chosen such that $\bar{\mathbf{G}} = \mathbf{G\Theta}$ has only one nonzero element in the first row. Any popular zeroing tool such as Givens rotation, fast Givens or Householder reflections [41] may be used to find $\mathbf{\Theta}$. The column with the nonzero element in its first row is called the pivoting column $\bar{\mathbf{g}}_{pvt}$.

Since only $\bar{\mathbf{g}}_{pvt}$ has a nonzero first element, $\bar{\mathbf{g}}_{pvt}$ determines the first row and the first column of $\nabla_{\{\mathbf{Z},\mathbf{Z}\}}\mathbf{A}$ and thus also of $\mathbf{A}$. $\bar{\mathbf{g}}_{pvt}$ is therefore the first row of the cholesky factorization of $\mathbf{A}$. Subtracting $\bar{\mathbf{g}}_{pvt}\bar{\mathbf{g}}_{pvt}^H$ from $\mathbf{A}$ zeros the first column and the first row

$$\tilde{\mathbf{A}}_1 = \mathbf{A} - \bar{\mathbf{g}}_{pvt}\bar{\mathbf{g}}_{pvt}^H = \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_1 \end{pmatrix}. \tag{3.10}$$

$$\begin{aligned} \tilde{\mathbf{A}}_1 - \mathbf{F}\tilde{\mathbf{A}}_1\mathbf{F}^H &= \mathbf{A} - \bar{\mathbf{g}}_{pvt}\bar{\mathbf{g}}_{pvt}^H - \mathbf{F}(\mathbf{A} - \bar{\mathbf{g}}_{pvt}\bar{\mathbf{g}}_{pvt}^H)\mathbf{F}^H \\ &= \bar{\mathbf{G}}\mathbf{J}\bar{\mathbf{G}}^H - \bar{\mathbf{g}}_{pvt}\bar{\mathbf{g}}_{pvt}^H + \mathbf{F}\bar{\mathbf{g}}_{pvt}\bar{\mathbf{g}}_{pvt}^H\mathbf{F}^H \\ &= \begin{pmatrix} \bar{\mathbf{g}}_0 & \cdots & \mathbf{F}\bar{\mathbf{g}}_{pvt} & \cdots & \bar{\mathbf{g}}_{r-1} \end{pmatrix} \mathbf{J} \\ &\qquad \begin{pmatrix} \bar{\mathbf{g}}_0 & \cdots & \mathbf{F}\bar{\mathbf{g}}_{pvt} & \cdots & \bar{\mathbf{g}}_{r-1} \end{pmatrix}^H \\ &= \begin{pmatrix} \mathbf{0} \\ \mathbf{G}_1 \end{pmatrix} \mathbf{J} \begin{pmatrix} \mathbf{0} \\ \mathbf{G}_1 \end{pmatrix}^H \end{aligned} \tag{3.11}$$

shows the algorithm for transforming the displacement representation of $\mathbf{A}$ (order N) into the displacement representation of $\mathbf{A}_1$ (order $N-1$). This recursion step can be carried out until the problem is of size 1. (3.11) assumes that the signature matrix entry corresponding to the pivoting column is $+1$.

To summarize the algorithm:

1. Find $\mathbf{\Theta}$ such that the first row of $\mathbf{G}$ multiplied by $\mathbf{\Theta}$ results in a vector with only one nonzero element
2. Postmultiply $\bar{\mathbf{G}} = \mathbf{G\Theta}$. The column with the nonzero first element is called $\bar{\mathbf{g}}_{pvt}$.
3. Store $\bar{\mathbf{g}}_{pvt}$ into the appripriate column of the Cholesky Factor

4. Premultiply the pivoting column with $\mathbf{Z}$.

5. Delete the first row of the generator.

This algorithm factors the DFE matrix into its Cholesky factors. Just as for the Cholesky factorization, back substitution has to be performed to find the actual solution. Since $\mathbf{\Theta}$ is an $r \times r$ matrix and r is independent of the problem size N, the complexity of this algorithm is $O(N^2)$. Although $\mathbf{Z}$ can be any strictly lower triangular matrix, it is advantageous if $\mathbf{Z}$ only consists of 0 and 1 elements and furthermore only contains one 1 per row. In that case, premultiplying the pivoting column with $\mathbf{Z}$ can be realized with a temporary storage and appropriate read and write addresses. For the particular $\mathbf{Z}$ chosen in (3.6), this reduces to a simple downward shift by N_O elements, i.e. a N_O word FIFO.

3.2.5. Avoiding the Back Substitution

As has been mentioned in section 3.2.3, the back substitution, which is also required for the displacement structure factorization algorithm, is difficult to parallelize.

The goal of this section is to develop an algorithm, based on Displacement Structure Theory, that directly outputs the desired feedforward filter coefficients.

Displacement Structure Theory can be generalized to non hermitian symmetric matrices:

$$\nabla_{\{\mathbf{F}_1,\mathbf{F}_2\}}\mathbf{A} := \mathbf{A} - \mathbf{F}_1\mathbf{A}\mathbf{F}_2^H =: \mathbf{G}\mathbf{J}\mathbf{B}^H. \tag{3.12}$$

There are now two displacement operators $\mathbf{F_1}$ and $\mathbf{F_2}$. Also, $\mathbf{G} \neq \mathbf{B}$ in general. While $\mathbf{G}$ and $\mathbf{B}$ must have the same number r of columns, their number of rows differ for nonsquare matrices.

For hermitian symmetric matrices, the recursion step of the displacement structure algorithm (3.11) required the computation of an $r \times r$ matrix $\mathbf{\Theta}$ that cleared all but one entry of the first row of $\mathbf{G\Theta}$. In the general case, two $r \times r$ matrices $\mathbf{\Theta}$ and $\mathbf{\Gamma}$ have to be found that clear all but one entry of the first row of both $\mathbf{G\Theta}$ and $\mathbf{B\Gamma}$, and for which $\mathbf{\Theta J \Gamma}^H = \mathbf{J}$. This problem is much more involved than the corresponding problem in the hermitian symmetric case.

A back substitution free algorithm shall be derived. The block matrix

$$\mathbf{R} = \begin{pmatrix} \mathbf{A} & \bar{\mathbf{C}} \\ -\mathbf{I} & \mathbf{0} \end{pmatrix} \tag{3.13}$$

of size $2N_fN_O \times (N_fN_O + N_D)$ consists of the $N_fN_O \times N_fN_O$ matrix $\mathbf{A}$, the $N_fN_O \times N_D$ right hand side of that section, an $N_fN_O \times N_fN_O$ negative identity matrix $\mathbf{I}$, and an $N_fN_O \times N_D$ zero matrix.

Now the $1,1$-Schur complement [42, 43] of $\mathbf{R}$,

$$\mathbf{S} = 0 + \mathbf{I}\mathbf{A}^{-1}\bar{\mathbf{C}}, \tag{3.14}$$

is exactly the desired solution. The generators for the displacement structure representation of the $1,1$-Schur complement $\nabla_{\{\mathbf{F}_1,\mathbf{F}_2\}}\mathbf{S}$ can be found by running the recursion N_fN_O times.

In the single output ($N_D = 1$) case, $\mathbf{S}$ is an $N_fN_O \times 1$ vector, $\mathbf{F}_1\mathbf{S}\mathbf{F}_2 \equiv 0$, and therefore the generators directly represent the desired solution $\mathbf{S}$.

In the multiple output case, extra additions are required to convert from the displacement representation of $\mathbf{S}$ to $\mathbf{S}$ itself.

Suitable displacement operators

$$\mathbf{F}_1 = \begin{pmatrix} \mathbf{Z} & 0 \\ 0 & \mathbf{Z} \end{pmatrix} \tag{3.15a}$$

and

$$\mathbf{F}_2 = \begin{pmatrix} \mathbf{Z} & 0 \\ 0 & 0 \end{pmatrix} \tag{3.15b}$$

lead to low rank generators.

$$\nabla_{\{\mathbf{F}_1,\mathbf{F}_2\}}\mathbf{R} = \begin{pmatrix} \mathbf{C}_0^*\mathbf{C}_0^T+\check{\mathbf{N}}_{(0,0)} & \mathbf{C}_0^*\mathbf{C}_1^T+\check{\mathbf{N}}_{(0,1)} & \mathbf{C}_0^*\mathbf{C}_2^T+\check{\mathbf{N}}_{(0,2)} & \cdots & \mathbf{C}_0^*\mathbf{C}_{N_f-1}^T+\check{\mathbf{N}}_{(0,N_f-1)} & \mathbf{C}_0^* \\ \mathbf{C}_1^*\mathbf{C}_0^T+\check{\mathbf{N}}_{(1,0)} & \mathbf{C}_1^*\mathbf{C}_1^T & \mathbf{C}_1^*\mathbf{C}_2^T & \cdots & \mathbf{C}_1^*\mathbf{C}_{N_f-1}^T & \mathbf{C}_1^* \\ \mathbf{C}_2^*\mathbf{C}_0^T+\check{\mathbf{N}}_{(2,0)} & \mathbf{C}_2^*\mathbf{C}_1^T & \mathbf{C}_2^*\mathbf{C}_2^T & \cdots & \mathbf{C}_2^*\mathbf{C}_{N_f-1}^T & \mathbf{C}_2^* \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{C}_{N_f-1}^*\mathbf{C}_0^T+\check{\mathbf{N}}_{(N_f-1,0)} & \mathbf{C}_{N_f-1}^*\mathbf{C}_1^T & \mathbf{C}_{N_f-1}^*\mathbf{C}_2^T & \cdots & \mathbf{C}_{N_f-1}^*\mathbf{C}_{N_f-1}^T & \mathbf{C}_{N_f-1}^* \\ -\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{3.16}$$

shows the displacement structure representation using the chosen operators.

A suitable choice of $\mathbf{G}$, $\mathbf{J}$ and $\mathbf{B}$ is

$$\mathbf{GJB}^H = \begin{pmatrix} \mathbf{C}_0^* & \sqrt{N_0}\mathbf{I} \\ \mathbf{C}_1^* & \mathbf{0} \\ \mathbf{C}_2^* & \mathbf{0} \\ \vdots & \vdots \\ \mathbf{C}_{N_f-1}^* & \mathbf{0} \\ \mathbf{0} & -\frac{1}{\sqrt{N_0}}\mathbf{I} \\ \mathbf{0} & \mathbf{0} \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{C}_0^* & \sqrt{N_0}\mathbf{I} \\ \mathbf{C}_1^* & \mathbf{0} \\ \mathbf{C}_2^* & \mathbf{0} \\ \vdots & \vdots \\ \mathbf{C}_{N_f-1}^* & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \end{pmatrix}^H \tag{3.17}$$

for the white noise case and

$$\mathbf{GJB}^H = \begin{pmatrix} \mathbf{C}_0^* & \sqrt{N_0}\mathbf{I} & \mathbf{0} \\ \mathbf{C}_1^* & \frac{N_1}{\sqrt{N_0}}\mathbf{I} & \frac{N_1}{\sqrt{N_0}}\mathbf{I} \\ \mathbf{C}_2^* & \frac{N_2}{\sqrt{N_0}}\mathbf{I} & \frac{N_2}{\sqrt{N_0}}\mathbf{I} \\ \vdots & \vdots & \\ \mathbf{C}_{N_f-1}^* & \frac{N_{N_f-1}}{\sqrt{N_0}}\mathbf{I} & \frac{N_{N_f-1}}{\sqrt{N_0}}\mathbf{I} \\ \mathbf{0} & -\frac{1}{\sqrt{N_0}}\mathbf{I} & -\frac{1}{\sqrt{N_0}}\mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{C}_0^* & \sqrt{N_0}\mathbf{I} & \mathbf{0} \\ \mathbf{C}_1^* & \frac{N_1}{\sqrt{N_0}}\mathbf{I} & \frac{N_1}{\sqrt{N_0}}\mathbf{I} \\ \mathbf{C}_2^* & \frac{N_2}{\sqrt{N_0}}\mathbf{I} & \frac{N_2}{\sqrt{N_0}}\mathbf{I} \\ \vdots & \vdots & \\ \mathbf{C}_{N_f-1}^* & \frac{N_{N_f-1}}{\sqrt{N_0}}\mathbf{I} & \frac{N_{N_f-1}}{\sqrt{N_0}}\mathbf{I} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{pmatrix}^H \tag{3.18}$$

for the coloured noise case.

Again there is some flexibility in choosing the generators. A remarkable fact about the generators in (3.17) and (3.18) is however that the first $N_f N_O$ rows of $\mathbf{G}$ and $\mathbf{B}$ are equal. There are two implications of this:

- The general problem of finding $\mathbf{\Theta}$ and $\mathbf{\Gamma}$ that zeros $r-1$ elements in the first row of $\mathbf{G}$ and $\mathbf{B}$ and that satisfies $\mathbf{\Theta J \Gamma} = \mathbf{J}$ reduces to the much simpler problem of finding a $\mathbf{J}$ unitary matrix that zeros $r-1$ elements in the first row of $\mathbf{G}$

- The number of rows that need to be stored and multiplied by an $r \times r$ matrix is reduced

To summarize the algorithm:

1. Perform the recursion of section 3.2.4, the order reduction step, $N_f N_O$ times. Additionally, at each step, the last N_D rows of $\mathbf{B}$ have to be multiplied by $\mathbf{\Theta}$ as well.

2. Multiply the remaining N_D rows of $\mathbf{B}$ with the remainder of $\mathbf{G}$ to obtain the displacement representation of the feedforward filter coefficients, or, in the single output ($N_D = 1$) case, the feedforward filter coefficients itself.

Clearly, the possibility to find generators where the first $N_f N_O$ lines of $\mathbf{G}$ and $\mathbf{B}$ are the same is caused by the special right hand side (i.e. the matrix $\bar{\mathbf{C}}$). For general right hand sides, two possibilities exist to fit the problem into the framework of this section.

Two block columns both containing the right hand side of the equation system and having $\mathbf{I}$ and $-\mathbf{I}$ as corresponding block diagonal entries of the signature matrix $\mathbf{J}$ can be added to the generator. Because the right hand side now equals a generator (block) column, the methods of this section can be applied. The downside is an increase of the displacement rank by two times the number of right hand side columns.

Another solution is the computation of the inverse of $\mathbf{A}$ by computing the Schur complement of

$$\mathbf{R} = \begin{pmatrix} \mathbf{A} & \mathbf{I} \\ \mathbf{I} & 0 \end{pmatrix}. \tag{3.19}$$

The matrix $\mathbf{R}$ in (3.19) is now hermitian symmetric again, and the $1,1$-Schur complement is $\mathbf{S} = 0 - \mathbf{I}\mathbf{A}^{-1}\mathbf{I}$ the negative inverse of $\mathbf{A}$. The final solution may then be obtained by computing a matrix-matrix product, which exhibits more parallelism than the back substitution step.

3.2.6. Bounds for the Diagonal of the Cholesky Factor L

In this section, upper and lower bounds for the diagonal elements of the cholesky factor $\mathbf{L} = [l_{i,j}]$ are derived.

The trace of $\mathbf{A}$

$$\begin{aligned}\mathrm{Tr}\{\mathbf{A}\} = \mathrm{Tr}\{\mathbf{L}\mathbf{L}^H\} &= \sum_{i=0}^{N_f N_O-1} \sum_{j=0}^{N_f N_O-1} l_{(i,j)} l^*_{(i,j)} \\ &= \sum_{i=0}^{N_f N_O-1} |l_{(i,i)}|^2 + \sum_{i=0}^{N_f N_O-1} \sum_{\substack{0\le j<N_f N_O \\ j\neq i}} |l_{(i,j)}|^* \\ &\geq \sum_{i=0}^{N_f N_O-1} |l_{(i,i)}|^2 \geq |l_{(i,i)}|^2\end{aligned} \tag{3.20}$$

upper bounds the square of the diagonal elements of the cholesky factor. The trace of $\mathbf{A}$ can be upper bounded by

$$\mathrm{Tr}\{\mathbf{A}\} \leq N_f \sum_{i=0}^{N_O-1} \left(\breve{\mathbf{N}}^{(i,i)}_{(0,0)} + \sum_{j=0}^{N_f-1} (\mathbf{C}^*_j \mathbf{C}^T_j)^{(i,i)} \right), \tag{3.21}$$

in other words, N_f times the energy of the noise and the channel. The bound is exact for an ideal channel

$$\mathbf{C}^*_i \mathbf{C}^T_i = \begin{cases} \mathbf{I} & i = 0 \\ \mathbf{0} & i \neq 0 \end{cases}. \tag{3.22}$$

A lower bound for the diagonal elements can be derived from the displacement structure factorization algorithm for the white noise case. $l_{i,i}$ is the first element of the pivoting column at the i-th iteration. The first N_O diagonal elements can be computed with

$$l_{(i,i)} = \sqrt{N_0 + \ldots} \qquad 0 \leq i < N_O, \tag{3.23}$$

and subsequent diagonal elements with

$$l_{(i,i)} = \sqrt{l^2_{(i-N_O,i-N_O)} + \ldots} \qquad N_O \leq i < N_O N_f, \tag{3.24}$$

where "..." denotes additional nonnegative terms. Therefore, $l_{(i,i)} \geq \sqrt{N_0}$.

3.3. Iterative Methods

Iterative solution methods [43] first "guess" the solution vectors $\bar{\mathbf{F}}$. They subsequently successively reduce the distance between the true solution and the estimate.

$$\mathbf{M}\bar{\mathbf{F}}^{(k+1)} = \mathbf{N}\bar{\mathbf{F}}^{(k)} + \bar{\mathbf{C}} \tag{3.25}$$

shows the equation for the k-th iteration. $\bar{\mathbf{F}}^{(k)}$ represents the k-th estimate of the desired solution $\bar{\mathbf{F}}$, and $\mathbf{A} := \mathbf{M} - \mathbf{N}$ is called the *splitting* of the matrix $\mathbf{A}$. Obviously the splitting should be chosen such that systems of the form $\mathbf{Mx} = \mathbf{y}$ can be computed easily. The spectral radius, that is the magnitude of the largest eigenvalue, of $\mathbf{M}^{-1}\mathbf{N}$ determines the convergence rate.

The most basic iterative solution methods are the Jacobi and the Gauss-Seidel iteration. The Jacobi iteration uses an $\mathbf{M}$ that contains the diagonal part of $\mathbf{A}$, and the Gauss-Seidel iteration uses an $\mathbf{M}$ that contains the lower triangular part including the diagonal of $\mathbf{A}$. Both require the same number of computations, but the Gauss-Seidel method convergences more quickly. Jacobi or mixed Gauss-Seidel/Jacobi may however be advantageous for highly parallel implementations. It can be proved that the Gauss-Seidel iteration converges for hermitian symmetric positive definite matrices.

Methods with faster convergence such as the Successive Overrelaxation method (SOR) exist – but these methods require many more computations per iteration. As will be seen in the next section satisfactory results can be achieved with only a few Gauss-Seidel iterations, therefore these more complex methods have not been investigated.

3.3.1. Gauss-Seidel

When using iterative methods, one needs to know how many iterations need to be performed to achieve a satisfactory solution. Figures 3.1 and 3.2 show computer simulations of the Gauss-Seidel solution method. The optimal equalizer tap vector for the ideal nondispersive channel is used as the starting vector. From these graphs it can be concluded that after 3–4 iterations the results are close enough to the steady state solution.

3.4. DFE Solution Algorithm Comparison

In order to compare the different algorithms for solving the DFE equations, the exact number and type of operation that needs to be performed and the number of memory words needed for each algorithm is derived in this section.

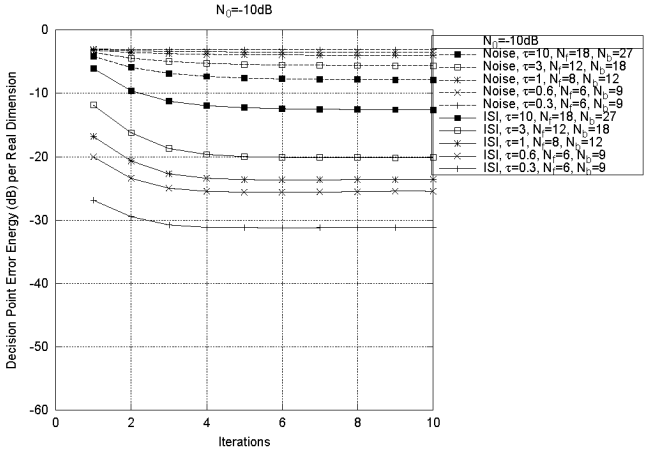

Figure 3.1: Decision point error energy vs. number of iterations for $N_0 = -10dB$ and 2-dim. eq. ($N_D = 1$, $N_O = 1$, $d_i \in \mathbb{C}$)

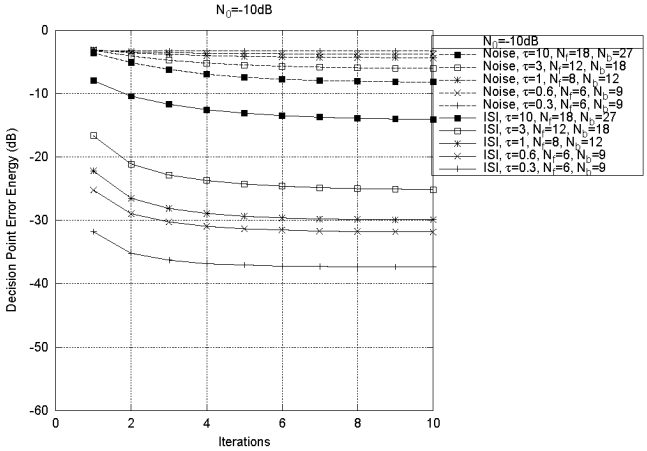

Figure 3.2: Decision point error energy vs. number of iterations for $N_0 = -10dB$ and 1-dim. eq. ($N_D = 1$, $N_O = 2$, $d_i \in \mathbb{R}$)

Note however that the number of operations alone is not the only criterion to select an algorithm for parallel hardware implementation – issues such as inherent parallelism and locality of communication are important too. In this section, two specific equalizers are considered, namely $N_D = 1$, $N_O = 1$ and $d_i \in \mathbb{C}$ termed the 2-d Equalizer, and $N_D = 1$, $N_O = 2$ and $d_i \in \mathbb{R}$ termed the 1-d Equalizer. The 2-d Equalizer represents the simplest case, namely the transmitter generating one symbol per time step, the receiver performing one measurement per time step and the equalizer optimized for a complex constellation. The 1-d Equalizer represents the same case but optimized for a real constellation.

3.4.1. Cholesky Factorization

Three tasks need to be performed, namely the computation of the matrix to be factored from the CIR estimate, the factorization, and the back substitution. As mentioned before, the matrix and the factor can be stored into the same matrix memory. Furthermore, a temporary vector is needed during the back substitution.

Table 3.1 lists the number of arithmetic operations and the memory words required for solving the DFE feedforward equation with the Cholesky factorization.

3.4.2. Displacement Structure Algorithm with Back Substitution

The main computations that need to be performed are the computation of $\mathbf{\Theta}$ and then the postmultiplication $\mathbf{G\Theta}$.

The matrix $\mathbf{A}$ does not need to be computed explicitly. At the end, however, back substitution is required to compute the final solution.

Instead of computing an $r \times r$ matrix, $\mathbf{\Theta}$ is split into several smaller tasks. First, all columns of $\mathbf{G}$ are multiplied with suitable complex values such that the imaginary parts of the elements in the first row disappear. Then, two columns are treated pairwise at a time. A real 2×2 angular rotation (or hyperbolic rotation, if the corresponding entries in the signature matrix have different signs) matrix is then computed such that the first element of one column becomes zero. This makes the multiplication $\mathbf{G\Theta}$ somewhat less regular, but the number of arithmetic operations to be performed is smaller. Table 3.2 lists the number of operations needed to perform these actions.

Multiplying the pivoting column with the displacement operator matrix does not require any arithmetic operations. It is a memory move operation.

Table 3.3 lists the total number of operations for computing the final solution.

2-dimensional Equalizer

Operation	Mult	Add
Matrix computation	$4N_f^2$	$2N_f^2 - 3N_f + 2$
Cholesky Factorization	$\frac{2}{3}N_f^3 - \frac{2}{3}N_f$	$\frac{2}{3}N_f^3 + \frac{1}{3}N_f$
Back Substitution	$4N_f^2$	$4N_f^2$
Total	$\frac{2}{3}N_f^3 + 8N_f^2 - \frac{2}{3}N_f$	$\frac{2}{3}N_f^3 + 6N_f^2 - \frac{8}{3}N_f + 2$

Operation	$1/\sqrt{x}$	Mem
Matrix computation	0	$2N_f^2$
Cholesky Factorization	N_f	0
Back Substitution	0	$2N_f$
Total	N_f	$2N_f^2 + 2N_f$

1-dimensional Equalizer

Operation	Mult	Add
Matrix computation	$4N_f^2$	$4N_f^2 - 6N_f + 4$
Cholesky Factorization	$\frac{4}{3}N_f^3 + 2N_f^2 - \frac{4}{3}N_f$	$\frac{4}{3}N_f^3 + 2N_f^2 + \frac{2}{3}N_f$
Back Substitution	$4N_f^2 + 2N_f$	$4N_f^2 + 2N_f$
Total	$\frac{4}{3}N_f^3 + 10N_f^2 + \frac{2}{3}N_f$	$\frac{4}{3}N_f^3 + 10N_f^2 - \frac{10}{3}N_f + 4$

Operation	$1/\sqrt{x}$	Mem
Matrix computation	0	$4N_f^2$
Cholesky Factorization	$2N_f$	0
Back Substitution	0	$2N_f$
Total	$2N_f$	$4N_f^2 + 2N_f$

Table 3.1: Real operations required for cholesky factorization

2-dimensional Equalizer

Operation: computing Θ	Mult	Add	$1/\sqrt{x}$	Mem
Rotate to real	$4r$	r	r	$2r$
Rotate columns	$4(r-1)$	$r-1$	$r-1$	$2(r-1)$
Total	$8r-4$	$2r-1$	$2r-1$	$4r-2$

Operation: row mult by Θ	Mult	Add	$1/\sqrt{x}$	Mem
Rotate to real	$4r$	$2r$	0	0
Rotate columns	$8(r-1)$	$4(r-1)$	0	0
Total	$12r-8$	$6r-4$	0	0

1-dimensional Equalizer

Operation: computing Θ	Mult	Add	$1/\sqrt{x}$	Mem
Rotate columns	$4(r-1)$	$r-1$	$r-1$	$2(r-1)$
Total	$4r-4$	$r-1$	$r-1$	$2r-2$

Operation: row mult by Θ	Mult	Add	$1/\sqrt{x}$	Mem
Rotate columns	$4(r-1)$	$2(r-1)$	0	0
Total	$4r-4$	$2r-2$	0	0

Table 3.2: Real operations required for Θ computation

2-dimensional Equalizer

Operation	Mult	$1/\sqrt{x}$
N_f Θ computations	$8N_f r - 4N_f$	$2N_f r - N_f$
$N_f(N_f+1)/2$ row mult	$6N_f^2 r - 4N_f^2 + 6N_f r - 4N_f$	0
Back Substitution	$4N_f^2$	0
Total	$6N_f^2 r + 14N_f r - 8N_f$	$2N_f r - N_f$
Total white n. $r = 2$	$12N_f^2 + 20N_f$	$3N_f$
Total coloured n. $r = 3$	$18N_f^2 + 34N_f$	$5N_f$

Operation	Add	Mem
N_f Θ computations	$2N_f r - N_f$	$4r - 2$
$N_f(N_f+1)/2$ row mult	$3N_f^2 r - 2N_f^2 + 3N_f r - 2N_f$	$N_f^2 + 2N_f r$
Back Substitution	$4N_f^2$	$2N_f$
Total	$3N_f^2 r + 2N_f^2 + 5N_f r - 3N_f$	$N_f^2 + 2N_f r + 2N_f + 4r - 2$
Total white n. $r = 2$	$8N_f^2 + 7N_f$	$N_f^2 + 6N_f + 6$
Total coloured n. $r = 3$	$11N_f^2 + 12N_f$	$N_f^2 + 8N_f + 10$

1-dimensional Equalizer

Operation	Mult	$1/\sqrt{x}$
$2N_f$ Θ computations	$8N_f r - 8N_f$	$2N_f r - 2N_f$
$N_f(2N_f+1)$ row mult	$8N_f^2 r - 8N_f^2 + 4N_f r - 4N_f$	0
Back Substitution	$4N_f^2 + 2N_f$	0
Total	$8N_f^2 r - 4N_f^2 + 12N_f r - 10N_f$	$2N_f r - 2N_f$
Total white n. $r = 3$	$20N_f^2 + 26N_f$	$4N_f$
Total coloured n. $r = 4$	$28N_f^2 + 38N_f$	$6N_f$

Operation	Add	Mem
$2N_f$ Θ computations	$2N_f r - 2N_f$	$4r - 4$
$N_f(2N_f+1)$ row mult	$4N_f^2 r - 4N_f^2 + 2N_f r - 2N_f$	$2N_f^2 + 2N_f r$
Back Substitution	$4N_f^2 + 2N_f$	$2N_f$
Total	$4N_f^2 r + 4N_f r - 2N_f$	$2N_f^2 + 2N_f r + 2N_f + 4r - 4$
Total white n. $r = 3$	$12N_f^2 + 10N_f$	$2N_f^2 + 8N_f + 8$
Total coloured n. $r = 4$	$16N_f^2 + 14N_f$	$2N_f^2 + 10N_f + 12$

Table 3.3: Real operations required for displacement structure factorization

3.4.3. Displacement Structure Algorithm without Back Substitution

Computations necessary for this algorithm is similar to the Displacement Structure Factorization algorithm, except that:

2-dimensional Equalizer

Operation	Mult	$1/\sqrt{x}$
N_f $\ominus$ computations	$8N_f r - 4N_f$	$2N_f r - N_f$
$3N_f(N_f+1)/2$ row mult	$18N_f^2 r - 12N_f^2 + 18N_f r - 12N_f$	0
Generator multiplication	$4N_f r$	0
Total	$18N_f^2 r - 12N_f^2 + 30N_f r - 16N_f$	$2N_f r - N_f$
Total white n. $r=2$	$24N_f^2 + 44N_f$	$3N_f$
Total coloured n. $r=3$	$42N_f^2 + 74N_f$	$5N_f$

Operation	Add	Mem
N_f $\ominus$ computations	$2N_f r - N_f$	$4r-2$
$3N_f(N_f+1)/2$ row mult	$9N_f^2 r - 6N_f^2 + 9N_f r - 6N_f$	$4N_f r + 2r$
Generator multiplication	$4N_f r - 2N_f$	0
Total	$9N_f^2 r - 6N_f^2 + 15N_f r - 9N_f$	$4N_f r + 6r - 2$
Total white n. $r=2$	$12N_f^2 + 21N_f$	$8N_f + 10$
Total coloured n. $r=3$	$21N_f^2 + 36N_f$	$12N_f + 16$

1-dimensional Equalizer

Operation	Mult	$1/\sqrt{x}$
$2N_f$ $\ominus$ computations	$8N_f r - 8N_f$	$2N_f r - 2N_f$
$3N_f(2N_f+1)$ row mult	$24N_f^2 r - 24N_f^2 + 12N_f r - 12N_f$	0
Generator multiplication	$2N_f r$	0
Total	$24N_f^2 r - 24N_f^2 + 22N_f r - 20N_f$	$2N_f r - 2N_f$
Total white n. $r=3$	$48N_f^2 + 46N_f$	$4N_f$
Total coloured n. $r=4$	$72N_f^2 + 68N_f$	$6N_f$

Operation	Add	Mem
$2N_f$ $\ominus$ computations	$2N_f r - 2N_f$	$4r-4$
$3N_f(2N_f+1)$ row mult	$12N_f^2 r - 12N_f^2 + 6N_f r - 6N_f$	$4N_f r + r$
Generator multiplication	$2N_f r - 2N_f$	0
Total	$12N_f^2 r - 12N_f^2 + 10N_f r - 10N_f$	$4N_f r + 5r - 4$
Total white n. $r=3$	$24N_f^2 + 20N_f$	$12N_f + 11$
Total coloured n. $r=4$	$36N_f^2 + 30N_f$	$16N_f + 16$

Table 3.4: Real operations required for displacement structure solution

- at each recursion step, $N_f + 1$ or $2N_f + 1$ additional rows have to be multiplied by Θ for the 2-d and the 1-d equalizer, respectively.
- there is no back substitution
- the last row of $\mathbf{B}$ needs to be multiplied with the remainder of $\mathbf{G}$.

Table 3.4 lists the total number of operations.

3.4.4. Gauss-Seidel Iteration

As has been shown in the last section, four iterations are normally enough to obtain a satisfactory solution. Table 3.5 lists the number of operations required for four iterations.

2-dimensional Equalizer

Operation	Mult	Add	Div	Mem
Matrix computation	$4N_f^2$	$2N_f^2 - 3N_f + 2$	0	$2N_f^2$
Gauss-Seidel Iteration	$4N_f^2 - 4N_f$	$4N_f^2 - 4N_f$	$2N_f$	0
Total (4 Iter)	$20N_f^2 - 16N_f$	$18N_f^2 - 19N_f + 2$	$8N_f$	$2N_f^2$

1-dimensional Equalizer

Operation	Mult	Add	Div	Mem
Matrix computation	$4N_f^2$	$4N_f^2 - 6N_f + 4$	0	$4N_f^2$
Gauss-Seidel Iteration	$4N_f^2 - 2N_f$	$4N_f^2 - 2N_f$	$2N_f$	0
Total (4 Iter)	$20N_f^2 - 8N_f$	$20N_f^2 - 14N_f + 4$	$8N_f$	$4N_f^2$

Table 3.5: Real operations required for Gauss-Seidel iteration

3.4.5. Discussion

Figures 3.3 through 3.6 plot the number of multiplications and the number of memory words required for the different equalizer algorithms versus the number of feedforward taps N_f. The number of multiplications has been chosen because most adds can be fused with multiplications to obtain multiply-accumulate (MAC) operations, and together these operations outweigh any other arithmetic operations.

Looking at the number of multiplications, Cholesky factorization is still an attractive solution for typical problem sizes even though it is $O(N^3)$ and the alternatives all are $O(N^2)$ because of its low proportionality factor. For single MAC DSPs and RISC processors, the number of multiplications is indeed an important parameter. For VLSI hardware implementation and even multiple ALU/SIMD processors, the regularity of the data flow is at least as important as the raw number of operations [44]. Here, all other alternatives are better.

The Gauss-Seidel algorithm is not very attractive. It computes an approximate solution with almost the same number of multiplications as the displacement structure based algorithms.

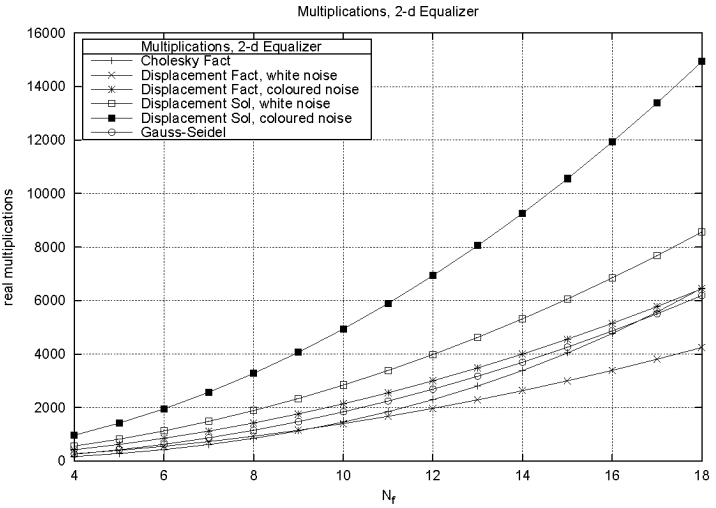

Figure 3.3: Number of multiplications of 2-d equalizer algorithms

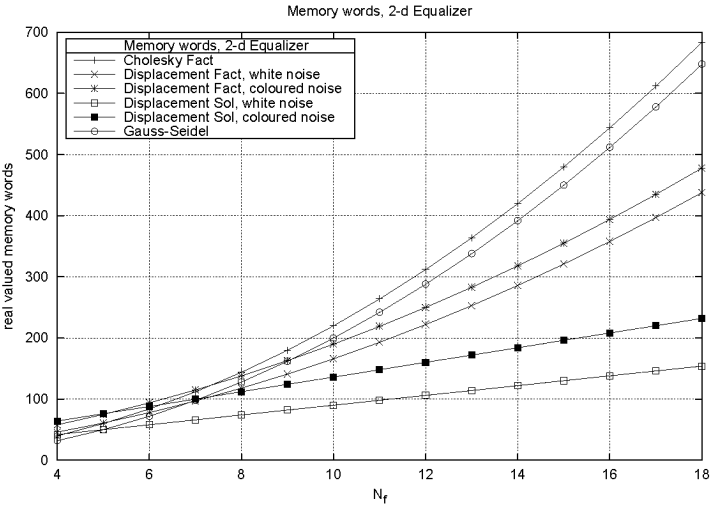

Figure 3.4: Storage requirements of 2-d equalizer algorithms

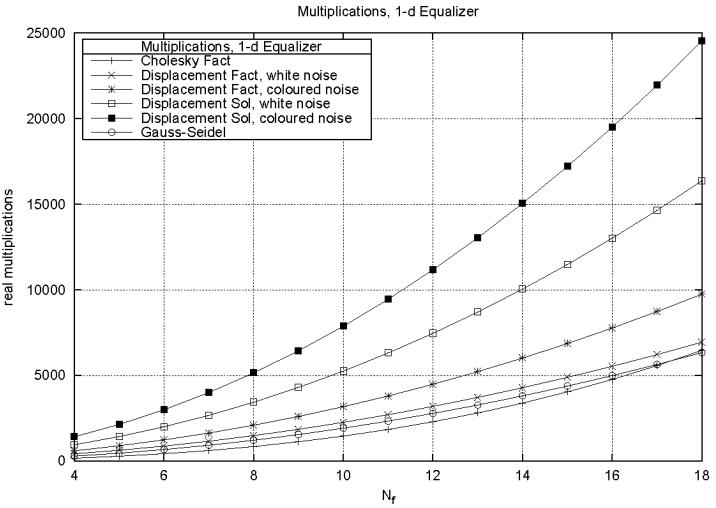

Figure 3.5: Number of multiplications of 1-d equalizer algorithms

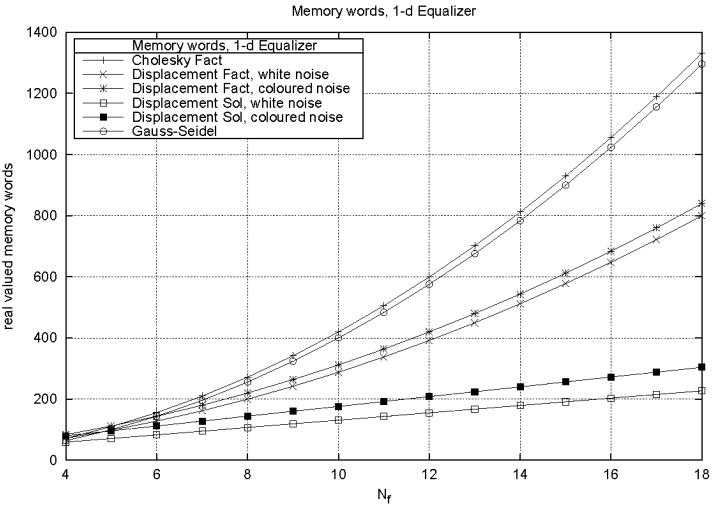

Figure 3.6: Storage requirements of 1-d equalizer algorithms

The Displacement Structure Solution algorithm has very desirable properties for VLSI implementation, so it will be considered in the next chapter.

Part III

Implementation

4

Fast VLSI Architectures for the Displacement Structure Algorithms

In section 4.1, previous publications are reviewed. In section 4.2, a family of architectures for implementing the displacement structure algorithms are presented. These architectures consist of a chain of processing elements that compute the result in $O(N)$ time. Section 4.3 presents the detailed architecture of the processing elements. To illustrate the capabilities of the proposed architectures, section 4.4 compares different architectures for computing the DFE coefficients of an equalizer suitable for HIPERLAN I.

4.1. State of the Art

Many publications suggest the use of the cholesky factorization for solving the DFE equations, eg. [29, 23].

4.1.1. Systolic Arrays for Cholesky Factorization

[45] proposed a systolic array for computing the $\mathbf{LDL}^T$ factorization. The array consists of $\frac{1}{2}N(N+1)$ processing elements. Each processing element consists of one or two real multipliers, depending whether the data is real or complex, except one processing element, which consists of a divider. Because full multiplications and divisions need to be performed per array clock cycle, the array needs to be clocked much slower than the CORDIC based arrays. The latency is $\approx 3N$ array cycles. [46] proposed another systolic array for cholesky factorization.

[47] also describes a systolic array for cholesky decomposition using $\frac{1}{2}N(N+1)$ processing elements. The processing elements contain hyperbolic CORDIC hardware.

4.1.2. Systolic Arrays for QR Factorization

Instead of the cholesky factorization, the so called $\mathbf{QR}$ factorization may be used as well. The $\mathbf{QR}$ factorization decomposes $\mathbf{A}$ into a product of a unitary matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$. Systolic arrays have been proposed for the $\mathbf{QR}$ factorization [48, 49, 50, 51]. The array consists of $\frac{1}{2}N(N+1)$ processing elements and has a triangular shape. The processing elements compute multiplies, adds, and possibly divisions or reciprocal square roots.

[52, 53, 54, 55] suggest the use of angular CORDIC based processing elements. The CORDIC (COordinate Rotation on DIgital Computers) technique [56, 57] computes angular or hyperbolic rotations in the cartesian plane by decomposing the rotation into multiple microrotations. The microrotations are chosen such that the total rotation angle converges to the desired angle and that the individual microrotations are easy to implement. In the simplest case, only two shifts and two adds/subtracts need to be computed per microrotation. In rotation mode, CORDIC rotates a given point in the cartesian plane by a given angle. In vectoring mode, CORDIC rotates a given point in the cartesian plane onto the horizontal axis and records the total rotation angle. CORDIC based processing elements are advantageous for VLSI implementation compared to the multiply-accumulate type processing elements.

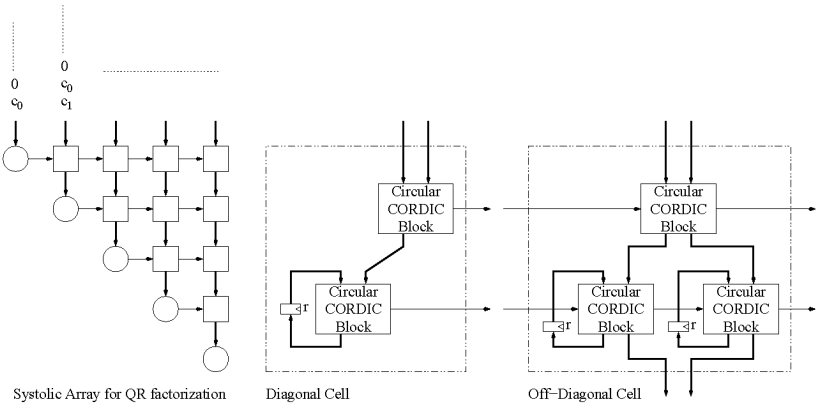

Figure 4.1: CORDIC based systolic array for **QR** factorization

Figure 4.1 depicts the systolic array and the contents of the processing elements. The N diagonal cells operate in vectoring mode, while the off diagonal cells operate in rotation mode. The array requires N array clock cycles to compute the result, but due to the feedback loop involving $\mathbf{r}$ inside the cells, the cells themselves cannot be fully pipelined, and an array clock requires N_{ROT} cycles, where N_{ROT} denotes the number of microrotations of the CORDIC blocks.

Haykin [50] suggested to follow the triangular systolic array with a linear one for performing the back substitution.

4.1.3. Linear Equalizers

Linear Equalizers resemble Decision Feedback Equalizers. The difference is the missing feedback filter. The equations for the optimal linear equalizer coefficients have Toeplitz form. Several publications, such as [58, 59], present efficient architectures for solving Toeplitz systems. Since the DFE equations do not have the Toeplitz property, these architectures cannot be adapted to solve the DFE equations.

4.1.4. Computing the Feedback Coefficients using QR Decomposition

Al-Dhahir and Sayed [60, 61] proposed the use of the **QR** factorization specifically for computing the feedback taps of a DFE. Their derivation however leads to matrices of size $N + N_{CIR}$, where N_{CIR} is the length of the channel impulse response. For typical systems, $N \approx N_{CIR}$, which leads to twice the latency and four times the hardware resources.

4.1.5. Summary

Algorithm	# PE	Latency	PE contents
QR	$\frac{1}{2}N(N+1)$	$(3N-1)N_{ROT}$	3 CORDIC
Al-Dhahir/Sayed	$\frac{1}{2}(N+N_{CIR})\cdot$ $(N+N_{CIR}+1)$	$(3(N+N_{CIR})-1)\cdot$ N_{ROT}	3 CORDIC
$\mathbf{LDL}^T$	$\frac{1}{2}N(N+1)$	$3N$	2 real Mul
Disp. Struct. Fact.	$\leq N$	$2NN_{ROT}+1$	4 CORDIC

Table 4.1: Hardware architecture comparison for 2-d equalizer coefficient computation

Table 4.1 compares the different algorithms. The displacement structure factorization hardware architecture to be introduced in sections 4.2 and 4.3 has been included as well. While all algorithms have a latency of approximately N clocks, all but the displacement structure algorithm have a hardware complexity of N^2. When directly mapping the displacement structure algorithm onto a linear chain of processing elements, it requires N processing elements. However, due to the regular data flow, processing elements may be reused without destroying the local communication property, leading to even lower hardware complexity while sacrificing the possibility to start computing a new problem while the previous one is still being processed.

4.2. General Architecture

The natural way to implement the Displacement Structure family of algorithms is to process the generator matrix (or matrices) row wise. This directly leads to the architecture in Figure 4.2 for the Displacement Structure Factorization algorithm.

The generator rows enter one row at a time at the left and are fed through a chain of $N = N_f N_O$ processing elements. One processing element is re-

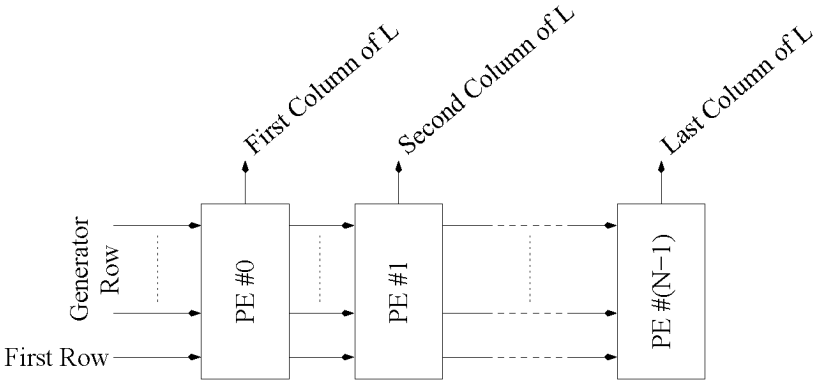

Figure 4.2: Architecture for displacement structure factorization

sponsible for one recursion step of the algorithm. The following tasks need to be performed by the processing elements:

1. Upon receiving the first row (as indicated by the signal `FIRST_ROW`, compute Θ and multiply the row with Θ.
2. Multiply all subsequent rows with Θ.
3. Output the pivoting column to the top for storing it into the Cholesky Factor $\mathbf{L}$.
4. Multiply the pivoting column with $\mathbf{F}$. With the given displacement operators this amounts to shifting and selectively zeroing elements in the column.
5. Delete the first row and pass the remaining rows to the right, including generation of the `FIRST_ROW` signal for the new first row.

A detailed discussion of the implementation of the individual processing elements is given in section 4.3, It is assumed that the processing elements are pipelined and that they accept a new row at every pipeline clock. A pipeline clock may however consist of a (constant) multiple of hardware clocks.

Due to the order recursive nature of the algorithm, not all processing elements are fully utilized. The first one (PE #0) is fully utilized, the second

one (PE #1) has an utilization ratio of $(N-1)/N$, and the last one has an utilization ratio of $1/N$.

Under the assumption that each PE takes at least one pipeline clock cycle to process each row, there is no benefit in processing multiple rows at the same time. Since the last PE only has to process one row, the latency for the full result to be available is NL_{PE}, where L_{PE} is the number of pipeline clock cycles each PE needs to process one row. Under the assumption that $L_{PE} \geq 1$, at the time when the single element last column of the cholesky factor $\mathbf{L}$ is available, the first PE will already have processed all N elements of the first column of $\mathbf{L}$.

With the architecture given in Figure 4.2, it is possible to start a new computation every N pipeline clock cycles, while the previous one is still being processed.

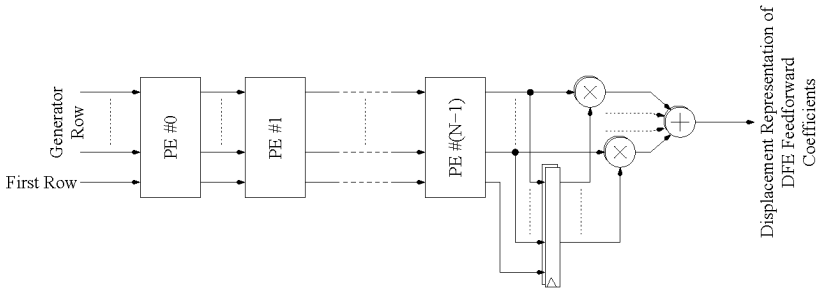

Figure 4.3: Architecture for displacement structure solution

Figure 4.3 shows the corresponding architecture for the displacement structure solution algorithm. For this algorithm, $2N + N_D$ rows need to be processed, so a new computation can be started every $2N+N_D$ pipeline clock cycles. Instead of storing the columns of the cholesky factor, the desired output is the generator output of the last processing element.

It is beneficial to feed the rows of both generator matrices $\mathbf{G}$ and $\mathbf{B}$ in the order given by

$$\begin{pmatrix} \mathbf{G}_1 = \mathbf{B}_1 \\ \mathbf{B}_2 \\ \mathbf{G}_2 \end{pmatrix} \tag{4.1}$$

from top to bottom. $\mathbf{G}_1$ and $\mathbf{B}_1$ are the rows of $\mathbf{G}$ and $\mathbf{B}$ that are equal, $\mathbf{B}_2$ denotes the remaining N_D rows of $\mathbf{B}$ and $\mathbf{G}_2$ consists of the remaining

N rows of $\mathbf{G}$. That way, the remaining rows of $\mathbf{B}$, which must be multiplied/accumulated with the remainig rows of $\mathbf{G}$, leave the last processing element first. They can then be latched and fed to $N_D r$ possibly complex multipliers, together with the remaining rows of $\mathbf{G}$. The output of the adders that follow the multipliers is the displacement representation of the desired feedforward filter coefficients, or in the single output ($N_D = 1$) case, the desired feedforward filter coefficients itself.

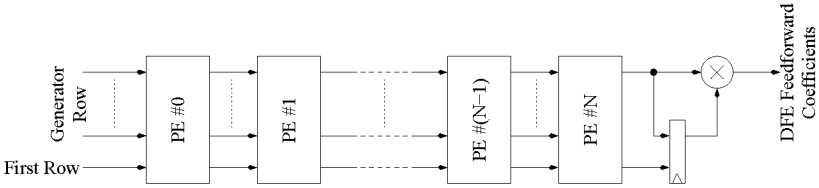

Figure 4.4: Modified architecture for displacement structure solution

A slight modification of the architecture in Figure 4.3 suitable for the single output ($N_D = 1$) case is depicted in Figure 4.4. Here, another slightly modified processing element is appended at the end of the chain of PE's. This last processing element only performs the Θ related operations. It does not apply the displacement operator $\mathbf{F}$ nor does it delete the first row.

This last processing element zeros $r - 1$ elements of the remaining row of $\mathbf{B}$, leaving only one nonzero element whose imaginary part is also zeroed. Thus, the r complex multiplications required in Figure 4.3 are now reduced to a single multiplication by a real value per feedforward coefficient. This multiplication does not even need to be performed; it only scales the feedforward coefficients and thus also the feedback coefficients and the bias α by a constant. For the binary antipodal constellation, whose decision device is scaling invariant, only the sign of α has to be retained and xor'ed with the output of the decision device. In order to avoid increasing the dynamic range of the DFE filter sections, it is desirable however to do some normalization of the feedforward coefficients, eg. by using a barrel shifter.

4.2.1. Reduced Hardware Complexity

As mentioned before, a new computation can be started while the current one is still being processed. This capability is often not necessary. But since all processing elements perform the same computation, the number of processing

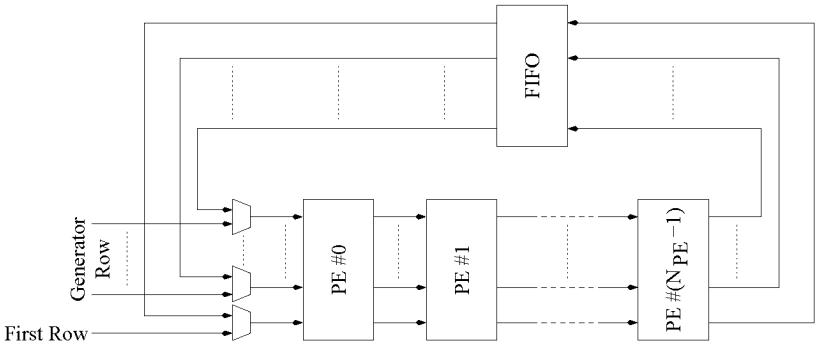

Figure 4.5: Recycling processing elements

elements may be reduced and the data may be cycled multiple times through a (shorter) chain of PE's. Figure 4.5 illustrates this idea.

The FIFO is only necessary if $N_{PE}L_{PE} < N$ for the Displacement Structure Factorization or $N_{PE}L_{PE} < 2N + N_D$ for the Displacement Structure Solution. On the other hand, if $N_{PE}L_{PE} \geq N$ for the Displacement Structure Factorization or $N_{PE}L_{PE} \geq 2N + N_D$ for the Displacement Structure Solution, then the utilization of the processing elements may be increased at the expense of an increased latency.

4.3. The Processing Elements

The main purpose of each processing element is to find a suitable $\mathbf{J}$ unitary matrix $\mathbf{\Theta}$ that zeros all but one element in the first generator row and then multiply the whole generator with $\mathbf{\Theta}$.

The case where $\mathbf{G} \in \mathbb{R}^{n\times r}$ will be discussed first, with the necessary extensions for $\mathbf{G} \in \mathbb{C}^{n\times r}$ presented afterwards.

4.3.1. Working with Real Numbers

Instead of solving the whole problem at the same time, a divide and conquer approach shall be chosen. The matrix $\mathbf{\Theta}$ is split into a number of simpler matrices, i.e. $\mathbf{\Theta} := \mathbf{\Theta}_1\mathbf{\Theta}_2 \cdots \mathbf{\Theta}_j$. Each of these simpler submatrices only deals with two columns of the generator, while leaving all other columns unaffected.

The purpose of these smaller matrices $\mathbf{\Theta}_i$ is to zero the first element of one of the two columns it processes. The condition is that $\mathbf{\Theta}_i$ must be $\mathbf{J}_i$ unitary, i.e. $\mathbf{\Theta}_i \mathbf{J}_i \mathbf{\Theta}_i^H = \mathbf{J}_i$, where $\mathbf{J}_i$ is the diagonal signature matrix consisting of the two entries from the signature matrix $\mathbf{J}$ corresponding to the two columns selected.

$$\begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{pmatrix} = \begin{pmatrix} \cos^2\theta_i + \sin^2\theta_i & 0 \\ 0 & \cos^2\theta_i + \sin^2\theta_i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{4.2a}$$

and

$$\begin{pmatrix} \cosh\theta_i & \sinh\theta_i \\ \sinh\theta_i & \cosh\theta_i \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \cosh\theta_i & \sinh\theta_i \\ \sinh\theta_i & \cosh\theta_i \end{pmatrix} = \begin{pmatrix} \cosh^2\theta_i - \sinh^2\theta_i & 0 \\ 0 & \sinh^2\theta_i - \cosh^2\theta_i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{4.2b}$$

show $\mathbf{J}_i$ unitary matrices for the two possible cases, namely the signature matrix entries having the same sign (4.2a) or a different sign (4.2b).

Looking at these two matrices, multiplying a row vector of two numbers with the matrix of (4.2a) is exactly what the CORDIC technique [56, 57] computes in angular rotation mode (apart from a constant factor), and multiplying by the matrix in (4.2b) performs the same computation as the CORDIC technique in hyperbolic rotation mode (again apart from a constant factor).

The problem of finding a suitable θ_i that zeros one element of the first row has to be solved. But this is exactly what the CORDIC technique computes in vectoring mode. This is an elegant result because the same hardware can be used for both tasks, namely finding $\mathbf{\Theta}$ and multiplying the generator rows with $\mathbf{\Theta}$. The CORDIC block just needs to be switched into vectoring mode for the first generator row and then back into rotation mode for subsequent rows. Note that the angle θ_i does not need to be computed explicitly. It is sufficient to store the direction of each CORDIC microrotation.

4.3.2. Working with Complex Numbers

Processing elements for complex numbers are similar to those for real numbers. There are two modifications:

1. r angular CORDIC blocks are used to make all r elements of the first row real (i.e. make their imaginary part vanishing).

2. for each CORDIC block for the Θ_i matrices, there is a second CORDIC block working always in rotation mode and processing the imaginary parts slaved to the CORDIC block for the real part.

4.3.3. The CORDIC Blocks

Figure 4.6 depicts two CORDIC blocks. Each CORDIC block consists of multiple microrotations. The microrotations may be implemented in a pipelined way, as shown, or executed serially on the same microrotation hardware. The signal FIRST_ROW is active when the first generator row is fed into the CORDIC block. It switches the upper or master CORDIC block into vectoring mode. If inactive, the master CORDIC block operates in rotation mode. In vectoring mode, the master CORDIC block zeros its lower output.

Hyperbolic CORDIC blocks need to swap their inputs if the magnitude of the upper input of the first row is smaller than the magnitude of the lower input to ensure convergence.

The complex Equalizers employ "stacked" CORDIC blocks, depicted with an arrow from the upper to the lower one. The lower or slave CORDIC block always operates in rotating mode with the rotation directions supplied by the master CORDIC block.

Figure 4.7 depicts the proposed CORDIC microrotation circuit. The circuit computes one microrotation per clock. The microrotations of a CORDIC block can either be computed sequentially on a single microrotation circuit, or they can be computed using a chain of as many microrotation circuits as there are microrotations, possibly with pipeline registers in between. In the former case, the shifter can be realized with a barrel shifter and the microrotation direction storage with an array of latches or a small RAM block. In the latter case, the shifters can be realized with wiring, because the shift count is constant.

Domain of Convergence The domain of convergence is limited, though. Angular CORDIC converges if the angle of the input vector in the cartesian plane lies within $\approx \pm 1.74$ radians. There are, however, two possible choices of θ_i, one that results in the first element of the pivoting column to be positive and another one that results in a negative element. Instead of using a prerotation stage which rotates the input into the domain of convergence, it is possible to choose the θ_i that lies in the domain of convergence. That is the purpose of the XOR gate in Figure 4.7.

For hyperbolic rotations, there are additional problems. If the magnitude of both input operands are approximately the same $|a| \approx |b|$, then $\theta_i \rightarrow \pm\infty$,

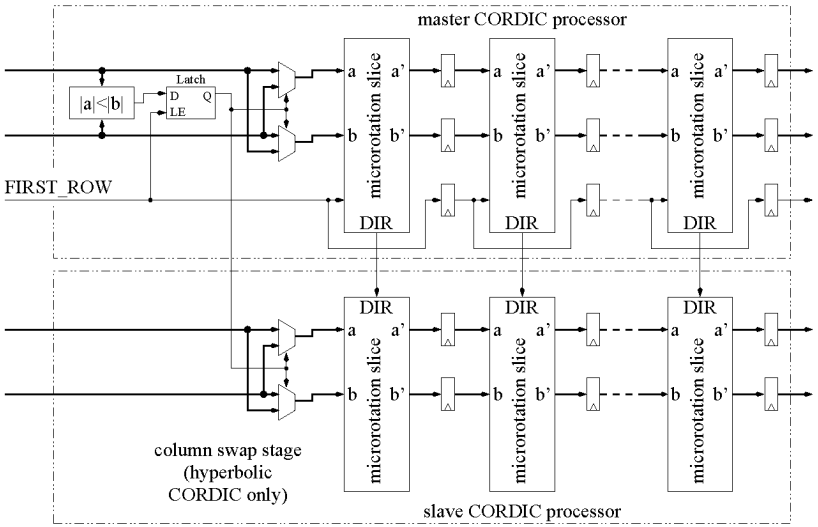

Figure 4.6: CORDIC blocks

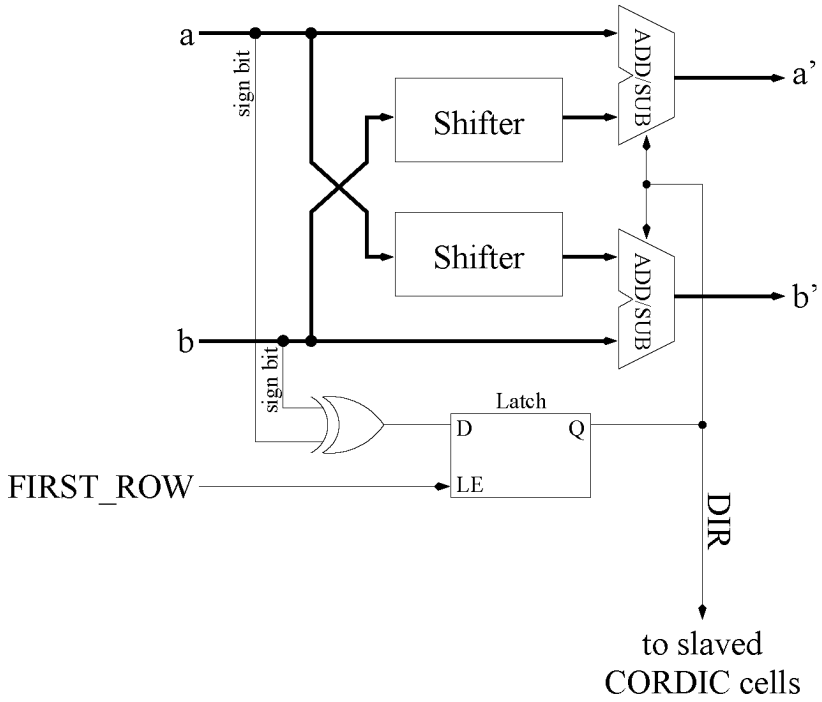

Figure 4.7: CORDIC microrotation slice

The following truth table indicates whether the Adder/Subtractors add or subtract the shifter output to or from the input, depending on the rotation direction signal DIR.

	angular CORDIC		hyperbolic CORDIC	
DIR	0	1	0	1
Upper ADD/SUB	+	−	−	+
Lower ADD/SUB	−	+	−	+

and $\sinh\theta_i$ and $\cosh\theta_i$ get very large. The generator columns whose signature matrix entry has the same sign can be grouped together. Angular CORDIC rotations may be used within both groups to zero all but one element of the first row in each group. Only one hyperbolic CORDIC rotator is then required to zero the single nonzero element in the first row of the column group having a -1 signature matrix entry. If both first row inputs to this single CORDIC rotator have approximately the same magnitude, then the corresponding diagonal element of the cholesky factor $\mathbf{L}$ is close to zero, resulting in at least one very large feedforward filter coefficient. For well behaved problems, this does not happen.

Furthermore, in hyperbolic mode, the magnitude of the input operands determine which one gets zeroed, namely the operand with the smaller magnitude. This can be circumvented by a stage in front of the hyperbolic CORDIC circuitry that swaps the columns if the magnitude of the first element of the column to be zeroed is bigger than the magnitude of the first element of the pivoting column.

4.3.4. Processing Element Examples

Figures 4.8 through 4.11 illustrate the processing element structure for $N_D = 1$ Equalizers for white and coloured noise using real and complex numbers.

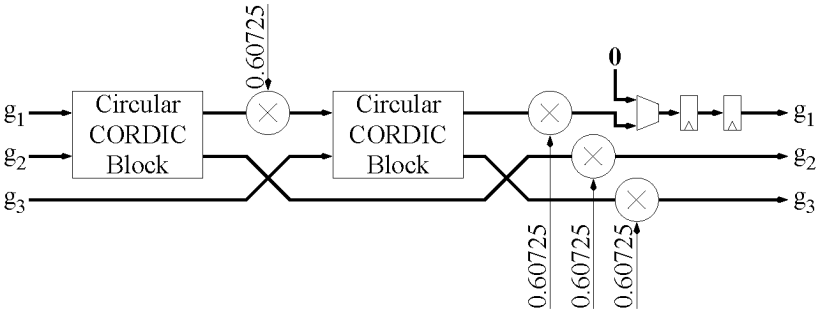

Figure 4.8: Processing element for $N_D = 1$, $N_O = 2$, real numbers, white noise

The constant coefficient multipliers cancel the gain of the CORDIC blocks. The registers and the multiplexer perform the premultiplication of the pivoting column with the displacement operators. g_1, g_2 and g_3 denote the generator columns.

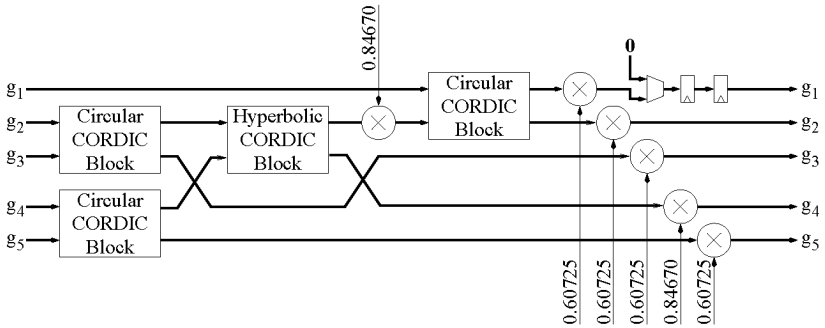

Figure 4.9: Processing element for $N_D = 1$, $N_O = 2$, real numbers, coloured noise
The constant coefficient multipliers cancel the gain of the CORDIC blocks. The registers and the multiplexer perform the premultiplication of the pivoting column with the displacement operators. $g_1 \ldots g_5$ denote the generator columns.

The constant multipliers cancel the gain introduced by the CORDIC blocks. The constants given are approximate; they depend on the number of microrotations performed by the CORDIC blocks. For the hyperbolic processors, the first stage (shift = 1) is assumed to be executed twice to improve precision when the magnitude of both input signals is approximately the same.

The multiplexer(s) and register(s) implement the multiplication of the pivoting column with the displacement operator.

g_i denotes the i-th column of the generator, g_1 is the pivoting column. g_i^{Re} and g_i^{Im} denote the real and imaginary part of the i-th column, respectively.

Figure 4.12 shows a detailed diagram of a processing element for the $N_D = 1$ Equalizer for complex numbers and white noise using pipelined CORDIC blocks.

4.3.5. Reducing Hardware Complexity further

The hardware complexity can be reduced further at the expense of the number of clock cycles required for the computation. Instead of using a processing element that operates on all columns simultaneously, it is possible to use a processing element that operates only on a lower number of columns per pass, thus requiring multiple passes per iteration. Figure 4.13 illustrates this idea. Multiplying the pivoting column with the displacement operator must

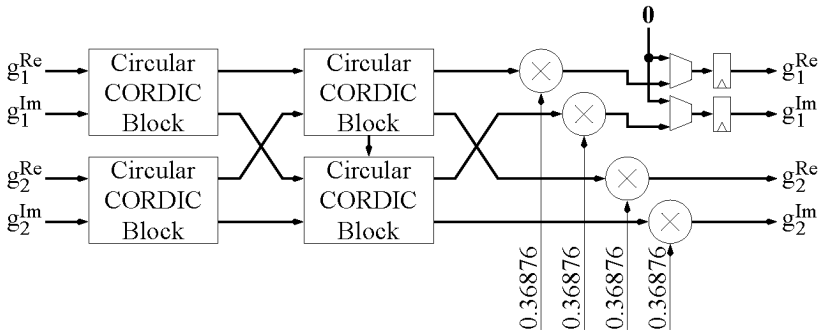

Figure 4.10: Processing element for $N_D = 1$, $N_O = 1$, complex numbers, white noise

The constant coefficient multipliers cancel the gain of the CORDIC blocks. The registers and the multiplexer perform the premultiplication of the pivoting column with the displacement operators. g_1^{Re}, g_1^{Im}, g_2^{Re} and g_2^{Im} denote the real and imaginary parts of the generator columns.

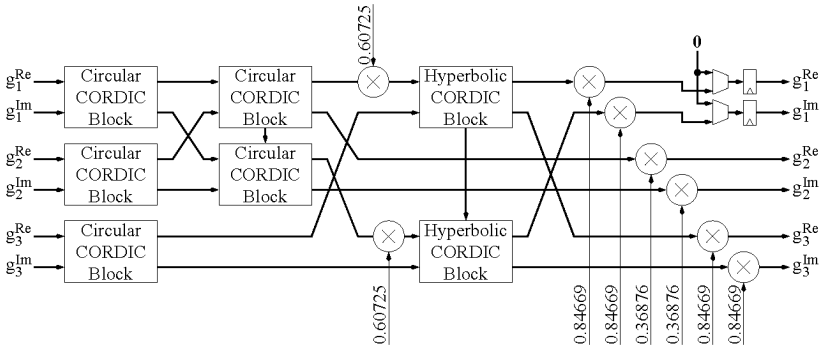

Figure 4.11: Processing element for $N_D = 1$, $N_O = 1$, complex numbers, coloured noise

The constant coefficient multipliers cancel the gain of the CORDIC blocks. The registers and the multiplexer perform the premultiplication of the pivoting column with the displacement operators. g_1^{Re}, g_1^{Im} $\ldots g_3^{Re}$, g_3^{Im} denote the real and imaginary parts of the generator columns.

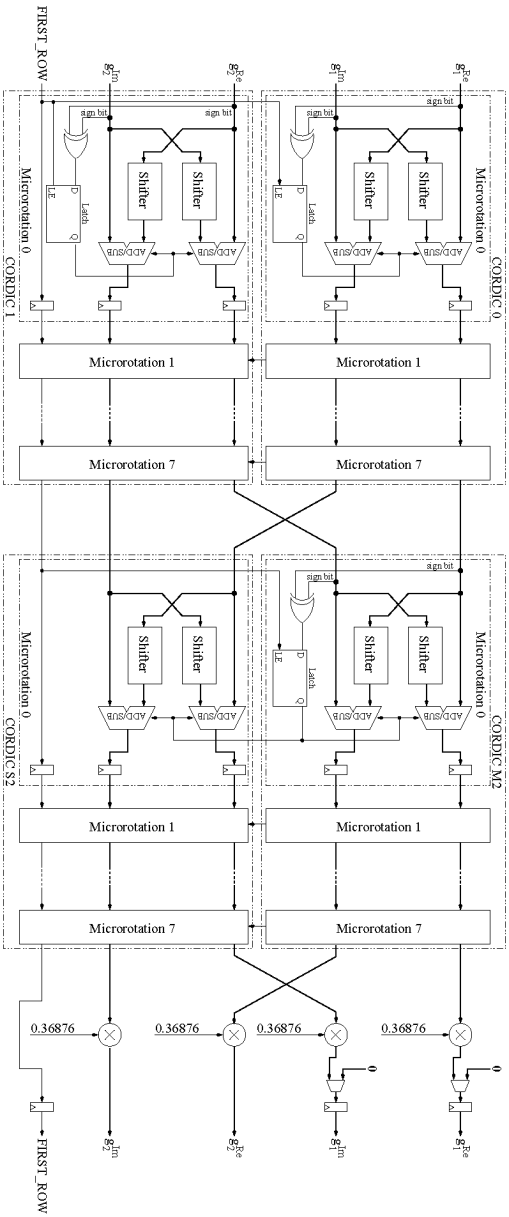

Figure 4.12: Detailed diagram of the processing element for $N_D = 1$, $N_O = 1$, complex numbers, white noise

be performed only during the last pass; the two registers may therefore be bypassed.

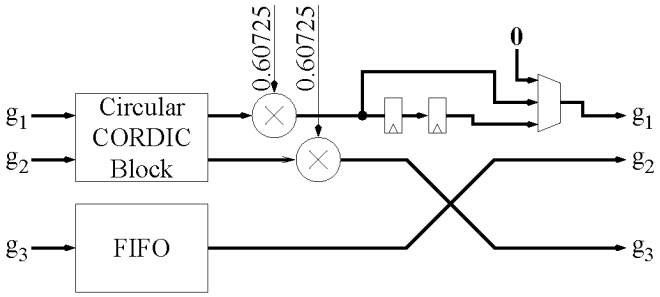

Figure 4.13: Simplified processing element for $N_D = 1$, $N_O = 2$, real numbers, white noise, two passes per iteration

4.3.6. Speeding up the Computation

Each pass through a CORDIC cell zeros one of the two input elements. A CORDIC technique exists that can zero multiple elements per pass at the expense of hardware complexity. It is called Householder CORDIC [62]. Its usefulness depends heavily on the target technology being able to implement multi-input adder/subtractors that are faster than a tree of two-input adder/subtractors.

4.4. Architecture for HIPERLAN

To illustrate the feasibility of the proposed architecture, several Equalizer configurations for the HIPERLAN I [63] system are discussed.

According to the HIPERLAN I specification [63], a terminal must be able to start transmitting a response packet 512 bit periods or $25.6\mu s$ after the arrival of a packet at its antenna. Clearly, only a fraction of this time can be allocated to the computation of the DFE coefficients. In [10] it was hypothesized that a DFE might be too complex to implement. The filtering operation itself is no problem. The feedforward filter can be implemented for example by four multipliers operating at 60 MHz, three times the HIPERLAN I bit rate. The feedback filter consists of only $N_b = N_f - 1 = 11$ Adder/Subtractors, since $d_i \in \{\pm 1\}$. The difficulty lies in the computation of the DFE coefficients.

In this section, we assume a single output symbol spaced DFE with twelve feedforward taps, i.e. $N_D = 1$, $N_O = 1$ and $N_f = 12$. This equalizer performs well for typical indoor channels (50ns delay spread), and experiences only minor degradation for bad channels (150ns delay spread).

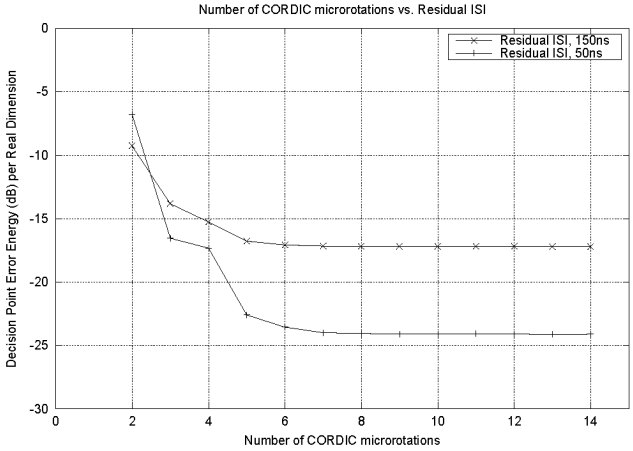

Figure 4.14: Computer simulation of residual ISI vs. number of microrotations

Furthermore, fully pipelined CORDIC elements are assumed, each CORDIC block executes eight microrotations and a data path width of twelve bits is assumed. The computer simulations of Figure 4.14 show that no further reduction in residual ISI energy at the decision point can be achieved by increasing the number of microrotations. Figure 4.15 contains a computer simulation plot of the residual ISI energy versus wordlength, assuming a perfect receiver automatic gain control (AGC). The infinite wordlength performance is reached at a datapath width of 8 Bits, and choosing 12 Bits results in approximately 20dB margin for imperfect AGC.

Since the processing elements have two CORDIC blocks and the deletion of a row in series, the total processing element latency is 17 clocks. The generator rows pass through $N_f N_O + 1 = 13$ processing elements. Table 4.2 summarizes the processing elements. In determining the implementation complexity the overall control circuitry, the microrotation direction register and XOR gate, the zeroing gates, and the constant coefficient multipliers

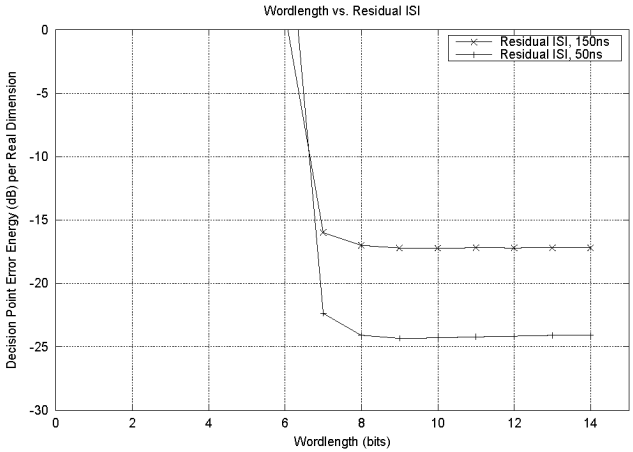

Figure 4.15: Computer simulation of residual ISI vs. wordlength

have been neglected, because their size is small compared to the CORDIC datapath. The constant coefficient multipliers from several or all PE's can be combined and can be implemented with few shift/adds. Shifts can be inserted to prevent excessive growth of the signal magnitude.

# of CORDIC blocks	4
# of Microrotations	8
Total PE Latency	17 clocks
Number of Adder/Subtractors	64
Number of wordlength sized Registers	66

Table 4.2: Summary of processing elements

Let us now consider two specific points in the design space. The minimum latency solution (Figure 4.16) uses a chain of two PE's and $6\frac{1}{2}$ passes through the chain. No FIFO is necessary.

The implementation complexity can be halved by using only one PE (Figure 4.17). Now a FIFO is necessary. It is assumed that the FIFO latency can be varied from 1 to 9.

The implementation numbers (area and power/energy consumption) include the CORDIC datapath only. The controller, the FIFO, and the mul-

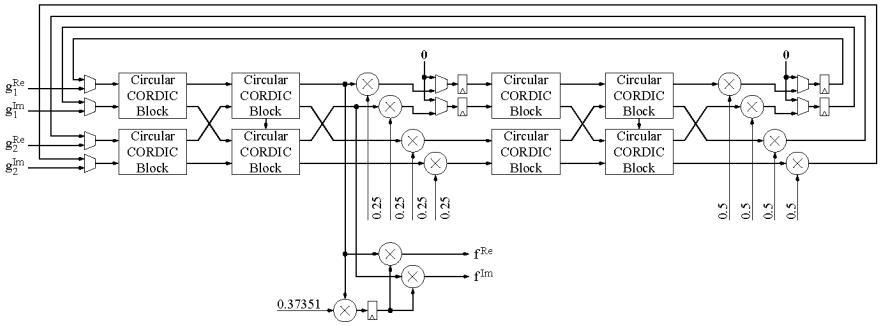

Figure 4.16: Minimum Latency Architecture for DFE coefficient computation

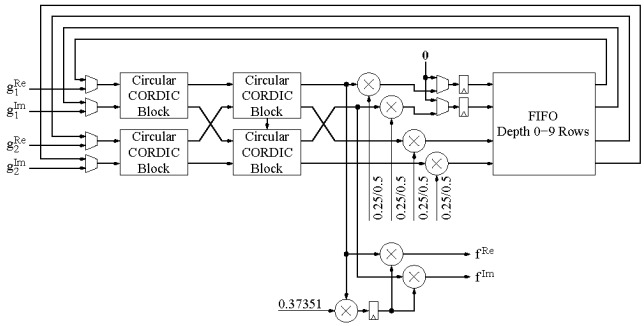

Figure 4.17: Single PE Architecture for DFE coefficient computation

tipliers are neglected. The constant coefficient multipliers that cancel the gain of the CORDIC blocks have been rearranged so that only one nontrivial constant coefficient multiplier remains in the output circuitry. The two variable×variable multipliers in the output circuitry do not need to be exact with respect to the common argument. It is sufficient to use an exponent detector and two shifters instead.

For reference, we also consider the architecture proposed by Al-Dhahir and Sayed [61]. The computationally most intensive task is the computation of the QR factorization of a band diagonal channel matrix.

The implementation numbers (area and power/energy consumption) include the CORDIC datapath only. The constant coefficient multipliers that cancel the CORDIC gain are not counted, although at least two would be necessary in the $\mathbf{r}$ loop. The controller, the maximum search and the back-substitution needed to compute the feedforward coefficients are neglected as well. Furthermore, we assume the same number of microrotations and the same data path width, which is justified by the computer simulation plots in [61].

Unfortunately, due to the feedback loop around the register $\mathbf{r}$, the QR factorization circuit cannot be fully pipelined. Therefore, the CORDIC blocks are implemented with only one microrotation slice that computes all microrotations sequentially.

Each CORDIC microrotation counts as one clock. No additional clock cycles are added for communicating the rotation directions in the horizontal direction.

Gate	Description	Area μm^2	Power $\mu W/MHz$
EO1	2-input XOR (1×)	146	0.883
FA1	Full-Adder (1×)	364	1.632
MU8	8:1 Multiplexer (1×)	619	1.318
DFS8	Scan D-Type Flip-Flop (1×)	382	1.933

Table 4.3: AMS 0.35μm standard cell library data

Table 4.3 lists area and power consumption data of some cells from the AMS 0.35μm standard cell library [64]. 1× drive strength gates are used because the datapath cell have a fan out of only one or two and the distances are small, leading to small capacitive loading. The power consumption numbers include a cell load capacity of 30fF. A ripple carry adder/subtractor may be built from a 2-input XOR gate and a full adder cell per bit. The shifter uses

Component	Area μm^2	Power $\mu W/MHz$	Prop. Delay ns
Adder/Subtractor	6120	30.180	6.5
Shifter	7428	15.816	1
Register	4584	23.196	1.2

Table 4.4: Area and power consumption of datapath components using the AMS 0.35μm standard cell process

a 8:1 multiplexer per bit. Table 4.4 lists the area and power consumption of wordlength sized components needed in the DFE computation datapath.

	Minimum Latency	Single PE	Al-Dhahir et al. [61]	Unit
Figure	4.16	4.17	4.1	
PE	2	1	—	
CORDIC blocks	8	4	1740	
Latency	221	270	568	clocks
Adder/Subtractors	128	64	3480	Wordlength sized
Shifter	—	—	3480	Wordlength sized
Registers	132	66	3480	Wordlength sized
Maximum FIFO depth	—	9	—	
Silicon cell area	1.39	0.69	63.10	mm^2
Power consumption	6.9	3.5	240.8	mW/MHz
Energy consumption	1.5	0.9	136.8	μJ/computation

Table 4.5: Summary of three HIPERLAN architectures
The silicon area and power/energy consumption numbers are approximate and derived from the AMS 0.35μm 3.3V standard cell process data book [64]. An activity factor of 100% and a load of 30fF is assumed.

The results from table 4.5 indicate that the DFE is indeed feasible for HIPERLAN. The proposed architectures are more than twice as fast and require more than ten times less silicon area and energy than [61]. They are therefore well suited to cost and power constrained terminals.

Assuming a clock frequency of 100 MHz, which is possible with the 0.35μm standard cell process, the computation of the DFE FFF coefficients requires only $2.7\mu s$ for the single PE architecture, which clearly demonstrates the feasibility of the proposed architecture.

5

FPGA DSP Core

The dedicated hardware architectures presented in chapter 4 can compute the DFE coefficients very quickly. They are however inflexible. If more time is available for the DFE coefficient computation, a general purpose DSP core offers a more flexible solution. Besides computing the DFE coefficients, it can be used to perform other modem functions as well.

This chapter discusses the motivation and the methodology of designing a high performance DSP core to be implemented on an FPGA.

Clock cycle counts are given for computing the feedforward coefficients. These results are representative for the class of single-MAC DSPs. The FPGA DSP core is also compared to dedicated hardware for computing DFE feedforward filter coefficients.

5.1. Motivation

For Powerline communications, the moderate channel dispersiveness leads to a moderate number of feedforward taps N_f and thus to moderate complexity. Furthermore, since no standard protocols exist so far, one may design a channel access protocol that can tolerate some decoding latency. In this case, a standard DSP core is sufficient for implementing the computation of the DFE

coefficients.

A DSP core puts most complexity into the software. But software can easily be changed late in the design cycle or even in the final product. Furthermore, it allows some flexibility to accomodate unforseen problems.

A DSP core can also perform other tasks than just computing the DFE coefficients, such as computing the CIR estimates, a frequency error estimate, AGC control signals, etc.

Commercial DSP cores and high performance RISC cores [65, 66] have been synthesized for an FPGA target. However, since the target architecture was an afterthought, performance has been disappointing.

High speed FPGA synthesizable DSP cores allow real-time testing of the modem on an FPGA prototype. Subsequent integration of the complete modem into an ASIC becomes much less risky, making first time right realistic.

Furthermore, a DSP core with the VHDL source code available allows a much higher degree of customizability than commercial cores.

The presented DSP core was developed for the Virtex family of FPGAs from the market leader Xilinx. Virtex is the current high performance FPGA family of Xilinx. The widely used synthesis tool from Synopsys Inc. has been used.

5.2. Designing a DSP Core for FPGA

Figure 5.1 shows a simplified diagram of the Xilinx Virtex configurable logic block (CLB) slice. The FPGA consists of a two dimensional grid of CLB slices, connected by a programmable interconnect structure. Other FPGA families have a different structure.

The programmable lookup tables (LUT) and the flip flops (FF) can be used to implement arbitrary logic functions. Additional special gates may be used to speed up common functions. `MUXCY` and `XORCY` speed up adder and subtractors by providing a special purpose fast carry chain. `MULT_AND` gates speed up multipliers, and `F5MUX` and `F6MUX` gates can be used to implement 4 input and 8 input multiplexers. To achieve a high clock frequency, these special gates must be used whenever possible. Contemporary synthesis tools however often cannot automatically synthesize these gates, so they must be instantiated manually.

The most important part of a DSP core is the execution unit. A fast clock frequency would be useless if even elementary operations took multiple clock cycles. On the other hand, additional pipeline stages in the instruction decoder only result in additional delay slots for control flow instructions, and since the DSP core supports zero overhead looping hardware, do not affect loop

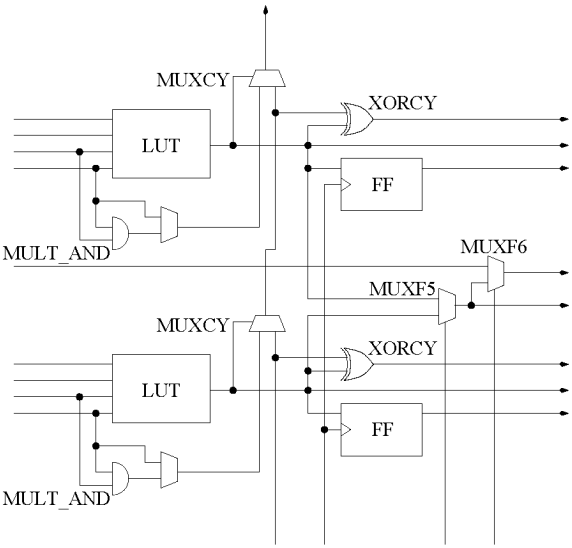

Figure 5.1: Simplified Xilinx Virtex CLB slice

performance. The zero overhead looping hardware automatically maintains the loop count and the loop begin and end addresses to obviate the need for explicit conditional jumps within counted loops. Therefore, the design should begin at the execution stage.

Figure 5.2 shows the block diagram of the DSP core. A modified harvard architecture is used. In order to sustain a single cycle throughput multiply accumulate (MAC, inner product), two busses to the 16 bit wide data memory are used. The program memory is 32 bit wide and a single port is used.

5.2.1. Execution Unit

Figure 5.3 shows the architecture of the execution unit.

Multiplier The central component of each DSP core is the parallel multiplier. Conventional DSP cores feature a single cycle MAC. Xilinx specifies the latency of a $16 \times 16 \rightarrow 32$ combinatorial multiplier on an XCV300-4 and XCV300-6 with 17ns and 7ns, respectively [67]. Realizing the multiplier together with input multiplexers and the final accumulator in a single cycle would result in an inacceptably high cycle time. Furthermore, most signal

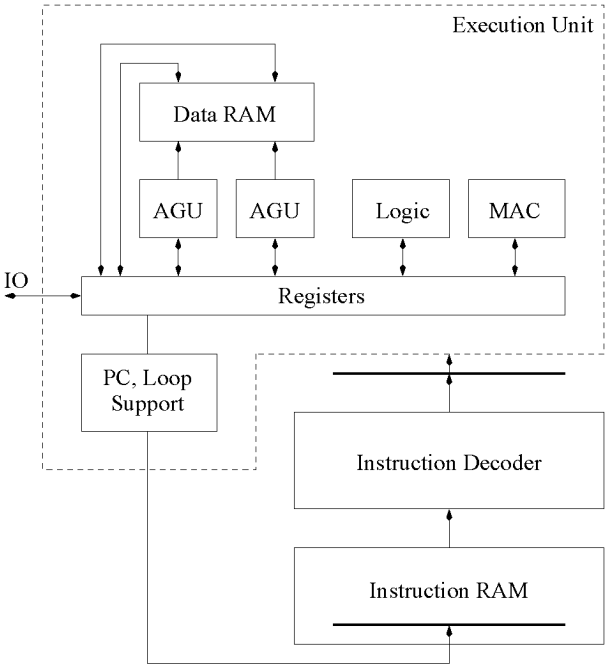

Figure 5.2: DSP core block diagram

processing tasks can hide some multiplier latency. Thus, the multiplier has been allocated three pipeline stages, and together with the final accumulate cycle the architecture features a four cycle latency MAC with one MAC per cycle throughput.

One would expect that designing a good multiplier for an FPGA is an easy routine task, especially since FPGA vendors advertise their products as being a suitable platform for digital signal processing. It turned out not to be the case.

First, the Synopsys FPGA Design Compiler (DC) was used to generate a combinatorial multiplier. Synopsys however did neither take advantage of the multiplier support gates of the FPGA, nor of the fast carry chain. The resulting multipliers were twice as big and twice as slow as necessary. Furthermore, automatic register balancing (moving registers around in combinatorial logic) did not work as expected.

Second, the Xilinx Coregen tool was used to generate a multiplier netlist. Coregen knows all FPGA specialities, but offers only an all or nothing approach to pipelining. The only options are a fully combinatorial multiplier array or pipeline registers after every stage in the adder tree. For a $16 \times 16 \rightarrow 32$ multiplier this would take five clock cycles. Also, Coregen uses relative location constraints for the logic elements. Experiments have shown that the overall execution unit performance is better without location constraints.

The final solution was to write a `Perl` script that generated a VHDL netlist of a Petzaris $17 \times 17 \rightarrow 32$ multiplier [38]. The netlist instantiated `MULT_AND` gates and carry chain multiplexers and had registers inserted at suitable places in the adder tree. The 17 bit input width supports a 16 bit data word width and signed/unsigned operation.

Commercial 16bit fixed point DSP cores allow their multiplier to be operated in integer mode or fixed point mode with 15 fractional bits. For the computation of the DFE coefficients, 11 fractional bits are better suited. This can easily be obtained by adjusting the wiring of the multiplier output multiplexer. This is an advantage of a VHDL DSP core.

Logic Unit The logic unit performs `AND`, `OR` and `XOR` operations and contains a seed table for reciprocal square root computations.

Address Generator Unit Since two operands are needed to sustain a single cycle throughput MAC operation, two addresses are needed as well. Therefore two address generators are provided. Each address generator comprises of two adder/subtractors to allow pre- and postmodify accesses.

DSPs usually provide special addressing modes for FIR filters and FFT computation, but since neither FIR filters nor FFTs are required for DFE coefficient computation, these addressing modes are not supported.

Many DSPs lack the addressing modes needed for efficient compiler support, especially the stack pointer offset addressing mode required to move data in and out of stack slots.

Registers Processors employ multiported static RAMs for the register file. Since the FPGA architecture only features RAMs with two ports, the registers have to be implemented with Flip-Flops.

A key aspect of high performance microprocessors is that data can be moved freely between registers and between registers and memory. This requires heavy multiplexing. On-chip tristate busses do exist on FPGAs, but are slow and should therefore not be used.

The FPGA architecture provides some support for efficient multiplexers. Two input, four input (using one `F5MUX` gate) and eight input (using two `F5MUX` gates and one `F6MUX` gate) multiplexers are supported. Since Synopsys DC was unable to automatically instantiate `F5MUX` and `F6MUX` elements, multiplexers were constructed manually. An implementation of the `F5MUX` and `F6MUX` using standard logic is available for functional simulation and ASIC synthesis.

The limited multiplexing capabilities limit the number of registers possible. There are five address registers `AR0`–`AR4`, where `AR0` is used as stack pointer by convention. `LB`, `LE` and `LC` are used by the zero overhead looping support. These registers are all 12 bits wide.

`Y0`–`Y3` are the data input registers, and `Z0`–`Z1` are the MAC result registers. Four input and two result registers have been provided to support complex inner products or double precision arithmetic. `Z2` is the result register for logic operations. All data registers are 16 bits wide. `Z0`–`Z2` are the only registers that cannot be loaded from memory or general registers, but only from a data input register. All other registers can be loaded from memory, immediate constants or other registers.

5.2.2. Instruction Decoder

Four data source registers for the multiplier are not enough to keep the four cycle latency multiplier busy with an interlocked pipeline. Experiments with out of order execution and register renaming according to the Tomasulo's approach [68, chapter 4.2] stressed the FPGA multiplexing and routing capability too much and lead to an explosion of the cycle time. Thus, a non

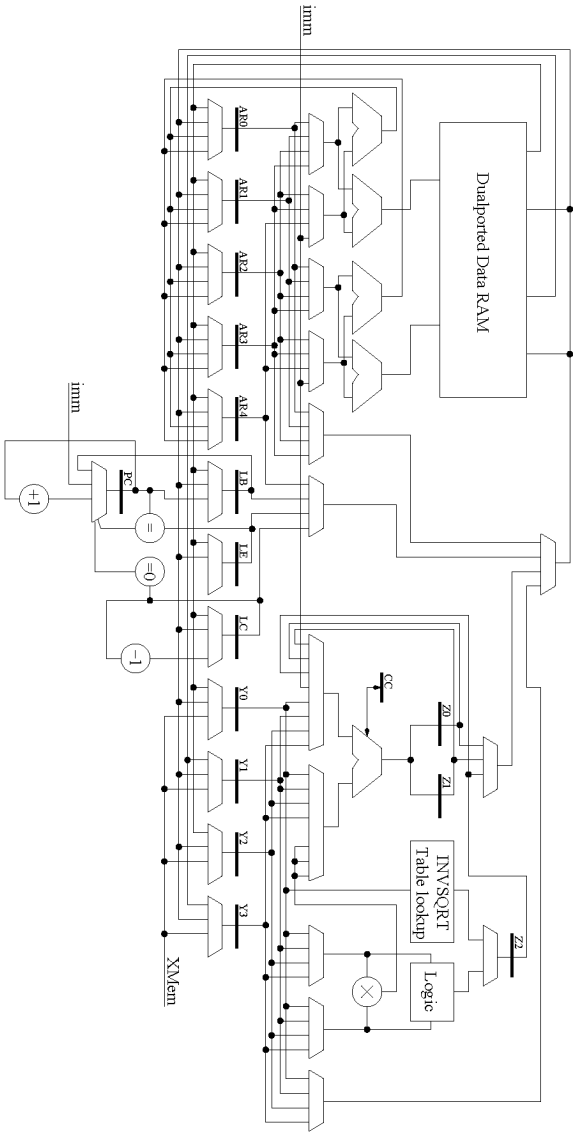

Figure 5.3: DSP execution unit architecture

interlocked in order pipeline is used. Therefore, it is the responsibility of the programmer to use values only when they are ready. Since all instructions have a fixed latency, these semantics are called non uniform access latency (NUAL) with equal semantics (EQ) [69].

It is tempting to use a Very Long Instruction Word (VLIW) [69, 70, 71] instruction encoding to get as much as possible out of the execution stage and to avoid instruction set design. VLIW instruction sets however lead to sparse instruction encoding with many NOPs und thus wasted program memory. Since FPGAs contain only relatively little on-chip RAM blocks (80kBits for the XCV400) VLIW instruction sets are unsuitable. Some ideas from VLIW research such as positional encoding have been utilized nevertheless.

The instruction word is split into two halves. The lower half encodes a data ALU operation, while the upper half specifies a store, a load, two loads with restricted addressing modes and target registers, or a register to register copy. Both halves may be used to encode an immediate value for the other half.

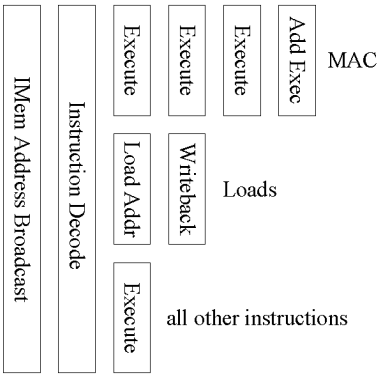

Figure 5.4: DSP core pipeline

Figure 5.4 shows the pipeline of the DSP core. One full pipeline stage had to be used just to broadcast the program address from the program counter register to the distributed program memory. Distributing the address takes more than 8ns.

A design goal of the DSP core was that as many instructions as possible execute in a single cycle. The exceptions are as follows:

- Load instructions have two cycle latency. This is due to the pipelined architecture of the Xilinx Block RAMs.

- MAC operations have four cycle latency. Note that the add operand is only read at the start of the last cycle, so MAC operations can be issued back-to-back (consecutively).
- Control flow instructions such as jumps and block repeats have two delay slots. The delay slots are executed unconditionally.

5.2.3. Development Tools

An assembler and a linker have been written. A cycle true simulator allows the DSP core to be embedded into the simulation environment of the complete modem, and allows the verification of the output of a VHDL simulator. Tools to convert an object file into the format required by VHDL simulation and implementation tools also exist.

The DSP core has been designed make efficient compiled code possible. A GNU Compiler Collection (GCC) backend [72] has been written. The GNU compilers have been chosen because their source code is available and they are the most complete and widely supported compiler. Since the GNU compilers already support a wide range of different target architectures, including DSPs, no modifications to the compiler proper had to be made. It was sufficient to add a backend for the DSP core.

The compiler supports reordering code to take advantage of delay slots and latency slots. If it does not find independent operations to fill latency or delay slots, it fills them with NOPs to ensure correctness.

An iterative development methodology has proven to be valuable. First, the core architecture was sketched and verified that it could roughly meet the cycle time goal. Then, the compiler backend was developed to find the problem spots of the architecture. Then the architecture was modified to avoid these problem spots and refined. Then the modifications to the compiler backend have been made, and so forth. This ensures that the resulting architecture balances the needs of the hardware and the compiler.

5.2.4. Reciprocal Square Root

The Cholesky $\mathbf{L}\mathbf{L}^H$ factorization requires the computation of reciprocal square roots ($1/\sqrt{z}$). Hardware support to accelerate the computation of the reciprocal square root is therefore advantageous. Newton-Raphson iteration [38, Chapter 21.5] has been chosen; accurate results can be obtained after only a few iterations, and very little additional hardware, namely the seed table, is required. The reciprocal square root seed table lookup instruction is an example for an application specific instruction, [73].

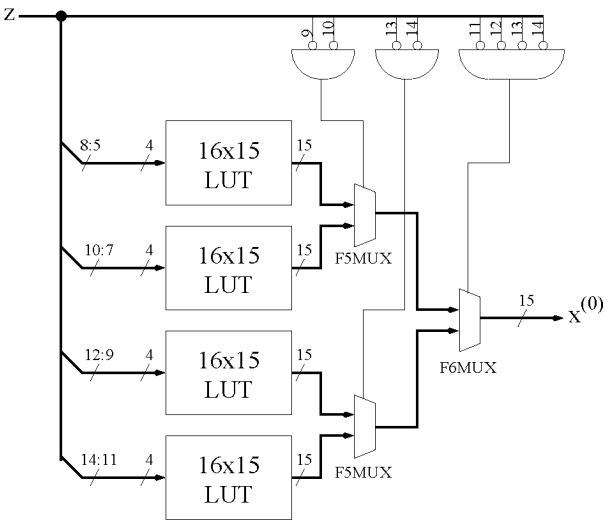

Figure 5.5: Reciprocal square root seed table

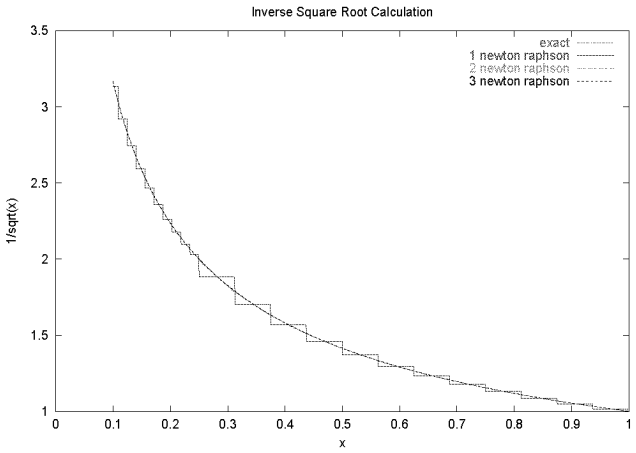

Figure 5.6: Reciprocal square root, below 1

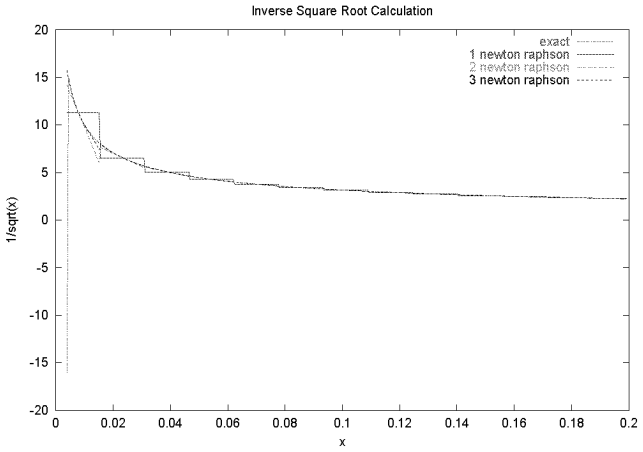

Figure 5.7: Reciprocal square root, below 0.2

It is advantageous to directly compute the reciprocal square root instead of splitting it into a square root operation and a division. A fast-converging Newton-Raphson recurrence does exist for the reciprocal square root, and both the square root and the reciprocal can easily be computed from the reciprocal square root; the converse is not true.

The root of the function $f(x) = 1/x^2 - z$ shall be found, where z is the value of which the reciprocal square root shall be computed. A root of $f(x)$ is at $x = 1/\sqrt{z}$. The recurrence equation is $x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}$, and with $f'(x) = -2/x^3$ one obtains

$$x^{(i+1)} = \frac{3}{2}x^{(i)} - \frac{1}{2}z{x^{(i)}}^3. \tag{5.1}$$

Unlike for floating point number representation, the order of the multiplications in (5.1) is important for fixed point computations to avoid overflows at internal nodes. The domain and range of the reciprocal square root computation are then limited only by the representability of the values with the chosen fixed point format.

Seed lookup table The seed table provides an $x^{(0)}$ from z that is close to $1/\sqrt{z}$ to ensure fast convergence. The idea is to first transform the fixed point number into a floating point number, feed the mantissa into the seed table,

and then scale the seed table output back into a fixed point number. This can be achieved by replicating the scaled seed table with different input shifts, and then selecting which table to use according to the exponent. This implementation fits the Xilinx Virtex architecture very well, see Figure 5.5. The multiplexer support circuitry is used for selecting the LUTs, thus achieving fast lookup speed.

Figures 5.6 through 5.7 illustrate the accuracy of the complete reciprocal square root computation. Apart from a spike below 0.02, the curve tightly matches the exact curve after only two iterations. The cholesky factorization uses the reciprocal square root operation for the computation of the diagonal elements of $\mathbf{L}$ (section 3.2.3). According to section 3.2.6, $l_{i,i} \geq \sqrt{N_0}$ and therefore the argument of the reciprocal square root $\geq N_0$. So for an SNR of 17dB or less, no distortions result due to the inaccuracy of the reciprocal square root below 0.02. For a larger SNR, distortions may occur, depending on the channel coefficients.

5.3. Results

In this section, the FPGA DSP core is compared to competing high performace processor cores for FPGAs and to dedicated DFE coefficient computation hardware.

5.3.1. Comparison to other FPGA Processors

Table 5.1 lists the main implementation results. The performance goal was reached; the DSP core exceeds 60 MIPS on the slowest Xilinx Virtex device. The large area of 933 slices (approximately 100k gates) is due to the lack of memory blocks with a large number of ports that could be used to implement the register file. Therefore, the register file had to be implemented with flip flops and multiplexers, which is the single largest contributor to core size. The implementation time includes the creation of the synthesizable VHDL code, the toolchain including C-Compiler, and the firmware for computing the 1-dimensional DFE coefficients.

Table 5.2 compares the design with commercially available FPGA microprocessor cores. There is a growing interest to provide microprocessor cores for FPGAs by the FPGA vendors, like Altera, and microprocessor core vendors, like Lexra, for prototyping. The table only lists cores potentially suitable for signal processing; simple 8bit cores have been omitted.

Table 5.2 indicates that this FPGA DSP core has a significantly higher operating frequency than the competing microprocessors. Nios has a selectable

Operating Frequency XCV400-4	62 MHz
Operating Frequency XCV400-6	80 MHz
Number of CLB Slices	910 (18% of XCV400)
Number of Block RAM's	11 (55% of XCV400)
Implementation time	≈2 Manmonth

Table 5.1: FPGA DSP core implementation results

Property	This work	Altera Nios 16b	Lexra LX4080P	ARC core w/ DSP ext
Datapath width	16	16	32	32
FPGA	Xilinx XCV400-6	Altera EP20K100E-1	Altera 10K200E	Xilinx XCV400E-8
Utilization	18%	26%	50%	100%
Clock Frequency	80 MHz	50 MHz	33 MHz	23 MHz
Instruction Set	proprietary	proprietary	MIPS-I	ARC
Source		[74]	[65]	[75]

Table 5.2: FPGA RISC/DSP cores

datapath width of 16 or 32bits. The numbers in the table are for the 16bit version. The detailed architecture and instruction set of Nios has unfortunately not been publicly disclosed. Therefore, the suitability for DSP tasks is unknown. The low area (approximately 25k gates) however suggests that the register file is indeed implemented as a RAM block. The limited number of ports of FPGA RAM blocks means however the Nios will likely require multiple clock cycles for many instructions.

The ARC core is a 32 bit RISC processor with DSP extensions, mainly a 24×24 bit multiplier. This processor is especially interesting, as the implementation numbers given in [75] are for a very similar device to the one used for the FPGA DSP. The implementation numbers may therefore be compared directly. These data underscore that a core not designed with FPGA idiosyncrasies in mind performs poorly on FPGAs.

5.3.2. Comparison of Different DFE Coefficient Computation Algorithms

In this section, the implementation results of three different algorithms for computing the Decision Feedback Equalizer Feedforward coefficients on the FPGA DSP core are discussed. The equalizer parameters are $N_D = 1$, $N_O = 2$, real numbers and white noise. These parameters are chosen because they

represent a common case in practice, namely the symbol spaced equalizer for BPSK and MSK.

	Program Words		
Task	Chol	DSFact	DSSol
Matrix $\mathbf{G}$ setup	—	41	42
Displacement recursions	—	231	196
Generator ($\mathbf{GB}^H$) multiplication	—	—	68
Matrix $\mathbf{A}$ Computation	110	—	—
Cholesky Factorization	113	—	—
Back Substitution	92	93	—
Total	315	365	306

Table 5.3: Code size

Size	Matrix Comp	Cholesky Fact	Back Subst	Total
$N_f = 4$	387	1473	633	2493
$N_f = 6$	739	3013	956	4708
$N_f = 8$	1203	5177	1333	7713
$N_f = 10$	1779	8029	1740	11548

Table 5.4: Execution time (number of cycles) of cholesky factorization

Size	Matrix Setup	Displacement Recursions	Back Subst	Total
$N_f = 4$	110	2456	655	3221
$N_f = 6$	156	4020	1023	5199
$N_f = 8$	202	5808	1423	7433
$N_f = 10$	248	7817	1858	9923

Table 5.5: Execution time (number of cycles) of displacement structure factorization

"Cholesky" and "Chol" denotes the factorization of the DFE matrix using the Cholesky algorithm (section 3.2.3) followed by back substitution.

Size	Matrix Setup	Displacement Recursions	Generator Multiplication	Total
$N_f = 4$	139	2739	55	2933
$N_f = 6$	193	4823	75	5091
$N_f = 8$	247	7387	95	7729
$N_f = 10$	301	10431	115	10847

Table 5.6: Execution time (number of cycles) of displacement structure solution

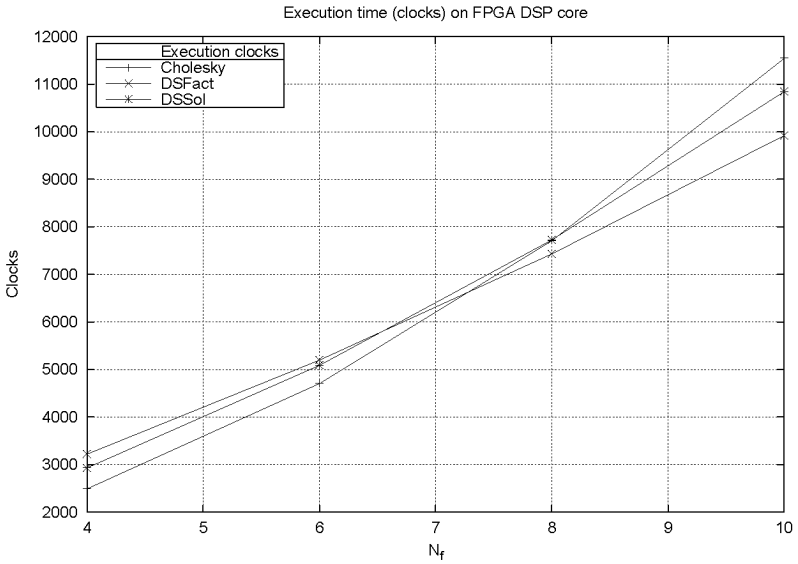

Figure 5.8: Execution time (number of cycles) for different algorithms on FPGA DSP core

"Displacement Structure Factorization" and "DSFact" denotes computing the cholesky factors using the Displacement Structure Theory based algorithm of section 3.2.4, followed by back substitution. "Displacement Structure Solution" and "DSSol" denotes the back substitution free algorithm of section 3.2.5.

All algorithms have been implemented in C with handoptimized innermost loops in assembler. DSFact and DSSol use two passes per iteration over the generators, each operating on two columns, instead of one pass with a 3×3 Θ matrix. The 3×3 matrix would not fit into the register file, while the 2×2 matrix does. All important innermost loops operate at the maximum throughput of one multiplication per clock cycle.

As table 5.3 shows, there are no significant differences in code size of the three algorithms.

	Cholesky			DSFact			DSSol		
N_f	Muls	Clocks	%	Muls	Clocks	%	Muls	Clocks	%
4	248	2493	9	424	3221	13	952	2933	32
6	652	4708	14	876	5199	17	2004	5091	39
8	1328	7713	17	1488	7433	20	3440	7729	45
10	2340	11548	20	2260	9923	23	5260	10847	48

Table 5.7: Number of datapath multiplications versus number of clocks

Tables 5.4 to 5.6 and Figure 5.8 list the number of clock cycles required for the different subtasks of the three algorithms and the total clock count for different N_f. Cholesky wins over the $O(N^2)$ algorithms DSFact and DSSol for $N_f < 8$ (DFE matrices smaller than 16×16 elements) even though it is $O(N^3)$. Furthermore, data memory requirements of Cholesky grow with N_f squared, while DSFact and DSSol data memory size grows only linearly with N_f.

DSSol wins over DSFact for $N_f < 7$ even though it requires the computation of more than twice the number of multiplications and additions than DSFact. The reason for this is the higher innermost loop iteration counts of DSSol over DSFact and Cholesky. While the iteration count lies in the range $0 \ldots 2N_f$ in Cholesky and DSFact, it lies in the range $2N_f + 1 \ldots 4N_f$ in DSSol. DSSol can therefore amortize loop setup and address computation costs over a larger number of loop iterations.

Table 5.7 compares the number of datapath multiplications with the total number of clocks required to execute the different algorithms. Multiplication has been chosen as the reference because the number of additions is almost

the same, in fact most additions can be fused together with a multiplication into a multiply accumulate (MAC) instruction. Again, multiplier utilization of DSSol is much higher compared to DSFact and Cholesky.

The results of this section are representative for the class of single multiplier DSPs, because the FPGA DSP has an architecture similar to many commercial DSPs, such as the Texas Instruments TMS320C3x [76] and the Analog Devices ADSP-21xx [77] families. The main difference to commercial DSPs is the higher multiplier latency required to achieve a low cycle time. Most of the time, the multiplier latency can be hidden. The most notable exception is the Newton-Raphson iteration that computes the reciprocal square root. Even on the FPGA DSP, however, where 40 clock cycles are required to compute the 10 multiplications needed for 2 iterations, reciprocal square roots account for less than 10% of the total execution time.

5.3.3. Comparison to Dedicated DFE Coefficient Computation FPGA Hardware

In this section, the FPGA DSP core implementing the DFE coefficient computation ($N_D = 1$, $N_O = 2$, real numbers, white noise) is compared to a dedicated hardware architecture. The hardware architecture uses a single processing element containing two CORDIC blocks as in Figure 5.9. The datapath width is 16bits, and the number of microrotations per CORDIC block is 8. A FIFO may be necessary; it is assumed to have at least one cycle latency. Two additional cycles of latency are introduced by the constant multipliers that remove the gain of the CORDIC blocks; they are implemented using four shift/add terms each. The first generator row leaving the processing element is furthermore discarded. The total number of clock cycles per iteration is therefore $2 * (8 + 2) + 1 + 1 = 22$ clocks.

Property	FPGA DSP	Dedicated FPGA Hardware
Clock Rate XCV400-4	62.7 MHz	73.7 MHz
Number of CLB Slices	910	498
% of CLB slices of XCV400	18	10
Approx. Gate Equivalents	100k	24k

Table 5.8: FPGA DSP core versus dedicated FPGA hardware

Table 5.8 lists the features of the two architectures. The simple structure of the dedicated hardware architecture results in a higher clock frequency

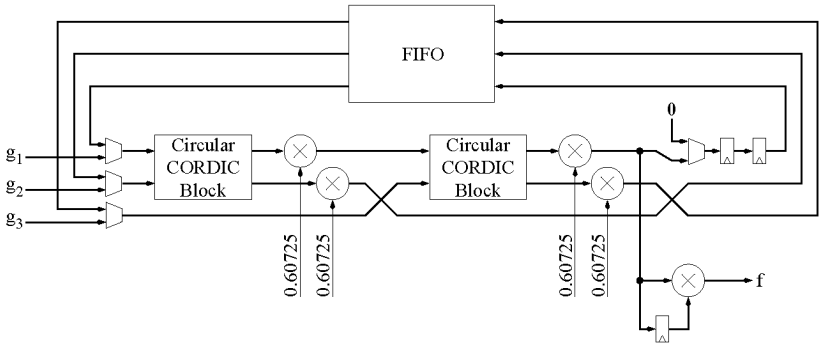

Figure 5.9: Dedicated DFE computation hardware

compared to the FPGA DSP core. The gate count of the FPGA DSP does not include the program and data RAMs, only the core logic. The gate count for the dedicated hardware includes everything except the controller, whose area contribution is negligible.

	FPGA DSP		dedicated hardware	
N_f	Clocks	Time	Clocks	Time
4	2493	$39.8\mu s$	226	$3.1\mu s$
6	4708	$75.1\mu s$	324	$4.4\mu s$
8	7433	$118.5\mu s$	476	$6.5\mu s$
10	9923	$158.3\mu s$	692	$9.4\mu s$

Table 5.9: FPGA DSP clock cycles and time versus dedicated hardware architecture

Table 5.9 compares the number of clocks and the execution time of both architectures. The number of clock cycles required for the dedicated hardware architecture can be computed with

$$N_{CLK} = \sum_{i=2N_f+1}^{4N_f+1} \max(i, LAT_{PE}) + LAT_{LASTPE} + 2N_f, \qquad (5.2)$$

where $LAT_{PE} = 22$ denotes the processing element latency, and $LAT_{LASTPE} = 20$ denotes the latency of the last processing element, that does not need to remove the first line (Figure 5.9) and whose output does not go through the FIFO.

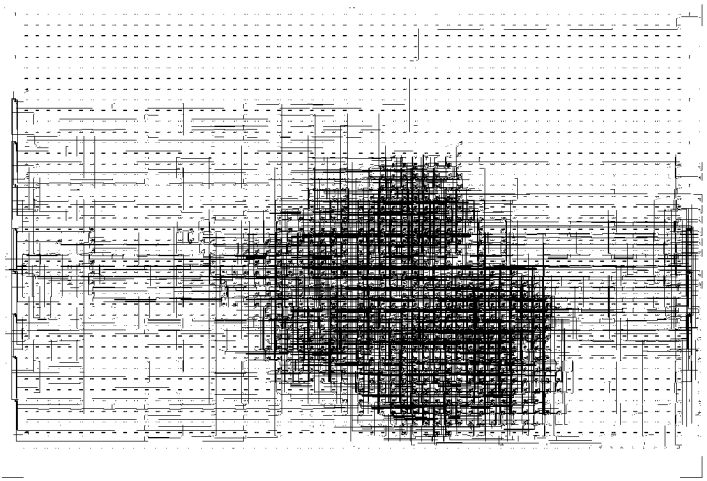

Figure 5.10: FPGA die plot of DSP core
The tiny rectangles depict the 40×60 Configurable Logic Block (CLB) array of the XCV400 device.

Figures 5.10 and 5.11 show FPGA die plots of both architectures. The plots have been made with Xilinx' `fpga_editor` version 3.1i. For the FPGA DSP, the XCV400 device size is inconvenient. The XCV400 has 10 block RAMs on the eastern and western edge each. Since the DSP uses 11 block RAMs, RAMs on both edges are needed. If a larger device is selected, the RAMs of only one edge can be used, leading to smaller distances and a faster design. If a smaller device is chosen, the distance between the edges becomes smaller, the design therefore faster.

To conclude, the dedicated hardware solution is more than 10 times faster and requires about $\frac{1}{2}$ the number of CLB's the FPGA DSP requires in the N_f range suitable for most communication systems. The main disadvantage of the dedicated hardware solution is its inflexibility – the DSP may perform other tasks such as receiver parameter estimation when not busy with the DFE coefficient computation.

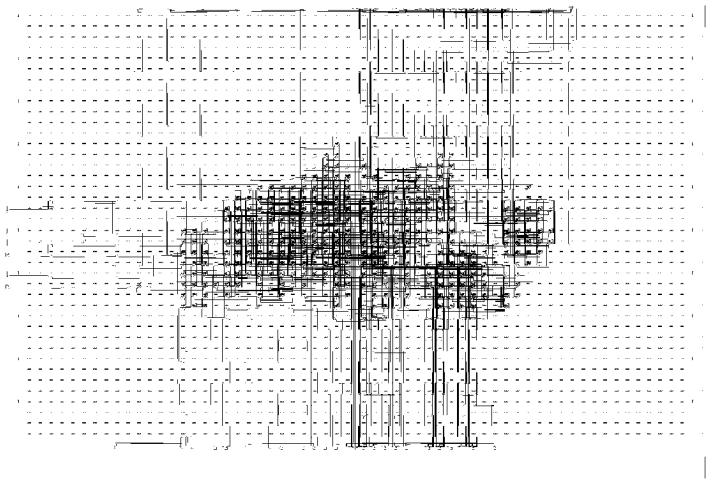

Figure 5.11: FPGA die plot of dedicated hardware
The tiny rectangles depict the 40×60 Configurable Logic Block (CLB) array of the XCV400 device.

6

Outlook

In the past couple of years, the main focus of transmission system designers has been to avoid frequency selective fading. The trend toward higher speed transmission systems will force designers to reconsider selective fading.

6.1. OFDM Systems over Higly Dispersive Channels

Instead of occupying a large bandwidth with a single high rate carrier, orthogonal frequency division multiplex (OFDM) has been used to divide the available bandwidth into a large number of small bandwidth subchannels. Within a single subchannel, fading could be treated as nonselective fading and therefore making equalization trivial. Systems such as Digital Audio Broadcast (DAB) and Digital Video Broadcast (DVB) use more than 1500 subchannels which are about 1kHz wide. A large number of subchannels is unfortunately problematic, as it leads to a high peak to average power ratio and increased phase noise susceptibility. Some systems therefore have to use a smaller number of subchannels, resulting in shorter OFDM symbols.

In order to keep successive OFDM symbols orthogonal, each OFDM symbol is cyclically extended. This extension is termed guard interval and its duration must be at least as long as the channel length for the OFDM symbols

to be orthogonal. For OFDM systems over highly dispersive channels, this results in a large fraction of the total time being "wasted" for the transmission of the guard intervals, lowering the user bitrate. Some OFDM systems therefore chose a guard interval shorter than the channel duration, leading to intersymbol (ISI) as well as interchannel (ICI) interference. High performance receivers for such OFDM systems will have to cope with the ISI and the ICI. One possibility is the use of Decision Feedback Equalizers in the frequency domain (i.e. after the FFT). Treating all subchannels as a single multiple input multiple output (MIMO) DFE will likely be too complex. Since the ICI is limited only to close subchannels, multiple DFEs, each dealing only with a small number of adjacent subchannels may be used. Since multiple DFE coefficient computations need to be performed, the architecture of Figure 4.4 should be well suited, because a new computation may be started after only a few clock cycles, while the previous one is still being processed.

6.2. Multiuser Detection for Wideband CDMA

In Code Division Multiple Access (CDMA) systems, all users transmit in the same frequency band, but the transmission of every user is scrambled with a different signature sequence so that the receiver can separate the different transmissions. The transmit waveforms of different users are not exactly orthogonal, however, for example because of nonideal synchronisation or even nonorthogonal signature sequences. First generation cellular CDMA systems decode each user separately, treating the effects of all other users as unwanted noise.

Ever higher bitrates expected by the users and the limited channel bandwidths lead to wideband CDMA proposals with a very low coding gain, leading to large nonorthogonal components. In order to achieve satisfactory channel capacities, receivers may no longer decode each user separately. Multiuser Detectors (MUDs) are needed. One proposed multiuser detector is the Decision Feedback Detector (DFD) [24].

6.3. Continuous Systems

In systems operating continuously, the channel will not change arbitrarily between two packets or frames, there will be significant correlation between subsequent channel impulse responses. In such a setup, iterative inversion methods such as Gauss-Seidel or Jacobi iteration may still be attractive. The solution of the previous timeframe may be used as the starting vector of the next timeframe.

A

Utilized Parameters and their Symbols

$\mathbf{0}$	All zero Matrix/Vector
$\mathbf{A}$	DFE Key Equations Matrix, $\mathbf{A} = \breve{\mathbf{C}} + \breve{\mathbf{N}}$
$\mathbf{A}_1$	Order-reduced version of $\mathbf{A}$
$\tilde{\mathbf{A}}_1$	Zero padded version $\mathbf{A}_1$
b_i	Feedback filter taps (single output transmitter)
$\mathbf{B}_i$	Feedback filter taps (multiple output transmitter)
$\mathbf{B}$	Displacement Structure generator
$\mathbf{B}_1$	Displacement Structure generator, common part with G
c_i	Channel taps (single output channel)
$\mathbf{c}_i$	Channel taps (multiple output channel, single output transmitter)
$\mathbf{C}_i$	Channel taps (multiple output channel, multiple output transmitter)

$\bar{\mathbf{c}}$	Channel taps stacked above each other (single output transmitter)
$\bar{\mathbf{C}}$	Channel taps stacked above each other (multiple output transmitter)
$\breve{\mathbf{C}}$	Channel dependent part of $\mathbf{A}$
$\breve{\mathbf{C}}_{(i,j)}$	i,j-th block element of $\breve{\mathbf{C}}$
d_i	Transmitted symbol (single output transmitter)
$\mathbf{d}_i$	Transmitted symbols (multiple output transmitter)
$\tilde{d}_i$	Decision point signal (single output transmitter)
$\tilde{\mathbf{d}}_i$	Decision point signal (multiple output transmitter)
$\hat{d}_i$	Receiver decision of d_i (single output transmitter)
$\hat{\mathbf{d}}_i$	Receiver decision of $\mathbf{d}_i$ (multiple output transmitter)
$\mathbf{D}$	Diagonal part of Cholesky factorization
e_i	Decision point error signal (single output transmitter)
$\mathbf{e}_i$	Decision point error signal (multiple output transmitter)
e_i^u	Decision point error signal of unbiased DFE (single output transmitter)
$\mathbf{e}_i^u$	Decision point error signal of unbiased DFE (multiple output transmitter)
$\mathbf{f}_i$	Feedforward filter taps (single output transmitter)
$\mathbf{F}_i$	Feedforward filter taps (multiple output transmitter)
$\bar{\mathbf{f}}$	Feedforward filter taps stacked above each other (single output transmitter)
$\bar{\mathbf{F}}$	Feedforward filter taps stacked above each other (multiple output transmitter)
$\mathbf{F}_1$	Left Displacement Structure operator
$\mathbf{F}_2$	Right Displacement Structure operator
$\mathbf{G}$	Displacement Structure generator
$\bar{\mathbf{G}}$	Zero padded version of $\mathbf{G}_1$
$\mathbf{G}_1$	Displacement Structure generator, common part with B or order reduced version of $\mathbf{G}$
$\mathbf{g}$	Row of $\mathbf{G}$
$\bar{\mathbf{g}}$	Zero padded version of $\mathbf{g}$
$\mathbf{g}_{pvt}$	Pivoting Row of $\mathbf{G}$
$\bar{\mathbf{g}}_{pvt}$	Zero padded version of $\mathbf{g}_{pvt}$
i	Time index
$\mathbf{I}$	Identity Matrix

$\mathbf{J}$	Signature Matrix
$\mathbf{L}$	Cholesky Factor
$\mathbf{n}_i$	Channel noise
$\mathbf{N}$	Noise dependent part of $\mathbf{A}$
$\mathbf{N}_{(i,j)}$	i,jth block element of $\mathbf{N}$
N_{i-j}	Noise Power $\mathbf{N}_{(i,j)} = N_{i-j}\mathbf{I}$
N_b	Number of Feedback coefficients
N_D	Number of Decision Feedback Detector outputs
N_f	Number of Feedforward coefficients
N_O	Number of Channel outputs
$\mathbf{r}_i$	Received signal vector (channel outputs)
$\mathbf{Q}$	Unitary factor of QR factorization
$\mathbf{R}$	Bordered version of $\mathbf{A}$; Triangular factor of QR factorization
$\mathbf{S}$	Schur Complement
$\mathbf{Z}$	Displacement Structure operator (Hermitian symmetric case), lower shift matrix
$\boldsymbol{\alpha}$	DFE Bias
Γ	Multiplier of $\mathbf{B}$
Δ	Decision delay
Θ	Multiplier of $\mathbf{G}$
τ	Channel delay spread normalized to symbol rate

B

Abbreviations

ASIC	Application Specific Integrated Circuit
BPSK	Binary Phase Shift Keying
CATV	Cable Television
CDMA	Code Division Multiple Access
CIR	Channel Impulse Response
CLB	Configurable Logic Block
CORDIC	COordinate Rotation on DIgital Computers
DC	Design Compiler, a VHDL synthesis tool from Synopsys, Inc.
DFD	Decision Feedback Detector
DFE	Decision Feedback Equalizer
DS-CDMA	Direct Sequence Code Division Multiple Access
DSP	Digital Signal Processor
ETSI	European Telecommunication Standards Institute
FIR	Finite Impulse Response Filter
FPGA	Field Programmable Gate Array
HIPERLAN	High PERformance Radio Local Area Network
ICI	Interchannel Interference

ISI	Intersymbol Interference
LUT	Lookup Table
MAC	Multiply Accumulate
MAC	Medium ACess layer/protocol
MIMO	Multiple Input Multiple Output
MIPS	Million Instructions Per Second; also a microprocessor intellectual property company
MLSD	Maximum Likelihood Sequence Detection
MMSE	Minimum Mean Square Error
MSE	Mean Square Error
MSK	Minimum Shift Keying
MUD	Multi User Detection
MUX	Multiplexer
NUAL	Nonuniform Access Latency
OFDM	Orthogonal Frequency Division Multiplex
PERL	Practical Extraction and Report Language
PLC	Powerline Communications
PSP	Per Survivor Processing
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SIMD	Single Instruction Multiple Data
SNR	Signal to Noise Ratio
TDMA	Time Division Multiple Access
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLIW	Very Long Instruction Word
WLL	Wireless Local Loop
ZF	Zero Forcing

Bibliography

[1] H. L. van Trees, *Detection, Estimation and Modulation Theory*. New York: Wiley, 1968.

[2] R. Sietmann, "Monopolfragen: das Wirtschaftsministerium und die Telekom-Regulierung," *c't Magazin für Computertechnik*, p. 57, Mai 2000.

[3] J. Aldis, A. Väisanen, and U. Lott, "WAND Deliverable 2D9: Results of outdoor measurements and experiments for the WAND system at 5 GHz," tech. rep., The Magic WAND (Wireless ATM Network Demonstrator) AC085, December 1998.

[4] H. Meyr, "Algorithm Design and System Implementation for Advanced Wireless Communication Systems," in *International Zurich Seminar on Broadband Communications (IZS) 2000*, 2000.

[5] T. Sailer and G. Tröster, "Performance-driven design of high speed receivers for wireless indoor networks," in *ISCAS*, 1999.

[6] G. David Forney, Jr., "Maximum-Likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference," *IEEE Transactions on Information Theory*, vol. IT-18, pp. 363–378, May 1972.

[7] J. B. Anderson, "Sequential Coding Algorithms: A Survey and Cost Analysis," *IEEE Transactions on Communications*, vol. COM-32, pp. 169–176, February 1984.

[8] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications," in *Proc., IEEE Globecom Conference*, pp. 1680–1686, 1989.

[9] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, pp. 284–287, March 1974.

[10] N. Benvenuto, P. Bisaglia, A. Salloum, and L. Tomba, "Worst Case Equalizer for Noncoherent HIPERLAN Receivers," *IEEE Transactions on Communications*, vol. 48, pp. 28–36, January 2000.

[11] R. van Nee and R. Prasad, *OFDM for Wireless Multimedia Communications*. Artech House, 2000.

[12] J. William C. Jakes, ed., *Microwave Mobile Communications*. Wiley, 1974.

[13] N. Al-Dhahir and J. M. Cioffi, "Mismatched Finite-Complexity MMSE Decision Feedback Equalizers," *IEEE Transactions on Signal Processing*, vol. 45, pp. 935–944, April 1997.

[14] W. Liu, H.-P. Widmer, J. Aldis, and T. Kaltenschnee, "Nature of Power Line Medium and Design Aspects for Broadband PLC System," in *International Zurich Seminar on Broadband Communications (IZS)*, 2000.

[15] P. A. Brown, "Digital PowerLine (DPL) and Aircraft Communication Systems," tech. rep., NOR.WEB DPL Ltd, 1999.

[16] R. P. Rickard and J. E. James, "A Pragmatic Approach to Setting Limits to Radiation from Powerline Communications Systems," tech. rep., NOR.WEB DPL Ltd, 1999.

[17] D. M. J. Devasirvatham, "Multipath Time Delay Spread in the Digital Portable Radio Environment," *IEEE Communications Magazine*, vol. 25, pp. 13–21, June 1987.

[18] R. J. C. Bultitude, S. A. Mahmoud, and W. A. Sullivan, "A Comparison of Indoor Radio Propagation Characteristics at 910 MHz and 1.75 GHz," *IEEE Journal on Selected Areas in Communications*, vol. 7, pp. 20–30, January 1989.

[19] C. E. Belfiore and John H. Park, Jr., "Decision Feedback Equalization," *Proceedings of the IEEE*, vol. 67, pp. 1143–1156, August 1979.

[20] J. Tellado-Mourelo, E. K. Wesel, and J. M. Cioffi, "Adaptive DFE for GMSK in Indoor Radio Channels," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 492–501, April 1996.

[21] S. U. H. Qureshi, "Adaptive Equalization," *Proceedings of the IEEE*, vol. 73, pp. 1349–1387, September 1985.

[22] D. A. George, R. R. Bowen, and J. R. Storey, "An Adaptive Decision Feedback Equalizer," *IEEE Transactions on Communication Technology*, vol. COM-19, pp. 281–293, June 1971.

[23] S. A. Fechtel and H. Meyr, "An Investigation of Channel Equalization Techniques for Moderately Rapid Fading HF-Channels," in *International Conference on Communications (ICC)*, vol. 2, pp. 768–772, 1991.

[24] C. Tidestav, *The Multivariable Decision Feedback Equalizer – Multiuser Detection and Interference Rejection*. PhD thesis, Uppsala University, December 1999. ISBN 91-506-1371-5.

[25] S. A. Fechtel, H. Meyr, and M. Moenenclay, *Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing*. Wiley Series in Telecommunications and Signal processing, Wiley, 1998.

[26] S. A. Fechtel and H. Meyr, "Optimal Parametric Feedforward Estimation of Frequency-Selective Fading Radio Channels," *IEEE Transactions on Communications*, vol. 42, pp. 1639–1650, February/March/April 1994.

[27] N. Al-Dhahir and J. M. Cioffi, "MMSE Decision-Feedback Equalizers: Finite-Length Results," *IEEE Transactions on Information Theory*, vol. 41, pp. 961–975, July 1995.

[28] J. G. Proakis, "Chapter 26: Channel Equalization," in *The Communications Handbook* (J. D. Gibson, ed.), pp. 339–363, CRC Press, 1997.

[29] J. G. Proakis, *Digital Communications*. McGraw-Hill, 1995.

[30] G. D. Forney, Jr. and M. V. Eyuboğlu, "Combined Equalization and Coding Using Precoding," *IEEE Communications Magazine*, vol. 29, pp. 25–34, December 1991.

[31] R. Raheli, A. Polydoros, and C.-K. Tzou, "Per-Surviror Processing: A General Approach to MLSE in Uncertain Environments," *IEEE Transactions on Communications*, vol. 43, pp. 354–364, February/March/April 1995.

[32] A. A. Giordano and F. M. Hsu, *Least Square Estimation with Applications to Digital Signal Processing*. Wiley, 1985.

[33] R. M. Gray, "Toeplitz and Circulant Matices: A Review," tech. rep., Stanford University, 2000. http://www-isl.stanford.edu/~gray/toeplitz.pdf.

[34] N. Al-Dhahir and J. M. Cioffi, "Fast Computation of Channel-Estimate Based Equalizers in Packet Data Transmission," *IEEE Transactions on Signal Processing*, vol. 43, pp. 2462–2473, November 1995.

[35] J. M. Cioffi, G. P. Dudevoir, M. V. Eyuboglu, and G. D. Forney, "MMSE Decision-Feedback Equalizers and Coding – Part I: Equalization Results," *IEEE Transactions on Communications*, vol. 43, pp. 2582–2594, October 1995.

[36] J. M. Cioffi, G. P. Dudevoir, M. V. Eyuboglu, and G. D. Forney, "MMSE Decision-Feedback Equalizers and Coding – Part II: Coding Results," *IEEE Transactions on Communications*, vol. 43, pp. 2595–2604, October 1995.

[37] J. Aldis, "Equalisation method, particularly for offset modulation types." European Patent EP0998083, May 2000.

[38] B. Parhami, *Computer Arithmetic – Algorithms and Hardware Designs*. Oxford University Press, 2000.

[39] T. Kailath and A. H. Sayed, "Displacement Structure: Theory and Applications," *SIAM Review*, vol. 37, pp. 297–386, September 1995.

[40] T. Kailath and J. Chun, "Generalized Displacement Structure for Block-Toeplitz, Toeplitz-Block, and Toeplitz-Derived Matrices," *SIAM Journal on Matrix Analysis and Applied Mathematics*, vol. 15, pp. 114–128, January 1994.

[41] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Johns Hopkins Series in the Mathematical Sciences, Johns Hopkins University Press, 1996.

[42] I. Schur, "Über Potenzreihen, die im Inneren des Einheitskreises beschränkt sind," *Journal für Reine und Angewandte Mathematik*, vol. 147, pp. 205–232, 1917.

[43] O. Axelsson, *Iterative Solution Methods*. Cambridge University Press, 1994.

[44] H. T. Kung and C. E. Leiserson, *Introduction to VLSI Systems*, ch. 8.3 Algorithms for VLSI Processor Arrays, pp. 271–292. Addison-Wesley, second ed., 1980.

[45] S. J. Bellis, W. P. Marnane, and P. J. Fish, "Alternative systolic array for non-square-root Cholesky decomposition," *IEE Proceedings Computers and Digital Techniques*, vol. 144, no. 2, pp. 57–64, 1997.

[46] R. Wyrzykowski, "On the Synthesis of Systolic Architecture for the Cholesky Decomposition," *Advances in Modelling & Simulation*, vol. 21, no. 2, pp. 19–31, 1990.

[47] H. M. Ahmed, J.-M. Delosme, and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *Computer*, pp. 65–82, January 1982.

[48] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays," *SPIE Real-Time Signal Processing IV*, vol. 298, pp. 19–26, 1981.

[49] J. G. McWhirter, "Recursive least-squares minimization using a systolic array," *SPIE Real-Time Signal Processing VI*, vol. 431, pp. 105–110, 1983.

[50] S. S. Haykin, *Adaptive filter theory*. Prentice Hall, second ed., 1991.

[51] C. Mead and L. Conway, *Introduction to VLSI Systems*. Addison-Wesley, second ed., 1980.

[52] B. Haller, J. Götze, and J. R. Cavallaro, "Efficient Implementation of Rotation Operations for High Performance QRD-RLS Filtering," in *Proceedings ASAP '97*, (Zürich, Switzerland), pp. 162–174, July 14–16 1997.

[53] Y. H. Hu, "CORDIC-Based VLSI Architectures for Digital Signal Processing," *IEEE Signal Processing Magazine*, vol. 9, pp. 16–35, July 1992.

[54] J. Ma, K. K. Parhi, and E. F. Deprettere, "Pipelined Implementation of CORDIC based QRD-MVDR Adaptive Beamforming," in *1998 Fourth International Conference on Signal Processing (ICSP) Proceedings* (Y. Baozong, ed.), vol. 1, pp. 514–517, The Chinese Institute of Electronics (CIE) Signal Processing Society, IEEE Press, October 1998.

[55] B. Haller, M. Streiff, U. Fleisch, and R. Zimmermann, "Hardware implementation of a systolic antenna array signal processor based on CORDIC arithmetic," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp. 4141–4144, 1997.

[56] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, vol. EC-8, pp. 330–334, 1959.

[57] J. S. Walther, "An Unified Algorithm for Elementary Functions," in *Proceedings Spring Joint Computer Conference*, vol. 38, p. 397, AFIPS press, 1971.

[58] J. Chun, R. Roychowdhury, and T. Kailath, "Systolic Array for Solving Toeplitz Systems of Equations," *SPIE Advanced Algorithms and Architectures for Signal Processing III*, vol. 975, pp. 19–27, 1988.

[59] S.-Y. Kung and Y. H. Hu, "A Highly Concurrent Algorithm and Pipelined Architecture for Solving Toeplitz Systems," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-31, pp. 66–76, February 1983.

[60] N. Al-Dhahir and A. H. Sayed, "A Parallel Low-Complexity Coefficient Computation Processor for the MMSE-DFE," in *Thirty-First Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1586–1590, 1998.

[61] N. Al-Dhahir and A. H. Sayed, "CORDIC-Based MMSE-DFE Coefficient Computation," *Digital Signal Processing*, vol. 9, pp. 178–194, 1999.

[62] S.-F. Hsiao and J.-M. Delosme, "Householder CORDIC Algorithms," *IEEE Transactions on Computers*, vol. 44, pp. 990–1001, August 1995.

[63] Radio Equipment and Systems (RES), "HIgh PERformance Radio Local Area Network (HIPERLAN), Type 1 functional specification," Tech. Rep. ETS 300 652, European Telecommunications Standards Institute (ETSI), December 1995.

[64] AMS Austria Mikro Systeme International AG, *0.35 Micron Standard Cell 3.3V Databook – 0.35μm CMOS Digital Core Cells 3.3V*, April 2000. http://asic.amsint.com/databooks/csx33/core/.

[65] "Lexra LX4080P Core." http://www.lexra.com/lx4080p.html.

[66] "ARC Processor Cores." http://www.arccores.com.

[67] Xilinx Inc., *LogiCORE Variable Parallel Virtex Multiplier*, v1.0.2 ed., October 1999.

[68] J. L. Hennessey and D. A. Patterson, *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann, second ed., January 1996.

[69] M. S. Schlansker and B. R. Rau, "EPIC: An Architecture for Instruction-Level Parallel Processors," tech. rep., HP Laboratories Palo Alto, February 2000.

[70] M. S. Schlansker, B. R. Rau, S. Mahlke, V. Kathail, R. Johnson, S. Anik, and S. G. Abraham, "Achieving High Levels of Instruction-Level Parallelism with Reduced Hardware Complexity," tech. rep., HP Laboratories Palo Alto, November 1994.

[71] S. Roos, R. Lamberts, and H. Corporaal, "Remove: A Computer Architecture Designed for Modern VLSI Technology." 1999.

[72] R. M. Stallman *et al.*, *Using and Porting the GNU Compiler Collection*. Free Software Foundation, 2.95.2 ed., 1999.

[73] M. Berekovic, H.-J. Stolberg, M. B. Kulaczewski, P. Pirsch, H. Möller, H. Runge, J. Kneip, and B. Stabernack, "Instruction Set Extensions for MPEG-4 Video," *Journal of VLSI Signal Processing*, vol. 23, pp. 27–49, October 1999.

[74] Altera, Inc., "Nios Soft Core Embedded Processor." http://www.altera.com/document/ds/ds_excnios.pdf.

[75] Xilinx, Inc., "The Xilinx and ARC Cores Alliance for Configurable Processor Cores on Xilinx FPGAs." http://www.xilinx.com/products/logicore/alliance/arc/arcspot.htm.

[76] Texas Instruments, "TMS320C3x." http://www.ti.com/sc/docs/products/dsp/other.htm.

[77] Analog Devices, "ADSP-21xx." http://www.analog.com/industry/dsp/.

Curriculum vitae

Personal Information

Thomas Michael Sailer
Born 29. April 1971
Citizen of Winterthur, ZH, Switzerland

Education

1978–1984	Primary school in Winterthur, ZH
1984–1986	Secondary school in Winterthur, ZH
1986–1990	College in Winterthur, ZH
1991–1996	M. Sc. in Electronic Engineering at the Swiss Federal Institute of Technology (ETH) in Zürich

Work

1988–1994	Internship at Walesch Electronic AG, Effretikon, CH: working on Vibration Measurement Devices
1993–1994	Internship at Druckerei Sailer & Cie, Winterthur, CH: Computer system administration
1993–1994	Internship at Schümperlin Avionics, Effretikon, CH: Calibration of HF and VHF radios
1996–2000	Teaching and Research Assistant at the Electronics Laboratory of the ETH Zürich