Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

A Scalable Architecture for Massive MIMO Base Stations Using Distributed Processing

Examensarbete utfört i Datorteknik
vid Tekniska högskolan vid Linköpings universitet
av

Erik Bertilsson

LiTH-ISY-EX--16/5019--SE

Linköping 2016

A Scalable Architecture for Massive MIMO Base Stations Using Distributed Processing

Examensarbete utfört i Datorteknik
vid Tekniska högskolan vid Linköpings universitet
av

Erik Bertilsson

LiTH-ISY-EX--16/5019--SE

Handledare: **Oscar Gustafsson**
ISY, Linköpings universitet

Examinator: **Kent Palmkvist**
ISY, Linköpings universitet

Linköping, 20 december 2016

Avdelning, Institution Division, Department		Datum Date
Avdelningen för Datorteknik Department of Electrical Engineering SE-581 83 Linköping		2016-12-20

Språk Language ☐ Svenska/Swedish ☒ Engelska/English ☐ _____	Rapporttyp Report category ☐ Licentiatavhandling ☒ Examensarbete ☐ C-uppsats ☐ D-uppsats ☐ Övrig rapport ☐ _____	ISBN _____ ISRN LiTH-ISY-EX--16/5019--SE Serietitel och serienummer Title of series, numbering _____
URL för elektronisk version		ISSN _____

Titel Title	Skalbar arkitektur för distribuerade beräkningar i Massiv MIMO system A Scalable Architecture for Massive MIMO Base Stations Using Distributed Processing
Författare Author	Erik Bertilsson

Sammanfattning Abstract
<p>Massive MIMO is an emerging technology for future wireless systems that has received much attention from both academia and industry recently. The most prominent feature of Massive MIMO is that the base station is equipped with a large number of antennas. It is therefore important to create scalable architectures to enable simple deployment in different configurations.</p> <p>In this thesis, a distributed architecture for performing the baseband processing in a massive OFDM MU-MIMO system is proposed and analyzed. The proposed architecture is based on connecting several identical nodes in a K-ary tree. It is shown that, depending on the chosen algorithms, all or most computations can be performed in a distributed manner. Also, the computational load of each node does not depend on the number of nodes in the tree (except for some timing issues) which implies simple scalability of the system.</p> <p>It is shown that it should be enough that each node contains one or two complex multipliers and a few complex adders running at a couple of hundred MHz to support specifications similar to LTE. Additionally the nodes must communicate with each other over links with data rates in the order of some Gbps.</p> <p>Finally, a VHDL implementation of the system is proposed. The implementation is parameterized such that a system can be generated from a given specification.</p>

Nyckelord Keywords	Elektroteknik, ASIC, Massiv MIMO
------------------------------	----------------------------------

Abstract

Massive MIMO is an emerging technology for future wireless systems that has received much attention from both academia and industry recently. The most prominent feature of Massive MIMO is that the base station is equipped with a large number of antennas. It is therefore important to create scalable architectures to enable simple deployment in different configurations.

In this thesis, a distributed architecture for performing the baseband processing in a massive OFDM MU-MIMO system is proposed and analyzed. The proposed architecture is based on connecting several identical nodes in a K -ary tree. It is shown that, depending on the chosen algorithms, all or most computations can be performed in a distributed manner. Also, the computational load of each node does not depend on the number of nodes in the tree (except for some timing issues) which implies simple scalability of the system.

It is shown that it should be enough that each node contains one or two complex multipliers and a few complex adders running at a couple of hundred MHz to support specifications similar to LTE. Additionally the nodes must communicate with each other over links with data rates in the order of some Gbps.

Finally, a VHDL implementation of the system is proposed. The implementation is parameterized such that a system can be generated from a given specification.

Acknowledgements

I would like to thank my supervisor Oscar Gustafsson and my examiner Kent Palmkvist for their help and involvement in this thesis project. Additionally, I would like to thank Erik G Larsson for providing key insights in the subject of Massive MIMO.

Finally, I would like to extend my gratitude to my family and friends for their support during my studies.

Erik Bertilsson

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Goals	4
1.3	Research Questions	5
1.4	Delimitations	5
1.4.1	Hardware Delimitations	5
1.4.2	Algorithmic Delimitations	6
2	Theory	7
2.1	Communication Link Duplex Modes	7
2.1.1	Time Division Duplex	7
2.1.2	Frequency Division Duplex	8
2.2	Massive MIMO	8
2.2.1	Channel Estimation	8
2.2.2	Downlink Precoding	9
2.2.3	Uplink Decoding	11
2.3	DSP algorithms	12
2.3.1	Signal-flow Graphs	12
2.3.2	Precedence Graph	13
2.3.3	Resource Allocation	13
2.4	Fast Fourier Transform	14
2.5	OFDM	15
2.5.1	Modulation and Demodulation	15
2.5.2	Cyclic Prefix	16
2.5.3	Zero Padding	16
2.6	Graphs	17
2.6.1	K -ary Trees	17
2.7	Hermitian Matrices	17
3	Method	19
3.1	Pre Study	19
3.2	Algorithm Analysis	20
3.3	Implementation	20

3.3.1	Node	20
3.3.2	Tree	21
4	System Architecture & Specifications	23
4.1	System Architecture	23
4.1.1	Node Interconnection Topology	24
4.1.2	Selected Interconnection Topology	25
4.1.3	Data Distribution	26
4.2	Node Architecture	26
4.3	Frame Structure	27
4.4	Specifications	28
5	Computational Analysis	29
5.1	Channel Estimation	29
5.2	Conjugate Beamforming Processing	30
5.2.1	MRT Precoding	30
5.2.2	MRC Decoding	30
5.2.3	Data Dependencies and Computational Complexity	31
5.2.4	Inter Chip Communication	33
5.3	ZF precoding and ZF decoding	34
5.3.1	Linear Decoding and Precoding Matrices	34
5.3.2	ZF Precoding	35
5.3.3	ZF Decoding	36
5.3.4	Data Dependencies and Computational Complexity	36
5.3.5	Inter Chip Communication	39
6	VHDL Implementation	41
6.1	Processing Elements	41
6.1.1	Complex Adder	42
6.1.2	Complex Multiplier	42
6.2	Node Implementation	43
6.2.1	Processing Elements Module	45
6.2.2	Channel Estimation Controller	47
6.2.3	Uplink Controller	47
6.2.4	Downlink Controller	48
6.2.5	Inter Chip Communication	50
6.3	Tree implementation	55
7	Discussion	57
7.1	Algorithms	57
7.1.1	Computational Complexity and Data Locality	57
7.1.2	Interconnection Requirements	58
7.2	Interconnection Topology	59
8	Conclusions	61
	Bibliography	63

List of Abbreviations

ASIC	Application Specific Integrated Circuit
BS	Base Station
CB	Conjugate Beamforming
CCU	Central Control Unit
CEC	Channel Estimation Controller
DC	Downlink Controller
DFT	Discrete Fourier Transform
DPC	Dirty Paper Coding
DPM	Downwards Propagation Module
DSP	Digital Signal Processing
FDD	Frequency Division Duplex
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
ICC	Inter Chip Communication
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
I/O	Input/Output
LTE	Long Term Evolution
MIMO	Multiple Input Multiple Output
MRC	Maximum Ratio Combining
MRT	Maximum Ratio Transmission
MU-MIMO	Multiple User MIMO
OFDM	Orthogonal Frequency Division Multiplexing
PE	Processing Element
QAM	Quadrature Amplitude Modulation
RF	Radio Frequency
SDMA	Spatial Division Multiple Access
SFG	Signal Flow Graph
SINR	Signal to Interference and Noise Ratio
SNR	Signal to Noise Ratio
TDD	Time Division Duplex
UC	Uplink Controller
UPM	Upwards Propagation Module
VLSI	Very Large Scale Integration
ZF	Zero Forcing

1

Introduction

This chapter introduces the thesis project. Section 1.1 contains a brief introduction to the problem at hand, and the motivation for solving it. The goals of the thesis project are described in Section 1.2. The research questions are described in Section 1.3. Finally, the delimitations of the project are explained in Section 1.4.

1.1 Motivation

In recent years, mobile data traffic usage has dramatically increased and is forecasted to continue to do so [1, 2]. The increased mobile data traffic requires that new technologies are developed to meet the demands of the users.

Massive Multi User (MU) Multiple Input Multiple Output (MIMO) is a technique where a cellular Base Station (BS) is equipped with a large number of antennas and is connected to several simultaneous terminals [3]. Massive MIMO significantly improves the spectral efficiency and reduces the radiated energy [4]. The spectral efficiency is improved by utilizing Space Division Multiplexing. Thus multiple terminals can operate in the same time and frequency slot simultaneously. Another benefit of the Massive MIMO technology is the use of many low cost and low power components. Due to the large number of antennas at the BS, the hardware impairments are diminished with an increasing number of antennas at the base station [5].

One of the challenges in massive MIMO is the large computational complexity.

In order to meet the computational requirements and reach low power consumption, Application Specific Integrated Circuits (ASIC) are favorable over implementations on more general platforms such as FPGAs [6]. In current base stations, the digital and analog signal processing makes up roughly 5–15% of the total power consumption [7].

1.2 Goals

The goal of this thesis project is to investigate a distributed architecture for baseband processing in Massive MIMO systems. The basic system architecture consists of a Central Control Unit (CCU) and a scalable part. The scalable part consists of a number of nodes connected in a tree structure, where each node is implemented by identical hardware components. The tree structure considered is a complete K -ary tree. An overview of the system architecture can be seen in Figure 1.1.

Figure 1.1: An example of a distributed system architecture for Massive MIMO. In this example the scalable part is comprised of a binary tree.

Each node in the tree performs baseband processing and implements the RF front ends for one or multiple antennas. This architecture has several benefits over a centralized implementation. Since all nodes are identical it is possible to add or remove nodes to change the number of antennas in the system. If one node in the tree is broken, it is cheap to replace it rather than to replace a large component in a centralized implementation. Additionally, since the analog to digital conversion and part of the computations reside physically close to the antenna, the analog signal path can be made short.

1.3 Research Questions

Do the limitations of the tree architecture affect the choice of precoding and decoding algorithms?

While the tree architecture has many benefits, there are also drawbacks. For algorithms where computations must be performed with data from all nodes, data must be collected in the CCU. The results of these computations may also have to be propagated back to the nodes.

With the proposed architecture, is there any bound on the number of antennas incorporated in the design of the system?

To meet the timing constraints for the calculations, is there a limit to the number of nodes in the tree structure? Since data from each node must be propagated to the CCU. This transmission of data causes some latency that must be accounted for when scheduling the computations.

What is the data rate and latency requirements for the chip interconnections?

To transmit data to and from the nodes, each node must incorporate a number of off-chip transceivers. These transceivers must be able to provide sufficient levels of latency and data rates to meet the timing constraints.

1.4 Delimitations

This thesis will focus on the system architecture and selected parts of the baseband processing of a Massive MU-MIMO communication system:

- Channel Estimation
- Linear Decoding for the uplink data transmission
- Linear Precoding for the downlink data transmission

The design of the baseband processing in each node as well as a high level interconnection scheme is included here. The node implementation will be simulated as a HDL model.

1.4.1 Hardware Delimitations

There are several parts of the hardware design that is not performed in the thesis project:

- *VLSI Design.* The nodes are described with a high level RTL model for simulation. No synthesis will be done.
- *Chip Synchronization:* The nodes in the tree are assumed to be synchronized with each other. No synchronization mechanism between chips will be included in the thesis.
- *Radio Frequency (RF) Chain:* No parts of the analog circuitry for the RF chain will be considered.

1.4.2 Algorithmic Delimitations

The system considered uses OFDM, where each node contains an OFDM subsystem. The OFDM subsystem is considered when analyzing the computational complexity in each node as well as the data dependency and timing. It is however not implemented in this thesis project.

Additionally, synchronization between terminals and base station will not be considered.

2

Theory

This chapter provides the theoretical background required to understand the subject of this thesis. Different duplex modes of communication link are described in section 2.1. Section 2.2 introduces massive MIMO concepts. In section 2.3 fundamental properties of DSP algorithms and their implementation is described.

Section 2.4–2.7 describes related topics, such as the fast fourier transform, OFDM modulation, graphs and hermitian matrices.

2.1 Communication Link Duplex Modes

In two way communication links, the uplink and downlink communication cannot share the same time and frequency resource simultaneously. There are two fundamental ways to share these resources.

2.1.1 Time Division Duplex

Time division duplex (TDD) is when the uplink and downlink communication share carrier frequency, but is multiplexed in time. Only one side of the communication system can transmit data at the time. One of the advantages of TDD is that the uplink and downlink radio channels are reciprocal to each other [8], meaning that their impulse response is the same when sending signals in both directions.

2.1.2 Frequency Division Duplex

In frequency division duplex (FDD) the uplink and downlink communication does not operate on the same carrier frequency. This means that both uplink and downlink data transmission can occur simultaneously. Since the uplink and downlink communication does not share carrier frequency, the fading characteristics of the two radio channels will differ [8].

2.2 Massive MIMO

Massive MIMO is a technology that utilizes Spatial Division Multiple Access (SDMA) to achieve higher spectral efficiency and energy efficiency than contemporary communication systems [9]. This is achieved by employing a large number of synchronized antennas at the BS, that are communicating with several terminals simultaneously. In this thesis, the MIMO system is considered to have M antennas and serves K terminals, see Figure 2.1.

Figure 2.1: A Massive MIMO system

The system under consideration also incorporates OFDM.

2.2.1 Channel Estimation

To transfer data between the BS and the terminals in the system, the channel impulse responses between the antennas in the BS and the terminals must be estimated. There are several different techniques for estimating these impulse responses. Assuming that the channels are frequency flat, the channel matrix $\mathbf{H} \in \mathbb{C}^{M \times K}$, can be written as

$$\mathbf{H} = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,K} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ h_{M,1} & h_{M,2} & \cdots & h_{M,K} \end{pmatrix}, \quad (2.1)$$

where $h_{i,j} \in \mathbb{C}$ is the channel frequency response coefficient between antenna i and terminal j . Let

$$\mathbf{x}_p^k = \begin{bmatrix} 0_{(k-1) \times 1} & p & 0_{(K-k) \times 1} \end{bmatrix} \quad (2.2)$$

be the pilot vector transmitted by terminal k . The scalar p is chosen to be $p = \sqrt{KE_s}$, where E_s is the average symbol energy. The BS then receives the signal matrix

$$\mathbf{Y}_p = \mathbf{H}\mathbf{X}_p + \mathbf{N}_p \in \mathbb{C}^{M \times K}, \quad (2.3)$$

where $\mathbf{X}_p \in \mathbb{C}^{K \times K}$ is the pilot matrix and $\mathbf{N}_p \in \mathbb{C}^{M \times K}$ is a noise matrix. The pilot matrix $\mathbf{X}_p$ can be written as

$$\mathbf{X}_p = \begin{bmatrix} (x_p^1)^T & (x_p^2)^T & \cdots & (x_p^K)^T \end{bmatrix} \quad (2.4)$$

A channel estimate is obtained according to equation 2.5 [10].

$$\hat{\mathbf{H}} = \frac{\mathbf{Y}_p}{p}. \quad (2.5)$$

2.2.2 Downlink Precoding

Precoding is a technique that utilizes the channel state information at the BS to adjust the transmission of symbols to the current channel conditions. In the downlink data transmission the BS precodes the K symbols intended for the terminals into M streams that are transmitted by the M antennas, according to Figure 2.2. The optimal precoder is based on dirty paper coding (DPC), but due to its high complexity it is not well suited for practical implementations [10]. There are many types of precoders, both linear precoders [11] as well as non-linear precoders. Linear precoders have been shown to be optimal under certain conditions. Linear precoding techniques are often preferred due to their lower computational requirements.

For linear precoders, the transmitted signal vector $\mathbf{x} \in \mathbb{C}^{M \times 1}$ can be written as

$$\mathbf{x} = \mathbf{W}\mathbf{q}, \quad (2.6)$$

where $\mathbf{W} \in \mathbb{C}^{M \times K}$ is the linear precoding matrix and $\mathbf{q} = [q_1 \ q_2 \ \cdots \ q_K]^T$ is the symbol vector to be transmitted.

Figure 2.2: The K symbols are precoded into M streams in the precoding step.

Maximum Ratio Transmission

The maximum ratio transmission (MRT) precoding is designed to maximize the signal to noise ratio (SNR) of the received stream for all terminals [12]. It does not however take inter-terminal interference into account. For maximum ratio transmission (MRT), the precoding matrix is determined by

$$\mathbf{W} = \mathbf{H}^*. \quad (2.7)$$

The MRT precoder performs well under low SNR conditions and has a low computational complexity. However, its performance deteriorates under high signal to interference and noise ratio (SINR) scenarios.

Zero Forcing

The zero forcing (ZF) precoder is designed to eliminate the inter-terminal interference. As opposed to the MRT precoder, ZF does not take noise into account. For ZF precoding, the precoding matrix is determined by

$$\mathbf{W} = \mathbf{H}^* \left(\mathbf{H}^T \mathbf{H}^* \right)^{-1}. \quad (2.8)$$

Generally, ZF precoding achieves better spectral efficiency than the MRT precoder [4]. The computational complexity for ZF is higher, since the pseudo inverse of the channel matrix must be computed.

2.2.3 Uplink Decoding

Decoding in the uplink data transmission is the process of separating the received signal vector $\mathbf{y} \in \mathbb{C}^{M \times 1}$ into K streams of symbols $\tilde{\mathbf{y}} \in \mathbb{C}^{K \times 1}$. This is illustrated in Figure 2.3.

Figure 2.3: Linear decoding at the base station

Similarly to the precoding, there are both linear and non-linear decoders. Due to their lower computational complexity, the linear precoders are often preferred.

The linear decoding is done in two steps. First the received signal vector is multiplied with a linear detection matrix $\mathbf{A} \in \mathbb{C}^{K \times M}$. The entries of the resulting matrix is then quantized to the closest symbol in the modulation alphabet.

$$\tilde{\mathbf{x}} = f(\mathbf{A}\mathbf{y}) \quad (2.9)$$

Maximum Ratio Combining

The Maximum Ratio Combining (MRC) decoder is very similar to the MRT precoder. MRC decoding minimizes the noise in the received symbol vector, but ignores the effects of inter-terminal interference. The linear detection matrix for MRC is

$$\mathbf{A} = \mathbf{H}^H \quad (2.10)$$

Just as in the MRT precoder, the computational complexity for MRC is low.

Zero Forcing

The Zero Forcing decoder is again very similar to the ZF precoder. The inter-terminal interference is eliminated but the noise effects are ignored. The decoding matrix for the ZF decoder is

$$\mathbf{A} = \left(\mathbf{H}^H \mathbf{H} \right)^{-1} \mathbf{H}^H \quad (2.11)$$

2.3 DSP algorithms

In this section, some of the most important tools for analyzing computational properties of DSP algorithms are introduced.

2.3.1 Signal-flow Graphs

DSP algorithms can be described using differential equations, such as the FIR filter in (2.12).

$$y[n] = x[n]h[0] + x[n-1]h[1] + x[n-2]h[2] + x[n-3]h[3] \quad (2.12)$$

The same DSP algorithm can also be described with a Signal Flow Graph (SFG). The SFG of the FIR filter in (2.12) can be seen in Figure 2.4. This is a variation of the SFG and is called a block-diagram.

Figure 2.4: Signal flow graph for a FIR filter.

The symbols in the SFG are chosen to mirror its behavior. In the SFG above the box containing Z^{-1} means a delay element of one sample period. The triangle represents a multiplication and the circle with a plus sign represents addition. SFGs are used to more describe the algorithm more clearly than a system of differential equations.

2.3.2 Precedence Graph

The SFG is a graphical representation of a DSP algorithm. However, to analyse its computational properties, more information is needed. The SFG can be redrawn into an SFG in precedence form, when the order between operations are specified.

The first step is to add names to all nodes and operations in the graph. In Figure 2.5 the operations are name A-G and the nodes are named v_1 - v_3 and u_1 - u_6 .

Figure 2.5: SFG with additional information

The input and the nodes v_1 - v_3 are available at the start of the computation and are called initial nodes. The values of the nodes u_1 - u_6 and the output must be computed. Figure 2.6 shows the SFG in precedence form for the FIR filter in Figure 2.5.

Figure 2.6: SFG in precedence form

Each set of nodes can be computed in parallel, while each set can be computed sequentially. The grey arrows represents updating the initial nodes when the current sample has been calculated.

2.3.3 Resource Allocation

The minimum number of processing elements of a specific type required are shown in equation 2.13.

$$N_{PE} = \left\lceil \frac{N_{\text{operations}} T_{\text{exec}}}{T_{\text{computation}}} \right\rceil, \quad (2.13)$$

where $N_{\text{operations}}$ are the number of operations to be performed, T_{exec} is the execution time for the processing element and $T_{\text{computation}}$ is the time available to finish the computations. Note that this is a lower bound on the number of processing elements and that the number may increase due to the scheduling of operations [13].

2.4 Fast Fourier Transform

The Fast Fourier Transform (FFT) is an algorithm for computing the Discrete Fourier Transform (DFT). Computing an N -point DFT directly is done by

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad (2.14)$$

where $k \in \{0, 1, \dots, N-1\}$ and $W_N = e^{-\frac{j2\pi}{N}}$. The computational complexity of the DFT is $\mathcal{O}(N^2)$. The input, intermediate results and output are complex values.

If N is non prime, the DFT can be decomposed by selecting P and Q such that

$$N = PQ. \quad (2.15)$$

The equation for the DFT can then be rewritten as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} = X(Qk_1 + k_2) = \sum_{n_1=0}^{P-1} \left[\left(\sum_{n_2=0}^{Q-1} x(n_1 + Pn_2) W_Q^{n_2 k_2} \right) W_N^{n_1 k_2} \right] W_P^{n_1 k_1}, \quad (2.16)$$

where the inner sum is a Q -point DFT, followed by P complex multiplications. The outer sum is then a P -point DFT. By decomposing the DFT this way. If P and/or Q is non-prime then their respective DFT can be further decomposed. By decomposing the DFT in this way, the computational complexity is decreased to $\mathcal{O}(N \log N)$ [14].

If the DFT size is $N = 2^m$, then it can be decomposed into a number of 2-point DFTs, called butterfly operations, see Figure 2.7, and complex multiplications. The equations for the butterfly operation are

$$\begin{aligned} y(0) &= x(0) + x(1), \\ y(1) &= x(0) - x(1). \end{aligned} \quad (2.17)$$

Figure 2.7: Butterfly operation.

For an N -point DFT (where $N = 2^m$), $\frac{N}{2} \log_2 N$ butterfly operations, and $\frac{N}{2}(\log_2(N) - 1)$ multiplications are required [13]. The SFG for an 8-point FFT algorithm can be seen in Figure 2.8.

Figure 2.8: Signal flow graph for an 8-point FFT algorithm.

2.5 OFDM

Orthogonal frequency division multiplexing (OFDM) is a technique for transmitting a data stream over multiple carrier frequencies. By utilizing several carrier frequencies, the system can deal with frequency selective fading and narrowband interference more efficiently than a single carrier system. If only a small portion of the subcarriers are faulty, error correction codes can correct these errors.

2.5.1 Modulation and Demodulation

When transmitting one OFDM symbol, N_{DFT} complex symbols are modulated onto N_{DFT} subcarriers. This is done by taking the IDFT of the input data. The

transmitter output is given by

$$x_n = \sum_{k=0}^{N_{\text{DFT}}-1} X_k e^{j2\pi \frac{kn}{N}}. \quad (2.18)$$

To recover the transmitted data, the receiver demodulates the OFDM symbol by calculating the DFT of the received signal.

$$X_k = \sum_{n=0}^{N_{\text{DFT}}-1} x_n e^{-j2\pi \frac{kn}{N}}. \quad (2.19)$$

2.5.2 Cyclic Prefix

To eliminate intersymbol interference (ISI) and intercarrier interference (ICI), a cyclic prefix is added to each OFDM symbol, see Figure 2.9. The cyclic prefix needs to be longer than the multipath delay in order to preserve orthogonality between subcarriers. [15].

Figure 2.9: Cyclic prefix of an OFDM Symbol.

2.5.3 Zero Padding

To ensure that no aliasing is introduced when the samples are passed through a digital-to-analog converter, oversampling is introduced. Oversampling can efficiently be implemented by inserting zeros in the input vector to the IDFT computation. The zeros should be added to the middle of the vector to ensure that the zeros are mapped to the frequencies close to half the sample rate and minus half the sample rate respectively. [16]

The size of the DFT and IDFT is denoted N_{DFT} and the number of nonzero elements (the number of subcarriers used for data transmission) in the input vector is N_{SC}

2.6 Graphs

A graph is called a tree if two vertices are connected by exactly one path. This means that all graphs that does not contain cycles are trees. A graph is said to be rooted if there is one vertex that is distinguished as the root [17].

2.6.1 K -ary Trees

A K -ary tree is a rooted tree where all nodes have at most K child vertices. A complete K -ary tree must be filled on all levels, except the last. An example can be seen in Figure 2.10.

Figure 2.10: An example of a complete K -ary tree. In this case $K = 2$.

2.7 Hermitian Matrices

A square matrix $A \in \mathbb{C}^{n \times n}$ is Hermitian if

$$A = A^H, \quad (2.20)$$

where A^H is the Hermitian transpose (sometimes referred to as conjugate transpose) of A [18]. The property

$$a_{i,j} = a_{j,i}^* \quad (2.21)$$

must hold for all $i, j \in \{1, 2, \dots, n\}$. This property also tells us that the diagonal of the matrix must be real-valued. If a matrix $A \in \mathbb{C}^{n \times n}$ is Hermitian, then only

$$\frac{n(n+1)}{2} \quad (2.22)$$

values need to be stored (the upper or lower half, including the diagonal elements), since the rest of the matrix can be obtained by conjugating the elements at the transposed indices.

3

Method

The thesis project was divided into three parts: *pre study*, *algorithm analysis*, *implementation*.

3.1 Pre Study

A large part of the pre study was to read relevant literature and contemporary research regarding the subject. This includes:

- *Massive MIMO*. Theory regarding the concept of massive MIMO in general, to improve understanding of the subject.
- *Linear Decoding and Precoding in massive MIMO*. The processes of linear Decoding and Precoding are a central part of the massive MIMO baseband processing and the thesis project.

During the pre study, an understanding of the problem was formulated. The information was primarily sourced from research articles and in some cases books and doctoral theses.

3.2 Algorithm Analysis

During the algorithm analysis phase, the algorithms considered for the linear precoding and linear decoding were subject to analysis from the perspective of the proposed architecture. The algorithms considered were

- Maximum Ratio Transmission precoder and Maximum Ratio Combining decoder
- Zero Forcing precoder and Zero Forcing decoder

The work was mainly conducted in two steps. The first step was to mathematically describe the algorithms in such a way that they would fit the proposed system architecture. The second step was to map the calculations to the architecture and to analyze the data dependencies and timing constraints that occur.

3.3 Implementation

The goal of the implementation phase was to create a VHDL model for simulating the behavior of the tree structure of the proposed architecture. It was also desirable that the VHDL code for the nodes were synthesisable. The VHDL model for the tree structure was not required to be synthesisable. The model was made to be parameterized such that the number of terminals, the number antennas and the word length could be changed quickly. The VHDL model for each node in the tree was generated from MATLAB, while the tree structure was constructed in as a VHDL entity for easy simulation in ModelSim.

3.3.1 Node

Large parts of the VHDL code for each node is generated in MATLAB. In each time slot of communication, the computations are performed in three different phases:

- Channel estimation
- Uplink data transmission
- Downlink data transmission

The computations for each of these phases were implemented by first describing the algorithm in the MATLAB and then to automatically generate the VHDL code.

Algorithm Description

The first step was to construct an SFG for the algorithm in MATLAB. This SFG is then used to create a naive schedule. By analyzing the precedence graph, rules were constructed to automatically rearrange the schedule to reduce the number of processing elements required while still meeting the timing constraints.

The modified schedule was then used to assign operations to processing elements. The memory variables were then assigned to memories and memory cells.

VHDL Code Generation

From the resulting schedule, processing elements assignment and memory assignment, VHDL code were generated for the interconnection between processing elements, memories and I/O ports.

3.3.2 Tree

The tree structure was implemented as non-synthesizable VHDL code using generate statements to instantiate a number of nodes in a binary tree.

4

System Architecture & Specifications

In this chapter the proposed architecture is introduced. The overall system architecture is described in section 4.1. The architecture of each node is described in section 4.2. Section 4.3 describes the sequence of OFDM symbols that are sent between the terminal and the BS. The specifications that are used for analyzing and implementing the system are introduced in section 4.4.

4.1 System Architecture

With a large number of antennas comes high computational requirements. One way of dealing with these demands is to create a parallel computation platform in order to achieve the required computational throughput.

To create this platform there are two major obstacles to overcome. The first is to partition the computations in a distributed manner, such that all or most computations can be performed independently of other partitions. The second part is to connect these separate instances in some topology that is well balanced for the given application.

In this thesis we propose a distributed architecture where the computations are shared among several identical nodes. An illustration of the proposed architecture can be seen in Figure 1.1. The architecture consists of two main parts. The first part is the CCU which is responsible for the system operation. The second part is a K -ary tree that is responsible for the radio frequency (RF) chains and digital baseband processing.

Figure 4.1: Two different arrays and a tree structure.

Figure 4.2: The number of routing hops in the different interconnection topologies.

4.1.1 Node Interconnection Topology

There has been some previous work in the same general direction as this master thesis. The authors in [19, 20] has proposed a system where the processing is distributed across several identical modules. These modules are connected in an array structure. The Argos demonstrator in [21] uses a hybrid interconnection topology. First of all, multiple nodes are connected serially (unary tree). Multiple of these chains are then connected in by hub nodes in a fat tree structure.

Figure 4.1 illustrates the maximum distance between nodes in different topologies. The two topologies to the left are array structures where the centralized structure is connected to different positions in the array. The distance from the furthest nodes in these two are roughly $\sqrt{M}$ and $\frac{3}{2}\sqrt{M}$ respectively. The right-most shows a binary tree, where the centralized structure is connected to the root node. The maximum distance in the tree is roughly $\log_2 M$. Here, it is assumed that each node contains one antenna, and thus the number of nodes in the system is equal to the number of antennas at the base station, M .

Figure 4.2 shows the number of routing hops for the different topologies in Figure 4.1. Clearly, for large M , the tree structure has a significantly lower maximum number of routing hops.

Another matter related to the routing of data in the topology is the actual routing scheme used. In the given application, data is transmitted from each node to the centralized structure and back. This means that much of the routing complexity can be omitted, since data does not need to be sent between arbitrary nodes. However, there are some issues left. Routing data in the tree topology is trivial. Data is simply sent to or from the parent node. This means that no explicit routing mechanism needs to be implemented.

In an array topology, some form of routing mechanism must be implemented. This could be either statically or dynamically configured. Therefore the node complexity will be higher than that for the tree topology. While this has a cost in complexity, it also has a benefit in the form of increased fault tolerance. If a node or communication link fails during operation, data can simply be routed another path to the CCU and thus lower the impact of component failures.

Finally, physical antenna placement will also influence the choice of interconnection topology. In systems where the antennas are physically placed in an array, the array interconnection topology will have the advantage of simpler cable routing. This advantage is however lost if the antennas are physically placed in some non-uniform pattern.

4.1.2 Selected Interconnection Topology

In this thesis, the tree structure is selected as the interconnection topology. The tree structure can be created in several ways. The most important parameters are the height of the tree and the number of antennas per node. The height of the tree, h_{tree} is calculated as

$$h_{\text{tree}} = \left\lceil \log_{D_{\text{node}}-1}(N_{\text{nodes}}) \right\rceil + 1, \quad (4.1)$$

where D_{node} is the degree of the internal nodes of the tree ($D_{\text{node}} = 3$ for a binary tree). The height of the tree is an important property when deciding the time it takes to propagate data through the tree, in both upwards and downwards direction.

The number of nodes in the tree is calculated as

$$N_{\text{nodes}} = \left\lceil \frac{M}{M_{\text{node}}} \right\rceil, \quad (4.2)$$

where M is the total number of antennas in the system, and M_{node} is the number of antennas per node. From Equation 4.1 and Equation 4.2, it is clear that by modifying the number of antennas per node, the height of the tree is affected.

Figure 4.3: The basic architecture of one node in the tree.

4.1.3 Data Distribution

Data is sent between the nodes over communication links with data rate R_{up} and R_{down} , in the upwards and downwards direction of the tree.

In downlink operation, the CCU feeds the nodes with QAM modulated symbols intended for the terminals. In uplink operation, the nodes in the tree structure compute the estimated received symbols and send them to the CCU.

The minimum required data rates for the communication links are calculated in chapter 5. The latency of sending data in one direction over one of these communication links is $T_{latency,link}$.

4.2 Node Architecture

The nodes in the tree are comprised of three main parts: *intra-chip I/O*, *digital baseband processing* and *RF-chains*. An illustration of the node architecture can be seen in Figure 4.3.

The intra-chip I/O consists of transceivers for data propagation upwards and downwards in the tree. The number of transceivers in each node is D_{node} (the same as the degree of the internal nodes of the tree). The baseband processing part consists of processing elements, memories and control structures required for the computations. The third and last part are the RF-chains. Each node contains M_{node} RF-chain.

Figure 4.4: Structure for one frame of data transmission.

4.3 Frame Structure

The data transmission scheme is based on the LTE frame structure and utilizes OFDM. Data transmission is divided into radio frames, subframes and slots as depicted in Figure 4.4.

One radio frame is 10 ms long and contains 10 subframes. The first subframe is used for synchronization between terminal and base station. This process however, is not included in this thesis. The other 9 subframes contains two slots each, used to transmit data between terminal and base station.

Each slot consists of seven OFDM symbols. The first symbol is a pilot symbol, used to measure the channel between the terminal and base station. The pilot is followed by two uplink data symbols and a guard interval to switch between uplink and downlink transmission. The base station then sends two downlink data symbols to the terminal, followed by another guard interval. There is no explicit downlink channel estimation performed, but since the uplink channel was estimated previously the channel reciprocity property of the radio channel is utilized.

While using this frame structure, the channels are estimated once every 0.5 ms. This means that if the coherence time of the radio channel is larger than 0.5 ms, the channel estimates are still valid. In other scenarios where the coherence time of the channel is different, the frame structure will have to be modified. This has significant implications on the results of the analysis performed in this thesis. It is however possible to perform the same analysis with the modified specifications.

Table 4.1: System specifications.

Name	Description	Value
K	Terminals served in the same time & frequency resource	20
M_{node}	Number of antennas in each node	1
D_{node}	Degree of the internal nodes of the tree	3 (binary tree)
N_{fft}	Size of the DFT/IDFT used in the OFDM subsystem	2048
N_{SC}	Number of subcarriers utilized in the OFDM subsystem	1200
f_{sample}	Sample rate	30.72×10^6 Hz
T_{slot}	Duration of one slot in the frame structure	0.5×10^{-3} s
N_{OFDM}	Number of OFDM symbols in one slot	7
W_{comp}	Internal wordlength used for computations	12 + 12
W_{comp}	Wordlength of QAM16 modulated symbols	2 + 2

4.4 Specifications

When analyzing the algorithms and implementing the nodes, the specifications in Table 4.1 are used. In this thesis, the selected precoding and decoding algorithms will be a combination of either the MRT precoding and the MRC decoder or the ZF precoder and the ZF decoder. Throughout the thesis, the combination of MRT and MRC will be referred to as Conjugate Beamforming (CB) processing and the combination of ZF and ZF will be referred to as ZF processing. This is for the sake of brevity and simplicity. The underlying algorithms are however the same as in section 2.2.

5

Computational Analysis

In this chapter the chosen algorithms will be analyzed and mapped to the proposed architecture. The channel estimation is analyzed in section 5.1. In this thesis, two different linear processing algorithms are analyzed from the perspective of the proposed architecture. The first is Conjugate Beamforming, and is detailed in section 5.2. The second is Zero Forcing, and is detailed in section 5.3.

5.1 Channel Estimation

The channel matrix is denoted $\mathbf{H} \in \mathbb{C}^{M \times K}$, where $h_{i,j} \in \mathbb{C}$ is the channel frequency response coefficient between antenna i and terminal j . The pilot vector transmitted by terminal k is

$$\mathbf{x}_P^k = \begin{bmatrix} 0_{(k-1) \times 1} & p & 0_{(K-k) \times 1} \end{bmatrix}. \quad (5.1)$$

Each node in the tree is responsible for determining one row of the channel matrix, $\mathbf{h}_i$. Since all terminals transmit its pilot vector simultaneously, the received pilot vector in node i is

$$\mathbf{x}_{P,\text{received},i} = \sum_{k=1}^K h_{i,k} \mathbf{x}_P^k. \quad (5.2)$$

Each node then determines its row of the channel matrix by

$$\mathbf{h}_i = \frac{\mathbf{x}_{P,\text{received},i}}{p} = \mathbf{x}_{P,\text{received},i} \times \frac{1}{p}. \quad (5.3)$$

This operation requires K divisions by p or K multiplications by $\frac{1}{p}$, which is more suitable for implementation, due to the complexity of performing divisions in hardware. The local row of the channel matrix $\mathbf{h}_i$ is then stored for further calculations.

5.2 Conjugate Beamforming Processing

The Conjugate Beamforming linear precoder and decoder presents an interesting proposition due to the low computational complexity and data locality properties that makes it well suited to a distributed implementation.

5.2.1 MRT Precoding

The transmitted signal vector $\mathbf{x} \in \mathbb{C}^{M \times 1}$ is calculated by multiplying the symbol vector $\mathbf{q} \in \mathbb{C}^{K \times 1}$ with the complex conjugate of the channel estimate matrix $\mathbf{H} \in \mathbb{C}^{M \times K}$.

$$\mathbf{x} = \mathbf{H}^* \mathbf{q} = \begin{pmatrix} \sum_{j=1}^K h_{1,j}^* q_j \\ \sum_{j=1}^K h_{2,j}^* q_j \\ \vdots \\ \sum_{j=1}^K h_{M,j}^* q_j \end{pmatrix}, \quad (5.4)$$

The value transmitted from node i is

$$x_i = \sum_{j=1}^K h_{i,j} q_j, \quad (5.5)$$

which is the inner product of the symbol vector and the local channel vector. This operation requires K multiplications and $K - 1$ additions and can be computed locally directly after the channel estimation is performed.

5.2.2 MRC Decoding

The received symbol vector is computed by multiplying the Hermitian transpose of the channel estimate matrix $\mathbf{H} \in \mathbb{C}^{M \times K}$ with the received signal vector $\mathbf{y} \in \mathbb{C}^{M \times 1}$.

$$\tilde{\mathbf{y}} = \mathbf{H}^H \mathbf{y} = \begin{pmatrix} h_{1,1}^* & h_{2,1}^* & \cdots & h_{M,1}^* \\ h_{1,2}^* & h_{2,2}^* & \cdots & h_{M,2}^* \\ \vdots & \vdots & \ddots & \vdots \\ h_{1,K}^* & h_{2,K}^* & \cdots & h_{M,K}^* \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} =$$

$$\begin{pmatrix} h_{1,1}^* y_1 + h_{2,1}^* y_2 + \cdots + h_{M,1}^* y_M \\ h_{1,2}^* y_1 + h_{2,2}^* y_2 + \cdots + h_{M,2}^* y_M \\ \vdots \\ h_{1,K}^* y_1 + h_{2,K}^* y_2 + \cdots + h_{M,K}^* y_M \end{pmatrix} = \sum_{i=1}^M \tilde{\mathbf{y}}_i \quad (5.6)$$

where

$$\tilde{\mathbf{y}}_i = \begin{pmatrix} h_{i,1}^* y_i \\ h_{i,2}^* y_i \\ \vdots \\ h_{i,K}^* y_i \end{pmatrix} \quad (5.7)$$

The column vector $\tilde{\mathbf{y}}_i$ can be calculated locally at node i and added together with the other local contributions as it is propagated upwards in the tree structure. Calculating the local contribution requires K multiplications, while adding together the contributions requires $2K$ additions (assuming that the tree structure is constructed as a binary tree).

5.2.3 Data Dependencies and Computational Complexity

The computational tasks and their data dependencies in the case of CB processing can be seen in Figure 5.1

Figure 5.1: Computational tasks, data dependencies, for a node using a straightforward scheduling of tasks in the case of MRT and MRC linear processing. CE denotes Channel Estimation.

The FFT and IFFT calculations of the OFDM system is taken into account, due to the data dependencies implied. The computational complexity of individual tasks can be seen in Table 5.1.

Table 5.1: Computational complexity of tasks when using the CB processing algorithms.

Name	Description	Multiplications	Additions
FFT	FFT computation for the received OFDM symbol	$\frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}})$	$N \log_2(N_{\text{FFT}})$
IFFT	IFFT computation for the OFDM symbol to be sent	$\frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}})$	$N \log_2(N_{\text{FFT}})$
CE	Channel Estimation	K	0
$\tilde{\mathbf{y}}_i$	Calculate $\tilde{\mathbf{y}}_i$ for each subcarrier, plus adding the vectors from the child nodes	$N_{\text{SC}} K$	$2N_{\text{SC}} K$
x_i	Calculate x_i for all subcarriers	$N_{\text{SC}} K$	$N_{\text{SC}}(K - 1)$

It can be noted for the CB algorithms that there are no need for any centralized computations, which is highly favorable for a distributed system.

Computational Complexity

In the selected frame structure, there are three factors limiting the number of required computational resources. The first is the total number of operations performed during one slot. The node must be equipped with enough resources to perform all operations in the same time as one slot. This gives us the average number of operations per sample over one slot $N_{\text{op,avg}}$. The other two are the critical paths in slot. The critical paths in the selected frame structure is from estimating the channel to sending the two downlink OFDM symbols. This gives us the number of operations per sample in the critical paths, $N_{\text{op,CP1}}$ and $N_{\text{op,CP2}}$ respectively.

Here, only the multiplications are considered for the analysis due to the higher complexity of a multiplication compared to an addition. However, every multiplication performed is followed by either zero, one or two additions. Therefore the number of adder circuits is favorably selected as two times the number of multiplier circuits. With the specifications in Table 4.1, the required number of multiplications per sample is

$$N_{\text{op,avg}} = \frac{5 \frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}}) + K (4N_{\text{SC}} + 1)}{T_{\text{slot}}/T_{\text{sample}}} \approx 9.91 \quad (5.8)$$

$$N_{\text{op,CP1}} = \frac{2 \frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}}) + K (2N_{\text{SC}} + 1)}{3T_{\text{OFDM}}/T_{\text{sample}}} \approx 9.32 \quad (5.9)$$

$$N_{\text{op,CP2}} = \frac{3 \frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}}) + K(3N_{\text{SC}} + 1)}{4T_{\text{OFDM}}/T_{\text{sample}}} \approx 7.07, \quad (5.10)$$

where T_{OFDM} is the duration of one OFDM symbol (one seventh of the total slot time). To keep up with the computational demands, the number of multipliers, $N_{\text{multipliers}}$, and the circuit clock frequency, f_{clock} , must be selected as

$$N_{\text{multipliers}} \frac{f_{\text{clk}}}{f_{\text{sample}}} \geq \max(N_{\text{op,avg}}, N_{\text{op,CP1}}, N_{\text{op,CP2}}), \quad (5.11)$$

where the ratio $\frac{f_{\text{clk}}}{f_{\text{sample}}}$ is favorably selected as an integer. For the given specifications, the average number of computations over a frame will be the limiting factor. For this case, it would be enough to use one PE running at $10f_{\text{sample}} = 307.2$ MHz.

5.2.4 Inter Chip Communication

Due to the favorable precoding and decoding matrices in the CB case, the inter chip communication can be made simple. The only data that needs to be sent from the CCU to each node are the symbol vectors to be transmitted to the terminals, each vector containing K values of QAM modulated symbols. In each frame there are two downlink OFDM symbols, where N_{SC} subcarriers are used each. The data rate required when sending data downwards from the CCU to the nodes are then

$$R_{\text{down,CB}} = \frac{2KN_{\text{SC}}W_{\text{QAM}}}{T_{\text{slot}}} \approx 384 \text{ Mbps}. \quad (5.12)$$

The only data that needs to be sent from nodes to the CCU are the received signal vector estimates. In each frame there are two uplink OFDM symbols, where N_{SC} subcarriers are used each. The only difference from the downwards data propagation is the wordlength used. Since the received symbol vector estimate is computed locally and added in all nodes, the wordlength must be increased. The data rate required in the upwards data propagation is

$$R_{\text{up,CB}} = \frac{2KN_{\text{SC}}W_{\text{comp}}}{T_{\text{slot}}} \approx 2.30 \text{ Gbps}. \quad (5.13)$$

The data rates calculated here are only the minimum required data rate. There are however other trade-offs to consider. For instance, if the data is not sent at the same pace as it is consumed at the received side, buffers need to be implemented, either in the receiver or transmitter or both.

5.3 ZF precoding and ZF decoding

The combination of ZF in both precoding and decoding is computationally more challenging than the CB processing. However, its performance is also higher in interference limited scenarios [22].

5.3.1 Linear Decoding and Precoding Matrices

For the CB processing algorithms, the linear detection matrix $\mathbf{A}$ and the linear precoding matrix $\mathbf{W}$ are obtained directly from the channel estimation. For the ZF precoder and decoder, calculating $\mathbf{A}$ and $\mathbf{W}$ requires substantially more calculations. The precoding matrix is

$$\mathbf{W} = \mathbf{H}^* \left(\mathbf{H}^T \mathbf{H}^* \right)^{-1} = \mathbf{H}^* \left(\left(\mathbf{H}^H \mathbf{H} \right)^* \right)^{-1} = \mathbf{H}^* \left(\left(\mathbf{H}^H \mathbf{H} \right)^{-1} \right)^*, \quad (5.14)$$

and the decoding matrix is

$$\mathbf{A} = \left(\mathbf{H}^H \mathbf{H} \right)^{-1} \mathbf{H}^H. \quad (5.15)$$

Let

$$\mathbf{D} = \left(\mathbf{H}^H \mathbf{H} \right)^{-1}. \quad (5.16)$$

The matrices can then be rewritten as

$$\mathbf{W} = \mathbf{H}^* \mathbf{D}^* = \left(\mathbf{H} \mathbf{D} \right)^* \quad (5.17)$$

and

$$\mathbf{A} = \mathbf{D} \mathbf{H}^H. \quad (5.18)$$

Given the fact that $\mathbf{H}^H \mathbf{H}$ is Hermitian, we know that its inverse is also Hermitian. With $\mathbf{D} = \mathbf{D}^H$, the decoding matrix can be written as

$$\mathbf{A} = \mathbf{D}^H \mathbf{H}^H = \left(\mathbf{H} \mathbf{D} \right)^H = \mathbf{W}^T. \quad (5.19)$$

To calculate $\mathbf{A}$ and $\mathbf{W}$, $\mathbf{D}$ must be known. The matrix multiplication $\mathbf{H}^H \mathbf{H}$ can be performed in a distributed manner across all nodes. The inverse is then calculated in the CCU.

$$\mathbf{H}^H \mathbf{H} = \sum_{i=1}^M \mathbf{B}_i, \quad (5.20)$$

where

$$\mathbf{B}_i = \begin{pmatrix} h_{i,1}^* h_{i,1} & h_{i,1}^* h_{i,2} & \cdots & h_{i,1}^* h_{i,K} \\ h_{i,2}^* h_{i,1} & h_{i,2}^* h_{i,2} & \cdots & h_{i,2}^* h_{i,K} \\ \vdots & \vdots & \ddots & \vdots \\ h_{i,K}^* h_{i,1} & h_{i,K}^* h_{i,2} & \cdots & h_{i,K}^* h_{i,K} \end{pmatrix}. \quad (5.21)$$

$\mathbf{B}_i$ is a Hermitian matrix and can be computed locally in each node, after the local channel estimate vector has been determined. This requires $\frac{K(K+1)}{2}$ multiplications. The local results are then added together as the matrices are propagated upwards in the tree structure.

This reduces the computational complexity of the CCU and also reduces the amount of data to be sent in the tree. Instead of propagating the entire channel matrix with $M \times K$ values, due to the Hermitian property of $\mathbf{B}$, only $\frac{K(K+1)}{2}$ values needs to be sent. However, the computational load on each node is increased.

When matrix inverse has been computed in the CCU, it is propagated downwards in the tree structure to all nodes. The nodes can then calculate their local detection and precoding vectors.

5.3.2 ZF Precoding

When the matrix $\mathbf{D}$ has been calculated, the linear precoding matrix for ZF can be calculated.

$$\mathbf{W} = \mathbf{H}^* \mathbf{D}^* = \begin{pmatrix} h_{1,1}^* & h_{1,2}^* & \cdots & h_{1,K}^* \\ h_{2,1}^* & h_{2,2}^* & \cdots & h_{2,K}^* \\ \vdots & \vdots & \ddots & \vdots \\ h_{M,1}^* & h_{M,2}^* & \cdots & h_{M,K}^* \end{pmatrix} \begin{pmatrix} d_{1,1}^* & d_{1,2}^* & \cdots & d_{1,K}^* \\ d_{2,1}^* & d_{2,2}^* & \cdots & d_{2,K}^* \\ \vdots & \vdots & \ddots & \vdots \\ d_{K,1}^* & d_{K,2}^* & \cdots & d_{K,K}^* \end{pmatrix} = \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_M \end{pmatrix}, \quad (5.22)$$

where

$$\mathbf{w}_i = \left[\sum_{j=1}^K h_{i,j}^* d_{j,1}^* \quad \sum_{j=1}^K h_{i,j}^* d_{j,2}^* \quad \cdots \quad \sum_{j=1}^K h_{i,j}^* d_{j,K}^* \right] \quad (5.23)$$

is the row of the precoding matrix that is used to calculate the transmitted signal in node i . After the $\mathbf{D}$ matrix has been propagated downwards in the tree, all data is available to calculate $\mathbf{w}_i$ in node i . Calculating $\mathbf{w}_i$ requires K^2 multiplications and $K(K-1)$ additions.

When the local precoding vector has been calculated, the transmitted signal in node i can be calculated in a similar fashion as in the MRT precoder.

$$\mathbf{x}_i = \sum_{j=1}^K w_{i,j} q_j. \quad (5.24)$$

5.3.3 ZF Decoding

Similarly to the ZF precoding, the ZF decoding relies on the $\mathbf{D}$ matrix to compute the linear decoding matrix.

$$\mathbf{A} = \mathbf{D}\mathbf{H}^H = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,K} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ d_{K,1} & d_{K,2} & \cdots & d_{K,K} \end{pmatrix} \begin{pmatrix} h_{1,1}^* & h_{2,1}^* & \cdots & h_{M,1}^* \\ h_{1,2}^* & h_{2,2}^* & \cdots & h_{M,2}^* \\ \vdots & \vdots & \ddots & \vdots \\ h_{1,K}^* & h_{2,K}^* & \cdots & h_{M,K}^* \end{pmatrix} = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_M], \quad (5.25)$$

where

$$\mathbf{a}_i = \begin{bmatrix} \sum_{j=1}^K d_{1,j} h_{1,j}^* \\ \sum_{j=1}^K d_{2,j} h_{1,j}^* \\ \vdots \\ \sum_{j=1}^K d_{K,j} h_{1,j}^* \end{bmatrix} \quad (5.26)$$

is the column of the decoding matrix that is used in node i to calculate the received symbol. When the $\mathbf{D}$ matrix has been propagated downwards in the tree, all data is available to calculate $\mathbf{a}_i$ in node i . Calculating $\mathbf{a}_i$ requires K^2 multiplications. However, as shown in subsection 5.3.1, the precoding and decoding matrices are each others transpose. This means that the row $\mathbf{w}_i$ will be equal to the column $\mathbf{a}_i$. Thus, the precoding and decoding vectors are identical and needs only be computed once.

When the local decoding vector has been calculated, the received symbol contribution from node i can be computed similarly to the MRC decoder.

$$\tilde{\mathbf{y}}_i = \begin{pmatrix} a_{i,1}^* y_i \\ a_{i,2}^* y_i \\ \vdots \\ a_{i,K}^* y_i \end{pmatrix} \quad (5.27)$$

5.3.4 Data Dependencies and Computational Complexity

The computational tasks and their data dependencies in the case of ZF precoding and ZF decoding can be seen in Figure 5.2.

The difference from the CB processing case is the additional computations required to determine the precoding and decoding matrices $\mathbf{A}$ and $\mathbf{W}$. In addition to increasing the computational load when using ZF, the data locality is significantly different. This is due to the fact that the matrix inversion needs to be performed centrally and the result must be redistributed to all nodes.

The computational complexity of the tasks in Figure 5.2 can be seen in Table 5.2.

Figure 5.2: Computational tasks, data dependencies, and communication requirements for a node using a straightforward scheduling of tasks in the case of ZF processing. *Inv* denotes the matrix inversion performed in the CCU. *CE* denotes Channel Estimation.

Table 5.2: Computational Complexity in Each Node when Using the ZF Precoding and ZF Decoding Algorithms.

Name	Description	Multiplications	Additions
FFT	FFT computation for the received OFDM symbol	$\frac{N_{FFT}}{2} \log_2(N_{FFT})$	$N_{FFT} \log_2(N_{FFT})$
CE	Channel Estimation	K	0
B_i	Calculation of the B_i matrix, plus adding the matrices from the child nodes	$\frac{K(K+1)}{2}$	$K(K+1)$
W/A	Calculate the local precoding and decoding vector.	K^2	$K(K-1)$
$\tilde{y}_i$	Calculate $\tilde{y}_i$ for each subcarrier, plus adding the vectors from the child nodes	$N_{SC}K$	$2N_{SC}K$
x_i	Calculate x_i for all subcarriers	$N_{SC}K$	$N_{SC}(K-1)$

Computational Complexity

Similarly to the CB processing, in the case of ZF processing there are three factors limiting the number of computational resources. Again, these are $N_{op,avg}$, $N_{op,CP1}$ and $N_{op,CP2}$. Not only does these contain more computations than in the CB case, but the critical paths to the downlink OFDM symbols will also be limited by the time taken to perform the centralized matrix inversion.

Figure 5.3: The required number of operations depending on the time taken to perform the centralized matrix inversion.

Let T_{inv} be the time taken to propagate the $\mathbf{B}_i$ matrices to the central node, performing the matrix inversion and redistribute the inverted matrix to all nodes. Given the specifications in Table 4.1 and $T_{\text{inv}} = 50 \mu\text{s}$, the number of operations in the ZF processing case is

$$N_{\text{op,avg}} = \frac{5 \frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}}) + K \left(\frac{3K}{2} + \frac{3}{2} + 4N_{\text{SC}} \right)}{T_{\text{slot}}/T_{\text{sample}}} \approx 9.96 \quad (5.28)$$

$$N_{\text{op,CP1}} = \frac{2 \frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}}) + K \left(\frac{3K}{2} + \frac{3}{2} + N_{\text{SC}} \right)}{(3T_{\text{OFDM}} - T_{\text{inv}})/T_{\text{sample}}} \approx 9.34 \quad (5.29)$$

$$N_{\text{op,CP2}} = \frac{3 \frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}}) + K \left(\frac{3K}{2} + \frac{3}{2} + 2N_{\text{SC}} \right)}{(4T_{\text{OFDM}} - T_{\text{inv}})/T_{\text{sample}}} \approx 11.38. \quad (5.30)$$

Again, to keep up with the computational demands, the number of multipliers, $N_{\text{multipliers}}$, and the circuit clock frequency, f_{clock} , must be selected as

$$N_{\text{multipliers}} \frac{f_{\text{clk}}}{f_{\text{sample}}} \geq \max(N_{\text{op,avg}}, N_{\text{op,CP1}}, N_{\text{op,CP2}}). \quad (5.31)$$

Unlike the CB case where the average number of operations was the limiting factor, in the ZF processing the critical path to the second downlink symbol is the limiting factor. For this case, it would be enough to use one PE running at $12f_{\text{sample}} = 368.74 \text{ MHz}$.

The influence of T_{inv} can be seen in Figure 5.3. This clearly shows that the complexity of each node depend on the time taken to propagate data to the central node, performing the matrix inversion and redistributing the data to the nodes quickly. This illustrates two things. The first is the need to quickly propagate data to and from the central node. This can be done by selecting low-latency

inter-chip communication links, selecting an interconnection topology with few routing hops, or a combination of the two.

The second is the need for a low-latency matrix inversion implementation. Interestingly, the throughput of the matrix inversion does not matter. Since only one matrix is inverted every 0.5 ms (with the current specifications), only the latency of this single matrix inversion is important for the distributed computations to be performed efficiently.

5.3.5 Inter Chip Communication

Compared to the CB case, the ZF processing introduces more complex communication between the nodes and the CCU. This is because of the computations to determine the precoding and decoding matrices. There are two types of data that needs to be propagated downwards in the tree structure.

- The result of the matrix inversion $\mathbf{D}$, that is performed in the CCU.
- Symbol vectors for the terminals for two OFDM symbols.

The inverted matrix is represented using the longer wordlength W_{comp} , while the symbol vectors use the shorter wordlength W_{QAM} . With the specifications in Table 4.1, the minimum data rate required in the downwards communication is

$$R_{\text{down,ZF}} = \frac{\left(\frac{K(K+1)}{2}\right) W_{\text{comp}} + 2KN_{\text{SC}} W_{\text{QAM}}}{T_{\text{slot}}} \approx 394 \text{ Mbps} \quad (5.32)$$

There are also two types of data that needs to be propagated upwards for each frame.

- The local contribution to the matrix $\mathbf{E}$ that is inverted in the CCU, $\mathbf{B}_i$.
- The local contributions to the received signal vector, $\tilde{\mathbf{y}}_i$ for each subcarrier of the two uplink data symbols.

Both these types use the longer wordlength W_{comp} . With the specifications in Table 4.1, the minimum data rate required in the upwards communication is

$$R_{\text{up,ZF}} = \frac{\left(\frac{K(K+1)}{2} + 2KN_{\text{SC}}\right) W_{\text{comp}}}{T_{\text{slot}}} \approx 2.314 \text{ Gbps.} \quad (5.33)$$

Comparing the resulting minimum data rates for the CB and ZF case shows that the communication requirements are virtually identical for the selected specifications. However, for the ZF case the story is not quite as simple since the latency

of propagating data upwards and downwards in the tree directly affects the required computational power.

Latency

The worst case latency of propagating a value from a node in the tree to the CCU or from the CCU to the farthest away node is

$$T_{\text{latency,node}} = h_{\text{tree}} T_{\text{latency,link}}, \quad (5.34)$$

where $T_{\text{latency,link}}$ is the latency of sending a value between two nodes, and h_{tree} is the height of the tree and is calculated as

$$h_{\text{tree}} = \left\lceil \log_{D_{\text{node}}-1}(N_{\text{nodes}}) \right\rceil + 1, \quad (5.35)$$

where D_{node} is the degree of the internal nodes of the tree (the degree of the internal nodes are tree for a binary tree).

6

VHDL Implementation

The implemented model uses the CB algorithms. It is parameterized such that the specifications can be changed and a new model for that case generated. Parts of the VHDL code is written manually, while some parts of the code are generated from MATLAB.

The system was implemented using the VHDL programming language. The goal was to use only synthesisable subsets of the language to implement the signal processing parts of the nodes. The models for the processing elements, adders and multipliers, are described in section 6.1. The implementation of the nodes are described in section 6.2. The implementation of the tree structure is not meant to be synthesisable, since it is mainly a matter of physically connecting the circuits. The implementation of the tree structure is described in section 6.3.

6.1 Processing Elements

All computations inside each node operate on complex values. The arithmetic operations that are performed are additions and multiplications. The implementation of the complex adder is described in subsection 6.1.1, and the implementation of the complex multiplier is described in subsection 6.1.2. For the remainder of the thesis, the symbols in Figure 6.1 will be used for adders and multipliers. Depending on the context, these will be either real-valued or complex-valued operators.

Figure 6.1: The symbols used for the processing elements. The left symbol is the adder and the right symbol is the multiplier.

Figure 6.2: Complex adder

6.1.1 Complex Adder

The complex adder implementation is rather straight forward. A schematic for the complex adder can be seen in Figure 6.2. The real and imaginary part of the inputs A and B are added separately to form the output Y .

The latency and execution time of the complex adder is one clock cycle each.

6.1.2 Complex Multiplier

The complex multiplier implementation consists of four real multipliers and two real adders. The schematic for the complex multiplier can be seen in Figure 6.3. The complex multiplier has two different modes of operation, either normal complex multiplication, or complex multiplication where operand A is conjugated. Assume that

$$A = a + ib \quad (6.1)$$

and

$$B = c + id, \quad (6.2)$$

the complex multiplication is then

$$AB = (a + ib)(c + id) = (ac - bd) + i(ad + bc). \quad (6.3)$$

Figure 6.3: Complex multiplier

However, if the operand A is conjugated, then the multiplication is

$$A^*B = (a - ib)(c + id) = (ac + bd) + i(ad - bc). \quad (6.4)$$

The multiplications of real and imaginary parts are still the same, but the subtraction of the real parts have now become an addition, and the addition of imaginary parts is now a subtraction. The output of the complex multiplier is calculated as

$$Y = \begin{cases} (ac - bd) + i(ad + bc), & \text{when } \text{ConjugateA} = 0 \\ (ac + bd) + i(ad - bc), & \text{when } \text{ConjugateA} = 1 \end{cases}. \quad (6.5)$$

The latency of the complex multiplier is two clock cycles, and its execution time is one clock cycle. This means that we can start a new multiplication each clock cycle, but each multiplication takes two clock cycles to finish.

6.2 Node Implementation

The goal was to create a node implementation that is scalable with regard to the number of simultaneous terminals and the number of OFDM symbols and radio slot duration. To do this, computational resources were instantiated to perform all computations related to an OFDM symbol in the same time as the duration of an OFDM symbol. An overview of the node architecture can be seen in Figure 6.4.

The node architecture is comprised of two main parts, one for performing the computations and one for storing intermediate data and communicating with the other nodes. The overall scheduling of computations in each node can be seen in Figure 6.5.

Figure 6.4: An overview of the node architecture. CE denotes Channel Estimation, UL denotes Uplink and DL denotes Downlink.

Figure 6.5: Schedule used for the CB computations in the implementation of the node.

This sequence is implemented as a Finite State Machine (FSM), which can be seen in Figure 6.6. The FSM moves to a new state every time a new OFDM symbol is received. The state of the FSM is used to determine which of the three controllers, Channel Estimation Controller (CEC), Uplink Controller (UC) or the Downlink Controller (DC), are responsible for setting the inputs and outputs of the processing elements. The controllers are described in sections 6.2.2 /- 6.2.4. During the CE state the Channel Estimation Controller is active, during the DL 1 and DL 2 states the Downlink Controller is active and during the UL 1 and UL 2 states the Uplink Controller is active. During the guard states no controller is active and therefore no computations are performed.

Figure 6.6: Finite State Machine controlling the frame sequence. CE denotes Channel Estimation, UL denotes Uplink and DL denotes Downlink.

6.2.1 Processing Elements Module

The number of computational resources was instantiated depending on the number of terminals and the clock frequency. Given the duration of each radio slot, the number of OFDM symbols per slot and the sample frequency, the number of samples per OFDM symbol is

$$N_{\text{samples,OFDM}} = \frac{T_{\text{slot}}}{N_{\text{OFDM}}} f_{\text{sample}}. \quad (6.6)$$

The number of clock cycles with the clock frequency f_{clk} occupied by one OFDM symbol is then

$$N_{\text{cycles,OFDM}} = N_{\text{samples,OFDM}} \frac{f_{\text{clk}}}{f_{\text{sample}}}, \quad (6.7)$$

where the ratio $\frac{f_{\text{clk}}}{f_{\text{sample}}}$ is selected as an integer. Given that the number of subcarriers utilized in each OFDM symbol is N_{SC} , the number of clock cycles per subcarrier is

$$N_{\text{cycles,SC}} = \frac{N_{\text{cycles,OFDM}}}{N_{\text{SC}}}. \quad (6.8)$$

Thus, the implementation can be generated from a specification and is not tied to any specific numbers for the variables listed so far. However, for the sake of brevity, the specifications in Table 4.1 is used, along with the clock frequency $f_{\text{clk}} = 11 f_{\text{sample}}$. This gives us $N_{\text{cycles,SC}} = 20$, which means that the computation for each subcarrier must be completed in 20 clock cycles.

In the downlink precoding phase, the value of each subcarrier is the inner product of the symbol vector intended for the K terminals and the local precoding vector. This requires K multiplications and $K - 1$ additions. Assuming the multipliers can perform one multiplication per clock cycle, the number of multipliers required is

$$N_{\text{multipliers,precoding}} = \left\lceil \frac{K}{N_{\text{cycles,SC}}} \right\rceil = \left\lceil \frac{K}{20} \right\rceil, \quad (6.9)$$

and the number of adders required is (assuming that additions take one clock cycle)

$$N_{\text{adders, precoding}} = \left\lceil \frac{K-1}{N_{\text{cycles, SC}}} \right\rceil = \left\lceil \frac{K-1}{20} \right\rceil. \quad (6.10)$$

In the uplink decoding phase, the value of each subcarrier is determined by computing a local contribution in the node, and adding the contributions from the child nodes, before sending the results to the parent node. Each subcarrier value received is multiplier by each entry in the local decoding vector. The number of required multipliers is

$$N_{\text{multipliers, decoding}} = \left\lceil \frac{K}{N_{\text{cycles, SC}}} \right\rceil = \left\lceil \frac{K}{20} \right\rceil, \quad (6.11)$$

however, since the additions needs to access data from the child nodes, these two additions are moved to the Inter-Chip Communication module, see subsection 6.2.5.

The number of instantiated multipliers and adders in the Processing Elements Module are then

$$N_{\text{multipliers}} = \max(N_{\text{multipliers, precoding}}, N_{\text{multipliers, decoding}}) = \left\lceil \frac{K}{20} \right\rceil, \quad (6.12)$$

and

$$N_{\text{adders}} = \left\lceil \frac{K-1}{20} \right\rceil. \quad (6.13)$$

Figure 6.7: Decoding SFG

6.2.2 Channel Estimation Controller

The CEC is responsible for the computation of the local channel vector. Once per output sample of the FFT computation, the sample is multiplied by $\frac{1}{p}$. In the case of $K = 4$, the code for the CEC is seen in Listing 6.1.

Listing 6.1: VHDL code generated for the Channel Estimation Controller

```

1 mult_in.op_a(0) <= sample;
2 mult_in.op_b(0) <= pilot_reciprocal;
3
4 process (clk)
5 begin
6     if rising_edge(clk) then
7         if (current_schedule_time = 0 and phase = CHANNEL_ESTIMATION) then
8             if sample_counter = 0 then
9                 channel_vector(0) <= mult_out.res(0);
10            elsif sample_counter = 1 then
11                channel_vector(1) <= mult_out.res(0);
12            elsif sample_counter = 2 then
13                channel_vector(2) <= mult_out.res(0);
14            elsif sample_counter = 3 then
15                channel_vector(3) <= mult_out.res(0);
16            end if;
17        end if;
18    end if;
19 end process;

```

During this phase of the computations, only one multiplier is used. The CEC is the only controller that initiates writes to the channel vector registers. These registers are readable for all other controllers.

6.2.3 Uplink Controller

The UC is responsible for the computation of the local contribution to the received symbol vector for each subcarrier of the uplink data symbols. The SFG for one sample can be seen in Figure 6.7.

Figure 6.8: Decoding schedule

An example of a schedule generated for $K = 20$ can be seen in Figure 6.8. In the schedule, the yellow boxes marks the execution time of a multiplication. This means that if two yellow boxes overlap, then multiple resources (multipliers in this case) must be instantiated. The blue boxes marks the latency of an operation. For instance, the the first multiplication starts at time 0 and the result is available at time 2.

When the schedule has been created, the operations are assigned to specific multipliers. After the processing elements assignment the VHDL code for performing the multiplications are generated. The results of the multiplications are stored in the buffer in the Inter Chip Communication module.

6.2.4 Downlink Controller

The DC is responsible for the computations of the precoding process for each subcarrier in the downlink OFDM data symbols. For each subcarrier, the inner product between the current symbol vector and the local channel vector is computed. The SFG for this computation is seen in Figure 6.9.

Due to the use of sample delays after each operation in the SFG, there are no data dependencies between operations when considering a single sample interval (20 clock cycles in this case). One benefit of this is the ease of automatically generating a valid schedule, since all operations can be moved anywhere in the schedule interval. However, this comes at the cost of memory cells. As can be seen in Figure 6.4, the DC contains memories. The memories are used to store intermediate data before the final output is computed.

Figure 6.9: Signal Flow Graph for calculating the precoded value of one subcarrier.

An example of a schedule generated for $K = 20$ can be seen in Figure 6.10. The memory variables for the generated schedule can be seen in Figure 6.11.

Figure 6.10: Precoding schedule

When the schedule is created, the arithmetic operations are assigned to specific processing elements. The memory variables are automatically partitioned into memories and memory cells. When generating the memory assignment, the memories are assumed to have one read port and one write port which can be used simultaneously.

Figure 6.11: Memory variables for the precoding schedule when $K = 20$.

The number of memories and the mapping of memory variables to memories are done automatically in MATLAB.

6.2.5 Inter Chip Communication

The Inter Chip Communication (ICC) module contains three transceivers, one for the parent node and two for the child nodes. The module also contains logic for handling the upwards and downwards data propagation. An overview of the module can be seen in Figure 6.12.

The transceivers are modelled such that they can transmit one complex value in each direction every clock cycle with 40 clock cycles latency. Given the assumptions in section 4.4, the data rate of the transceivers are

$$R_{\text{up}} = W_{\text{comp}} \frac{1}{T_{\text{clk}}} = W_{\text{comp}} f_{\text{clock}} \approx 8.110 \text{ Gbps} \quad (6.14)$$

and

$$R_{\text{up}} = W_{\text{QAM}} \frac{1}{T_{\text{clk}}} = W_{\text{QAM}} f_{\text{clock}} \approx 1.352 \text{ Gbps}. \quad (6.15)$$

Upwards Propagation Module

The Upwards Propagation Module (UPM) contains $N_{\text{numMultipliers}}$ memories to buffer the results of the multiplications during the decoding phases. This amount of memories is chosen because all multipliers will generate output the must be

Figure 6.12: Interconnect up

stored simultaneously. It also contains two adders for adding the child nodes contributions with the local contributions of the received symbol vector. A schematic for the module can be seen in Figure 6.13.

The memories are written to when the multipliers have valid results during the two uplink symbol phases. The code for generating the write address, read address and the memory select signals can be seen in Listing 6.2. The same address is used for all memories.

Figure 6.13: Interconnect up

Listing 6.2: VHDL code for generating the write address, read address and the memory select signals in the UPM.

```

1  inc_write_adr <= '1' when
2      ((mult_outputs.done = '1') and
3       (phase = UPLINK1 or phase = UPLINK2))
4      else '0';
5
6  process(clk)
7  begin
8      if rising_edge(clk) then
9          if(reset = '1') then
10             mem_write_adr <= 0;
11         elsif(inc_write_adr = '1') then
12             if(mem_write_adr = memory_size - 1) then
13                 mem_write_adr <= 0;
14             else
15                 mem_write_adr <= mem_write_adr + 1;
16             end if;
17         end if;
18     end if;
19 end process;
20
21 inc_read_adr <= child_node_data_available;
22
23 process(clk)
24 begin
25     if(rising_edge(clk)) then
26         if(reset = '1') then
27             mem_select <= 0;
28         elsif(inc_read_adr = '1') then
29             if(mem_select = num_mults - 1) then
30                 mem_select <= 0;
31             else
32                 mem_select <= mem_select + 1;
33             end if;
34         end if;
35     end if;
36 end process;
37
38 process(clk)
39 begin
40     if(rising_edge(clk)) then
41         if(reset = '1') then
42             mem_read_adr <= 0;
43         elsif(inc_read_adr = '1' and mem_select = num_mults - 1) then
44             if(mem_read_adr = memory_size - 1) then
45                 mem_read_adr <= 0;
46             else
47                 mem_read_adr <= mem_read_adr + 1;
48             end if;
49         end if;
50     end if;
51 end process;

```

It is assumed that all nodes in the tree are operating synchronously. This means that the local contribution to the received symbol vector is available before the child nodes contributions, due to the latency of inter chip communication. When the values from both the left and the right node is available, the corresponding local value is read from the buffer memories. If the node is a leaf, the values are sent upwards in the tree as soon as they are available.

Downwards Propagataion Module

The Downwards Propagation Module (DPM) is responsible for receiving data from the parent node and store it in a buffer memory. The data that is sent from the parent node is also forwarded to both child nodes as soon as it is available. An overview of the DPM can be seen in Figure 6.14.

Figure 6.14: Interconnect down

The DPM contains $N_{\text{multipliers}}$ memories for buffering data from the parent node. The values are written to one memory at the time, selected by the `mem_select` signal. When all memories have been written to once, the `mem_write_adr` is incremented. The VHDL code for the `mem_select` and `mem_write_adr` signals can be seen in Listing 6.3.

When the multipliers are activated, they read one value each from the memories. When the `mult_read_enable` signal is high, all memories are read and the `mem_read_adr` signal is incremented.

Listing 6.3: VHDL code for generating the write address, read address and the memory select signals in the DPM.

```

1  process(clk)
2  begin
3      if(rising_edge(clk)) then
4          if(reset = '1') then
5              mem_select <= 0;
6          elsif(inc_write_adr = '1') then
7              if(mem_select = num_mults - 1) then
8                  mem_select <= 0;
9              else
10                 mem_select <= mem_select + 1;
11             end if;
12         end if;
13     end if;
14 end process;
15
16 process(clk)
17 begin
18     if(rising_edge(clk)) then
19         if(reset = '1') then
20             mem_write_adr <= 0;
21         elsif(inc_write_adr = '1' and mem_select = num_mults - 1) then
22             if(mem_write_adr = memory_size - 1) then
23                 mem_write_adr <= 0;
24             else
25                 mem_write_adr <= mem_write_adr + 1;
26             end if;
27         end if;
28     end if;
29 end process;
30
31 process(clk)
32 begin
33     if rising_edge(clk) then
34         if(reset = '1') then
35             mem_read_adr <= 0;
36         elsif(mult_read_enable = '1') then
37             if(mem_read_adr = memory_size - 1) then
38                 mem_read_adr <= 0;
39             else
40                 mem_read_adr <= mem_read_adr + 1;
41             end if;
42         end if;
43     end if;
44 end process;

```

6.3 Tree implementation

The tree structure for the nodes was implemented as a VHDL model where `num_nodes` nodes were instantiated and connected as a binary tree. VHDL code for generating the nodes and the links between the nodes is seen in Listing 6.4.

Listing 6.4: VHDL code for instantiating `num_nodes` nodes.

```

1  GEN_NODES:
2  for i in 0 to num_nodes - 1 generate
3      NODES : node_top
4          port map(
5              clk                => clk ,
6              reset              => reset ,
7              parent_transceiver => top_transceivers(i),
8              left_transceiver   => left_transceivers(i),
9              right_transceiver  => right_transceivers(i),
10             ul_sample          => top_ul_sample ,
11             dl_sample          => dl_samples(i),
12             dl_sample_available => dl_samples_available(i),
13             leaf_node          => get_leaf_node(i)
14         );
15 end generate GEN_NODES;
16
17 GEN_LINKS:
18 for i in 0 to (2**tree_depth) - 2 generate
19     constant left_child_index : integer := i * 2;
20     constant right_child_index : integer := i * 2 + 1;
21 begin
22     LINK_LEFT : off_chip_link
23         generic map(
24             latency          => link_latency
25         )
26         port map(
27             clk              => clk ,
28             reset            => reset ,
29             A                => left_transceivers(i),
30             B                => top_transceivers(left_child_index)
31         );
32
33     LINK_RIGHT : off_chip_link
34         generic map(
35             latency          => link_latency
36         )
37         port map(
38             clk              => clk ,
39             reset            => reset ,
40             A                => right_transceivers(i),
41             B                => top_transceivers(right_child_index)
42         );
43 end generate GEN_LINKS;

```

Note that the code for creating the links only work for perfect binary trees, i.e. the number of nodes is

$$N_{\text{nodes}} = \sum_{i=0}^{\text{depth}-1} 2^i = 2^n - 1, \text{ depth} \in \mathbb{N}. \quad (6.16)$$

7

Discussion

This chapter provides a brief discussion of the results of the thesis work. In section 7.1, the algorithms and their respective complexities are discussed. In section 7.2 the selected interconnection topology is discussed.

7.1 Algorithms

The algorithms analyzed are Conjugate Beamforming and Zero Forcing. Here, the requirements of the two are compared.

7.1.1 Computational Complexity and Data Locality

Comparing the computational complexity of the two algorithms is not a trivial task. By counting the raw number of multiplications performed in each node, the following equations were established in chapter 5

$$N_{\text{op,avg,CB}} = \frac{5 \frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}}) + K (4N_{\text{SC}} + 1)}{T_{\text{slot}}/T_{\text{sample}}} \approx 9.91 \quad (7.1)$$

and

$$N_{\text{op,avg,ZF}} = \frac{5 \frac{N_{\text{FFT}}}{2} \log_2(N_{\text{FFT}}) + K \left(\frac{3K}{2} + \frac{3}{2} + 4N_{\text{SC}} \right)}{T_{\text{slot}}/T_{\text{sample}}} \approx 9.96. \quad (7.2)$$

Clearly, the average computational load in each node is very similar. The increased computational load on average for the ZF is

$$\frac{N_{\text{op,avg,ZF}}}{N_{\text{op,avg,CB}}} \approx 1.005. \quad (7.3)$$

For the specification given in Table 4.1, the ZF processing case only increases the computational load by 0.5 %. This does not however paint the whole picture, since the ZF processing algorithms rely on a centralized matrix inversion. This means that data has to be propagated through the tree to the CCU and back to the nodes, in addition to the actual computation performed in the CCU. The more time this process takes, the more computational resources must be added to the nodes in order to meet the deadlines of the frame structure.

The CB processing algorithms however have no such latency issues, since all computations are performed distributedly. This makes CB processing inherently more suited to a distributed implementation. The complexities of the ZF processing can however be overcome in a well designed system.

7.1.2 Interconnection Requirements

A similar comparison can be made for the communication requirements as was made for the computational resources. The minimum data rates for the two algorithms was determined in chapter 5. For the CB processing case, these are

$$R_{\text{down,CB}} = \frac{2KN_{\text{SC}}W_{\text{QAM}}}{T_{\text{slot}}} \approx 384 \text{ Mbps} \quad (7.4)$$

and

$$R_{\text{up,CB}} = \frac{2KN_{\text{SC}}W_{\text{comp}}}{T_{\text{slot}}} \approx 2.30 \text{ Gbps}. \quad (7.5)$$

For the ZF processing case, the minimum rates are

$$R_{\text{down,ZF}} = \frac{\left(\frac{K(K+1)}{2}\right)W_{\text{comp}} + 2KN_{\text{SC}}W_{\text{QAM}}}{T_{\text{slot}}} \approx 394 \text{ Mbps} \quad (7.6)$$

and

$$R_{\text{up,ZF}} = \frac{\left(\frac{K(K+1)}{2} + 2KN_{\text{SC}}\right)W_{\text{comp}}}{T_{\text{slot}}} \approx 2.314 \text{ Gbps}. \quad (7.7)$$

Again, the increased requirements for the ZF processing is very small. Similarly for the computational resources though, the ZF processing is sensitive to latency when centrally performing the matrix inversion. This means that having a slow data link in the case of ZF processing will increase the required number of computational resources.

An observation that can be made is that the same data is sent from the CCU to all nodes in the tree. So far, it is assumed that the nodes forward the received data to their child nodes. This makes the worst case latency grow with the number of nodes in the tree. This could be avoided by implementing some form of broadcasting mechanism when sending data from the CCU to all nodes. This would have a significant impact on the system design, so the advantages and disadvantages of this would have to be considered carefully.

7.2 Interconnection Topology

The selected interconnection topology was a K -ary tree. Some of its advantages and disadvantages were mentioned in subsection 4.1.1. So far only binary tree have been considered, but the analysis performed can easily be extended to include other values as well.

The tree structure shows promise because of mainly two things. The first is the reduced number of routing hops compared to for instance arrays. As have been shown in this thesis, when distributing the ZF processing algorithms the implementation becomes latency sensitive. Since the CB processing is not sensitive to latency in the same way as ZF processing, the reduced number of routing hops does not bring the same advantages.

The second advantage is the trivial routing. Since data is only propagated upwards to the parent node or passed on downwards to the child nodes, there is no need for any explicit routing mechanisms. If each node contains one antenna element, the required number of computational resources is very low, in the order of one or a couple of multipliers running at around 300 to 400 Mhz. This means that each node is very simple. Including routing protocols may incur a significant overhead to this simple node design.

8

Conclusions

This chapter concludes the thesis project and the research questions from section 1.3 are revisited.

Do the limitations of the tree architecture affect the choice of precoding and decoding algorithms?

Yes. This is a consequence of performing most of the computations among several identical nodes, rather than a limitation of the tree interconnection topology. It was shown in chapter 5 that the computational resources required in the case of ZF processing is sensitive to latency when propagating data upwards and downwards in the tree, as well as centrally computing a matrix inverse.

However, in the tree interconnection topology the number of routing hops grows logarithmically, which is slower than for instance array structures. This makes the tree a good choice in this regard.

With the proposed architecture, is there any bound on the number of antennas incorporated in the design of the system?

For the CB processing algorithms, no. In the ZF processing case, a centralized matrix inversion is required before the downlink data can be precoded. If the tree grows too large, the result of the matrix inversion may arrive too late at some nodes.

What is the data rate and latency requirements for the chip interconnections?

The minimum data rate required in the nodes for the CB case are

$$R_{\text{down,CB}} = \frac{2KN_{\text{SC}}W_{\text{QAM}}}{T_{\text{slot}}} \quad (8.1)$$

and

$$R_{\text{up,CB}} = \frac{2KN_{\text{SC}}W_{\text{comp}}}{T_{\text{slot}}} \quad (8.2)$$

In the case of ZF processing the required data rates are

$$R_{\text{down,ZF}} = \frac{\left(\frac{K(K+1)}{2}\right)W_{\text{comp}} + 2KN_{\text{SC}}W_{\text{QAM}}}{T_{\text{slot}}} \quad (8.3)$$

and

$$R_{\text{up,ZF}} = \frac{\left(\frac{K(K+1)}{2} + 2KN_{\text{SC}}\right)W_{\text{comp}}}{T_{\text{slot}}} \quad (8.4)$$

For the selected specifications, the datarates are roughly 400 Mbps for the down-link data rates, and roughly 2.4 Gbps for the uplink data rates. The difference between CB and ZF is very small.

While the data rates are not significantly different for the different algorithms, the latency requirements differs between the two. The CB processing is not affected significantly by latency, while in the ZF processing case, the processing resources will depend on the inter-node communication latency.

Bibliography

- [1] Ericsson AB, “5g radio access,” 2016, [Online; accessed 7-July-2016]. [Online]. Available: <https://www.ericsson.com/res/docs/whitepapers/wp-5g.pdf> Cited on page 3.
- [2] Cisco, “Global mobile data traffic forecast update, 2015–2020,” 2016, [Online; accessed 7-July-2016]. [Online]. Available: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html> Cited on page 3.
- [3] E. Larsson, O. Edfors, F. Tufvesson, and T. Marzetta, “Massive MIMO for next generation wireless systems,” *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 186–195, 2014. Cited on page 3.
- [4] H. Q. Ngo, E. G. Larsson, and T. L. Marzetta, “Energy and spectral efficiency of very large multiuser MIMO systems,” *IEEE Trans. Commun.*, vol. 61, no. 4, pp. 1436–1449, 2013. Cited on pages 3 and 10.
- [5] E. Björnson, J. Hoydis, M. Kountouris, and M. Debbah, “Massive mimo systems with non-ideal hardware: Energy efficiency, estimation, and capacity limits,” *IEEE Transactions on Information Theory*, vol. 60, no. 11, pp. 7112–7139, 2014. Cited on page 3.
- [6] I. Kuon and J. Rose, “Measuring the gap between fpgas and asics,” *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007. Cited on page 4.
- [7] Z. Hasan, H. Boostanimehr, and V. K. Bhargava, “Green cellular networks: A survey, some research issues and challenges,” *IEEE Communications surveys & tutorials*, vol. 13, no. 4, pp. 524–540, 2011. Cited on page 4.
- [8] J. D. Parsons and P. J. D. Parsons, “The mobile radio propagation channel,” 1992. Cited on pages 7 and 8.

- [9] T. L. Marzetta, "Noncooperative cellular wireless with unlimited numbers of base station antennas," *IEEE Trans. Wireless Commun.*, vol. 9, no. 11, pp. 3590–3600, Nov. 2010. Cited on page 8.
- [10] A. Chockalingam and B. S. Rajan, *Large MIMO systems*. Cambridge University Press, 2014. Cited on page 9.
- [11] M. Joham, W. Utschick, and J. A. Nossek, "Linear transmit processing in mimo communications systems," *IEEE Transactions on signal Processing*, vol. 53, no. 8, pp. 2700–2712, 2005. Cited on page 9.
- [12] T. K. Y. Lo, "Maximum ratio transmission," in *IEEE Int. Conf. Commun.*, vol. 2. IEEE, 1999, pp. 1310–1314. Cited on page 10.
- [13] L. Wanhammar, *DSP Integrated Circuits*. Academic Press, 1999. Cited on page 14.
- [14] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965. Cited on page 14.
- [15] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. Cambridge university press, 2005. Cited on page 16.
- [16] R. van Nee and R. Prasad, *OFDM for wireless multimedia communications*. Artech House, Inc., 2000. Cited on page 16.
- [17] N. Deo, *Graph theory with applications to engineering and computer science*. Courier Dover Publications, 2016. Cited on page 17.
- [18] B. N. Datta, *Numerical linear algebra and applications*. Siam, 2010. Cited on page 17.
- [19] A. Puglielli, N. Narevsky, P. Lu, T. Courtade, G. Wright, B. Nikolic, and E. Alon, "A scalable massive MIMO array architecture based on common modules," in *Proc. IEEE Int. Conf. Commun. Workshop*. IEEE, 2015, pp. 1310–1315. Cited on page 24.
- [20] A. Puglielli, A. Townley, G. LaCaille, V. Milovanovic, P. Lu, K. Trotskovsky, A. Whitcombe, N. Narevsky, G. Wright, T. Courtade, E. Alon, B. Nikolic, and A. M. Niknejad, "Design of energy-and cost-efficient massive MIMO arrays," *Proc. IEEE*, vol. 104, no. 3, pp. 586–606, Mar. 2016. Cited on page 24.
- [21] C. Shepard, H. Yu, N. Anand, E. Li, T. Marzetta, R. Yang, and L. Zhong, "Argos: practical many-antenna base stations," in *Proc. Int. Conf. Mobile Comput. Networking*, ACM. Association for Computing Machinery (ACM), 2012, pp. 53–64. Cited on page 24.
- [22] H. Yang and T. L. Marzetta, "Performance of conjugate and zero-forcing beamforming in large-scale antenna systems," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 2, pp. 172–179, Feb. 2013. Cited on page 34.

Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>