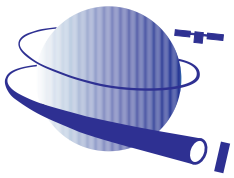

TELECOMMUNICATION
ENGINEERING

UNIVERSITY
of
TWENTE

University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science
Chair for Telecommunication Engineering

Modelling, Simulation and Implementation of an Optical Beam Forming Network Control Software System

by

Jack van Galen

Master thesis
Executed from July 2008 - Mei 2009

Supervisor: Dr. Ir. C.G.H. Roeloffzen
Advisors: ir. L. Zhuang
M. Burla, MSc
Dr. ir. P. T. de Boer

Summary

Beam shaping and beam steering, together called beam forming, is needed when processing the radio frequency signals received by a Phased Array Antenna (PAA). When correcting the arrivaltime differences between all the inputs of the PAA by adding small delays, and subsequently combining them, a strong signal can be obtained. At the Chair for Telecommunication Engineering (TE) at the University of Twente, research is done on achieving these delays fully in the optical domain using an Optical Beam Forming Network (OBFN). With an OBFN, very large bandwidths can be delayed continuously, and is thus suitable for high bandwidth applications like live television reception. The OBFN is based on thermo-optical tuning of Optical Ring Resonators (ORRs), where each ORR is capable of delaying a small fraction of the bandwidth of the signal continuously. The exact frequency range and the amount of delay are controlled by applying a voltage to small heater elements on top of the ORRs according to calculated ring settings. By combining more ORRs into a delay element, larger bandwidths can be delayed. Because of the large amount of heater elements, and the influence that one heater element has on another, a sophisticated control system is needed that is capable of automatically calculating the correct settings, and tuning all the heater elements given only the direction of arrival of the incoming satellite signal.

To achieve the goal of creating the automatic control system, two simulators were written in LabVIEW to see if underlying calculations would work in theory. The first of the two simulators was specifically designed to simulate the delay response of delay elements with a variable amount of rings. The settings for the rings were obtained by using an approximation algorithm with pre-calculated values. Several effects and their compensations have been incorporated. The end result is a scalable simulator capable of simulating delay elements containing a variable amount of rings.

The second simulator was an additional layer around the code of the first simulator, thereby creating a tool that can simulate an entire OBFN. The distribution of the delays across the rings and the calculation of the voltages is all done within this simulator. The connection to a previously designed amplifier board made it possible to apply these calculated voltages to the actual lab setup.

Finally, as a proof of concept, the simulator has been tested in the lab environment to see if the approach taken could work. The first measurements using the voltages

calculated by the control system look very promising. Also, the system is capable (with very small adjustments) of tuning future chip designs or using other tuning methods than thermo-optical.

Contents

List of Figures

List of Tables

Abbreviations

AE	Antenna Element
AOA	Angle of Arrival
API	Application Programmers Interface
CLI	Command Line Interface
DAC	Digital to Analog Convertor
DC	Directional Coupler
DE	delay element
DFD	Data Flow Diagram
DFS	Depth First Search
DLL	Dynamic link library
DUT	Device Under Test
DVB-s	Digital Video Broadcasting via Satellite
EDFA	Erbium Doped Fiber Amplifier
FPU	Floating Point Unit
FSR	Free Spectral Range
GUI	Graphical User Interface
IP	Internet Protocol
MMSE	Minimum Mean Sqared Error
MVC	Model-View-Controller
MZI	Mach-Zehnder Interferometer

NLP Non-Linear Programming

OBFN Optical Beam Forming Network

OOP Object Oriented Programming

OSBF Optical Side Band Filter

ORR Optical Ring Resonator

PAA Phased Array Antenna

PCB Printed Circuit Board

RF Radio Frequency

RISC Reduced Instruction Set Computer

RS232 Recommended Standard 232

RSS Ring Section Simulator

RTT Round Trip Time

SMART SMart Antenna systems for Radio Transceivers

SNR Signal to Noise Ratio

SPI Serial Peripheral Interface

TEC Temperature Controller

TTD True Time Delay

UI User Interface

UML Unified Modeling Language

USB Universal Serial Bus

Chapter 1

Introduction

Phased Array Antennas have been around for several decades now, and have some unique properties and advantages over conventional dish-like antennas we see every day for, for example, receiving satellite television signals. One of the most impressive PAAs is the Cobra Dane radar, located in Sheyma, Alaska and shown in figure 1.1. Built in 1976, the radar uses a ninety-five foot phased array antenna, and provides 120-degree coverage of a two thousand mile corridor that spans the eastern Russian peninsula and the northern Pacific Ocean.

PAAs are usually flat or slightly curved, and consist of a number of Antenna Elements (AEs) that act like individual micro antennas. Normally, a PAA is electronically steered using some form of controller. The ability of an antenna to steer and focus a beam to a specific target is a huge advantage over other kinds of antennas. When using electronical steering, there are no moving parts, and thus wear and tear is vastly reduced. To use the PAAs effectively, the time differences due to arrival delays of the signal between the different antenna elements should be corrected by some clever control system. After that, the signal can be combined, resulting in a signal with a high Signal to Noise Ratio (SNR). This signal can then be used for any suitable application.

This thesis consists of two parts. The first presents a design and implementation of a control system simulator that is capable of compensating the arrival time differences of all the AEs. The second part describes the implementation of a functional prototype which is then subjected to a set of measurements.

1.1 Motivation

At this moment, a working prototype of an antenna system consisting of a PAA for signal reception and an OBFN for combining the signals is being developed. The OBFN is controlled by a few dozen parameters, which are all set by hand, one by one. This is not only error prone, but also too time consuming. A better, more rigid and less time consuming solution is thus needed. A generic piece of control software will not only

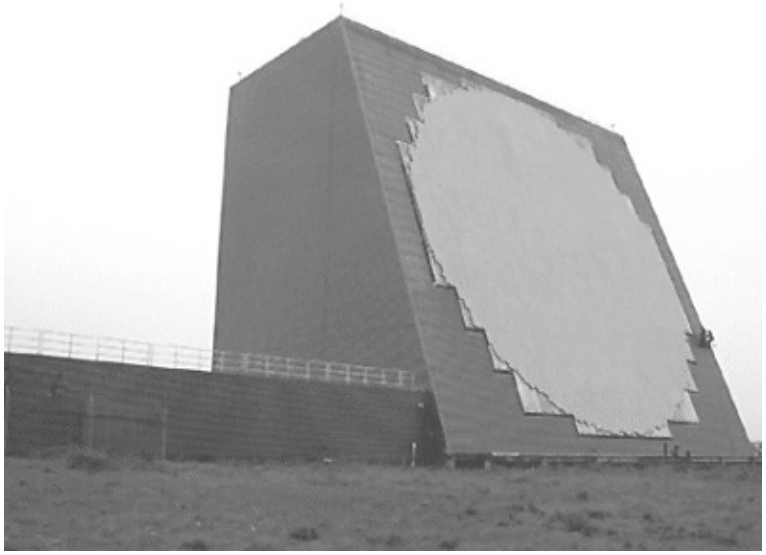

Figure 1.1: Example of a phased array antenna: the Cobra Dane Radar in Sheyma Alaska, built in 1976

help current researchers work with the OBFNs, but will also greatly reduce the time and effort future researchers will have to spent on yet to be created OBFNs based on ORRs.

Besides the direct results of this work, future use of these types of integrated systems, consisting of smart antennas and intelligent software, could provide new ways of communicating between moving objects. As a result, the project as a whole could bring an interesting new technology and new exiting applications one step closer to consumers and companies.

1.2 Background

To get a better understanding of the complete system of which the controller software will be part of, an overview is given in this section.

1.2.1 SMART project

The SMart Antenna systems for Radio Transceivers (SMART) project is aimed at providing live television services on airplanes through DVB-s by developing a novel antenna for airborne reception of satellite signals using a broadband conformal phased array antenna. The SMART project is a collaboration of different companies and research institutes. At the University of Twente, the development of a broadband integrated optical beamformer based on ORRs in CMOS-compatible waveguide technology has been done. The next step is controlling this optical chip in a manageable way. The

SMART system has a long list of requirements. The early prototype implements a subset of these requirements, shown in Table 1.1. The main advantages of the SMART concept are:

- Low loss and large instantaneous bandwidth;
- Continuous tunability (high resolution);
- Relatively compact and light-weight realization;
- Inherent immunity to EMI;
- Potential for integration with optical distribution network

1.2.2 System overview

A full system overview of the SMART system is shown in Figure 1.2. When used at the receiving end, the AEs collect radio waves coming from a satellite. These signals are converted from the electrical domain to the optical domain (E/O block) by intensity modulation, and afterwards fed into the OBFN. The OBFN is used to apply appropriate delays on each optical input, and combining them. After combining the signals, a strong optical signal is acquired, which can then be converted back to the electrical domain (O/E block). Finally, a receiver can process the signal, for example a Digital Video Broadcasting via Satellite (DVB-s) set-top box.

The OBFN shown in the system overview is managed by a control system. Ideally, the control system would automatically track a specific satellite, and use the elevation and azimuth information to calculate the correct tunings of the ORRs. The ORRs are tuned in such a way that there is constructive interference of the Radio Frequency (RF) signals coming from the desired direction. During the course of this research, a system will be developed that uses a simplified scenario consisting of linear array, and therefore dealing with only 1 variable angle.

Description	Value
Frequency range	10.7 - 12.75 GHz ($K_u band$)
Scan angle	-60 to +60 degrees
Selectivity	$<$ 2 degrees (continuous tuning)
No. elements	8
Element spacing	1.5cm or 40ps
Maximum delay	2ns

Table 1.1: Subset of the original requirements for the SMART project

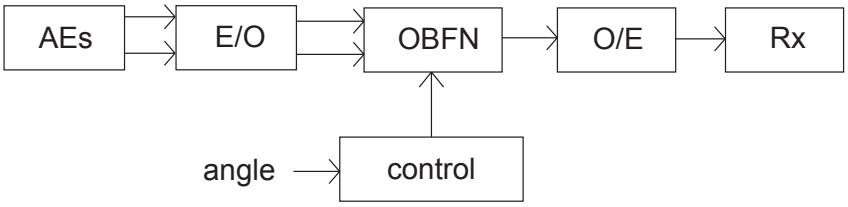

Figure 1.2: A high level overview of the system from input to processed output

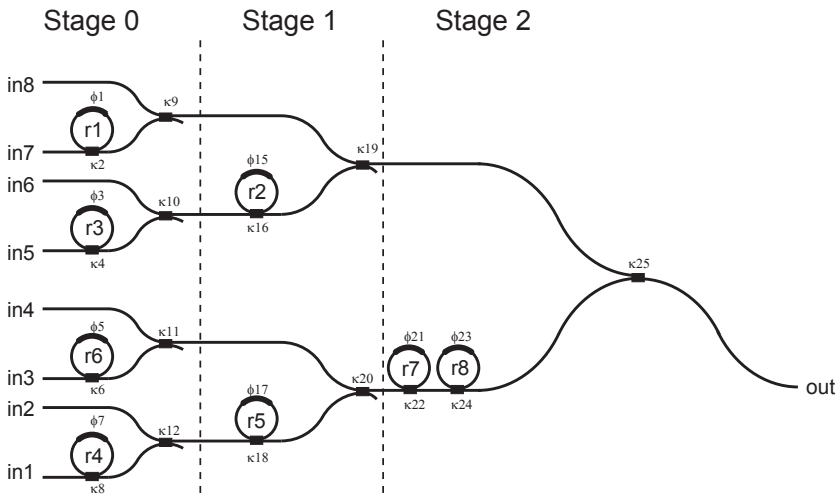

Figure 1.3: A 8×1 binary tree OBFN for a transmitter phased array antenna with 8 inputs, 1 output and 8 optical ring resonators.

1.2.3 Optical Beam Forming Networks (OBFN)

The optical chip in this system is manufactured using planar optical waveguide technology by LioniX B.V. [1]. It consists of the following building blocks: waveguides, Mach-Zehnder Interferometers (MZIs), couplers and ORRs. ORRs are chosen because they provide True Time Delay (TTD), so beam squinting will not occur. Beam squint usually occurs when working with phase shifters instead of TTDs - the position of the beam changes with frequency. The building blocks are combined to form an OBFN. A 8×1 OBFN for a receiving phased array is shown in Figure 1.3.

The OBFN is designed using a binary tree topology. Using this layout, only a small amount of ORRs have to be used to achieve a large range of delays per path, while the dimensions of the chip can be kept to a minimum. Although the freedom of tuning for every path is more restricted than in for example a parallel topology, the tuning complexity is reduced. The OBFN shown has 2^n rings in one of the branches of each stage, where n is the stage number (see Figure 1.3). Using this approach, every

path has a unique linearly increasing number of rings. Although this method seems attractive and uses a lot less rings than a parallel topology, the number of rings grows exponentially. Fortunately, the achievable delay with a ring section of a certain size is not linearly dependent on the number of rings, so far less rings have to be used. The physical layout of the chip actually produced is shown in Figure D.1 in Appendix D, and shows only 8 rings.

An ORR consists of a straight waveguide with a circular waveguide coupled to it. Using a ORR with a circumference L of 1.5cm and a waveguide group index n_g of 1.55, we can calculate the Round Trip Time (RTT) to be $T = L \cdot n_g/3 \cdot 10^8$. An ORR has a periodic group delay response, representing the effective time delay to the modulated RF signal, and a Free Spectral Range (FSR) of $1/T = 13GHz$. The group delay for a single lossless ORR as a function of frequency f is expressed by [2]:

$$\tau(f) = \frac{\kappa T}{2 - \kappa - 2\sqrt{1-\kappa}\cos(2\pi f T + \phi_{ring})} \tag{1.1}$$

Of course, no chip could be fabricated that behaves like the mathematical equation above. Although declining due to new production techniques, we have to take optical loss into account. When we consider the optical loss, the formula becomes:

$$\begin{aligned}\tau(f) = \frac{T}{2} \cdot \frac{1 - r^2(1-\kappa)}{1 + r^2(1-\kappa) - 2r\sqrt{1-\kappa}\cos(2\pi f T + \phi_{ring})} \\ + \frac{T}{2} \cdot \frac{r^2 - (1-\kappa)}{1 - \kappa + r^2 - 2r\sqrt{1-\kappa}\cos(2\pi f T + \phi_{ring})}\end{aligned} \tag{1.2}$$

The equations shown depend on the RTT T, the power coupling coefficient κ and additional round-trip phase shift of the ring ϕ. The equation involving the loss uses $r = 10^{(-l/20)}$ with l the loss of the ring in dB. When the loss is 0 dB, the second equation is of course equal to the first. Using heater elements, it is effectively possible to control the phase shift ϕ and the power coupling coefficient κ. Both parameters can be used to change the shape of one of the dotted curves shown in Figure 1.4. When changing κ, the height of the curve will be altered. When changing ϕ, the position on the x-axis (frequency) will be changed. The total area under each dotted line is constant, so there is a trade-off between peak delay and bandwidth. As a solution to the demand of higher delays for fixed bandwidths, ORRs can be cascaded, resulting in a curve that is simply the sum of the individual responses. The result of this is shown in Figure 1.4 as a solid line. The so called ripple is the effect that is clearly visible at the top of the concatenated response, and is a slight variation of group delay in a certain bandwidth. In general, the smaller the ripple, the better. Roughly speaking,

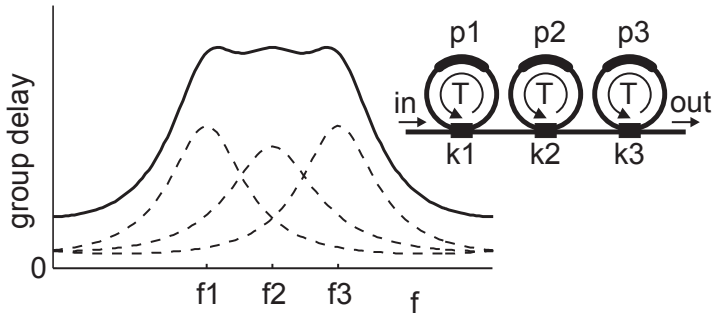

Figure 1.4: Theoretical group delay response of three cascaded ORRs.

the required number of rings is proportional to the product of the required bandwidth and the maximum delay.

The optical chip is tuned thermo-optically by electrical heating chromium resistors. As a consequence of the heat at specific places, the optical waveguide's refractive index changes. Because of this change, either the resonance frequency or the power coupling coefficient of an ORR is altered. Thermo-optical tuning itself is very well explained in Section 3.4 of [3].

1.3 Research organization

To have a well defined research project, several goals are determined. The methodology used to conduct the research, and the research questions that have to be answered when this assignment is finished are stated in the following sections.

1.3.1 Research goal

The research project described in this project has one main goal:

- *The creation of a maintainable and scalable software control system that can automatically tune all the parameters of an OBFN given only the direction of the incoming beam.*

Of course, to reach this goal, it has to be broken down into smaller, more comprehendable pieces. First, the problem will be modelled and simulated. Next, the model can be used to implement a working system using already available hardware. Finally, to verify the correctness of the system, we have to check if the simulation results correspond to real life measurement data.

1.3.2 Methodology

A methodology is used to identify distinctive actions taken in the process of this assignment. [5]. The following steps are taken in a more or less sequential order.

- *Literature study*: Related papers, theses and books must be studied to get acquainted with the subject.
- *Defining research question*: In this Thesis, a research goal is set which must be achieved.
- *Requirements analysis*: In order to present a proper design and implementation of a control system, the requirements have to be made clear.
- *Architecture design*: The design of the controlling system is presented based on which a prototype can be built.
- *Prototype implementation*: A prototype implementation is developed to test and verify the design, and to provide input for further research.
- *Performing measurements*: To test the prototype and the underlying model for correctness, the output of the system should be tested against expected results.
- *Results and conclusions*: The results are evaluated. Research questions will be answered and conclusions will be formulated.
- *Suggestions for further research*: Indications in which further research could be directed are pointed out.

1.4 Thesis organization

This chapter will start with a brief introduction to OBFNs, and provides some background information. Also a short motivation why this research project is of great interest is given.

This chapter provided an introduction to the project, its technology, related work and background information, a motivation, and finally the organization of this research. The rest of this thesis is presented as follows. In Chapter 2 the design, implementation and preliminary test results of a ring section simulator are given. In Chapter 3, the development of a complete OBFN simulator is described. Chapter 4 consists of implementation details concerning the porting of parts of the software to the hardware environment. In Chapter 5, measurement setup and results are presented. Finally, in Chapter 6, conclusions and suggestions for further research are given.

Chapter 2

Design and Implementation of the Delay Element Simulator

The system design as described in the introduction of this thesis can be broken down into smaller pieces. The ring itself must be simulated. A group of rings, which we will call a delay element (DE), has specific properties, and needs careful attention while modeling. Finally, the OBFN, a structured combination of several DEs and combiners following a specific schematic must arise. This chapter will deal with the first two steps that will result in a working DE-simulator. The third step will be dealt with in the next chapter.

2.1 Requirements

As every integrated hardware-software system, the one we are building has several general non-functional requirements. These are:

1. *Maintainability* Although the simulator can be seen as a stand alone application, it would be nice if new features could be added in the near future by others. For that reason, a programming environment should be chosen of which knowledge is widely available.

2. *Scalability* The simulator must be designed to cope with a wide variety of configurations, now and future versions. This means having the possibility of changing the number of inputs, changing the physical layout of the chip, and changing the different ORR parameters like loss and length.

3. *Resource usage* The simulator must work fluently even on an every day computer. A proper design of the simulator makes the most out of the available CPU-cycles, thereby maximizing the speed and responsiveness of the system.

4. *Ease of use* The simulator's Graphical User Interface (GUI) must be readily usable for anyone who has some knowledge about OBFNs. Clutter and unimportant input elements must either be hidden, or not be created at all. Also the ability to save and restore settings would be of great value. Not only does this improve the operating speed, but also prevents mistakes to be made in each initial simulation setup.

5. *Generic* Although the simulator is tailored to the available 8x1 OBFN chip, the simulator itself must be capable of simulating future chip designs with different tuning etc. For instance the new liquid crystal based version which will become available in the near future.

6. *Allowance for reusability* Since the Ring Section Simulator (RSS) will be part of a bigger software system later on, the software needs to be reusable.

7. *Operating system independence* Because of the wide variety of operating systems commonly used nowadays, it would be nice to make use of programming environments that are available on different platforms.

2.2 Delay element simulator design

With the requirements in mind, we can begin to design the simulator. The simulator will be built in National Instruments LabVIEW. LabVIEW is a graphical programming environment that enables the rapid development of test, measurement, and control applications. LabVIEW is also capable to comprise with all of the aforementioned requirements, and therefore the tool of choice. This section describes the steps taken to design the simulator, starting with the initial design, the flow of data and the software structure.

2.2.1 Dataflow

Dataflow is a software architecture based on the idea that changing the value of a variable should automatically force recalculation of the values of other variables. The graphical programming language LabVIEW is widely based on the idea of dataflow, and is thus very well suited for this approach. In our case, a change of the required delay forces the parameters of each ring to be automatically recalculated. The alteration is achieved by a somewhat lengthy process, and is therefore broken down into several smaller steps. These steps are displayed in a DFD as shown in Figure 2.1. Only the most important activity is displayed, being a change in the required delay by the user. In the DFD, some of the blocks deal with the physical capabilities and limitations of the optical chip. We will discuss them one by one.

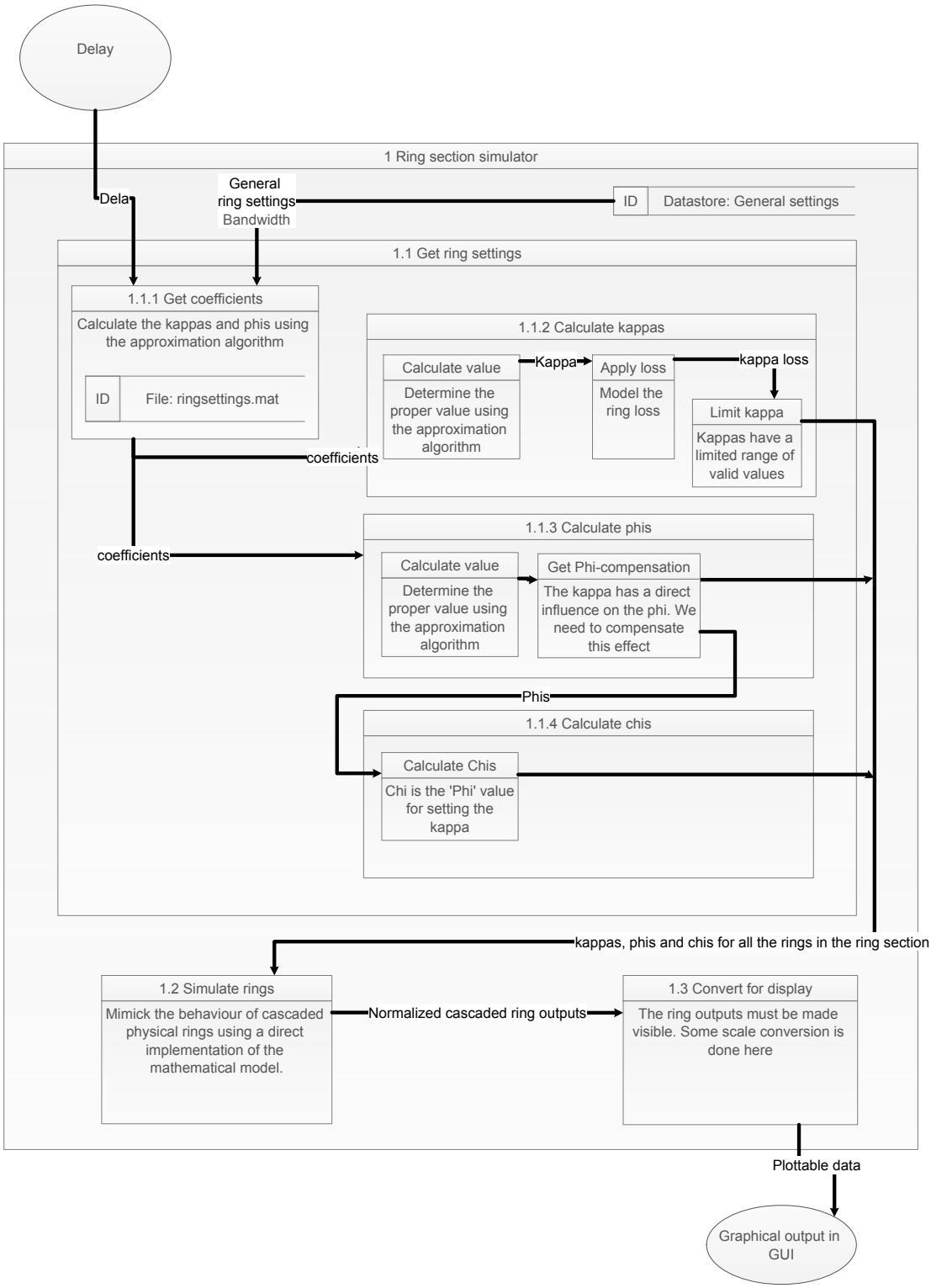

Figure 2.1: DFD for the delay element simulator

Loss

This method as shown in the DFD returns the power coupling coefficient κ when loss is applied. The resulting value equals the κ to be set. When loss would not be taken into account in the simulator, the resulting output response differs too much from the real measurement data.

The loss could be modelled and solved for in the Non-Linear Programming (NLP) solver which we will describe later. However, every change in the production process of the chip that would change the loss properties of the optical chip would mean that the entire precalculation process has to be restarted. Therefore it is desirable that the loss can be corrected for afterwards, so the precalculated coefficients, which are calculated for a lossless ring, remain the same.

Since the surface area below the response of a single ORR is constant, the time delay response curve is almost solely determined by the highest point, which has a delay of τ_{max}. Note that this is only the case for low losses, as we will see later on. The current optical chip batches have been produced with a loss between 0.1 and 0.3 dB per cm, and a ring length of 1.5 cm. The loss lies somewhere 0.15 and 0.45 dB per round trip. These losses are low enough to use the following method of compensation for the loss afterwards. Using this τ_{max}, the corresponding κ can be recoverd for the lossless case. In case there is loss, the τ_{max} must remain the same, and a new corresponding κ has to be calculated.

The maximum normalized group delay follows from Equation 3.16 in [2]:

$$\tau_{\max} = \frac{r \cdot c}{1 - r \cdot c} + \frac{r}{r - c} \tag{2.1}$$

where $c = \sqrt{1-\kappa}$ and $r = 10^{-\alpha/20}$ with α the loss in dB. In the lossless case, r would be 1, and the formula simplifies to:

$$\tau_{max} = \frac{1 + c}{1 - c} \quad \text{(lossless)} \tag{2.2}$$

When we fill in c in Equation 2.2, we acquire the maximum delay for the lossless case. To calculate the c for the case there is loss, we need to rewrite Equation 2.1 as a function of the loss factor r and the maximum delay τ_{max}:

$$c_{loss} = \frac{\tau_{\max} + \tau_{\max} r^2 \pm \sqrt{\tau_{\max}^2 - 2\tau_{\max}^2 r^2 + \tau_{\max}^2 r^4 + 4r^2}}{2(r + \tau_{\max} r)} \tag{2.3}$$

Equation 2.3 has been plotted for several losses in dB, see Figure 2.2. With this equation, the c can be determined for a given loss and given maximum delay. Now r follows directly from filling in the loss factor, and τ_{max} from the lossless case, which is

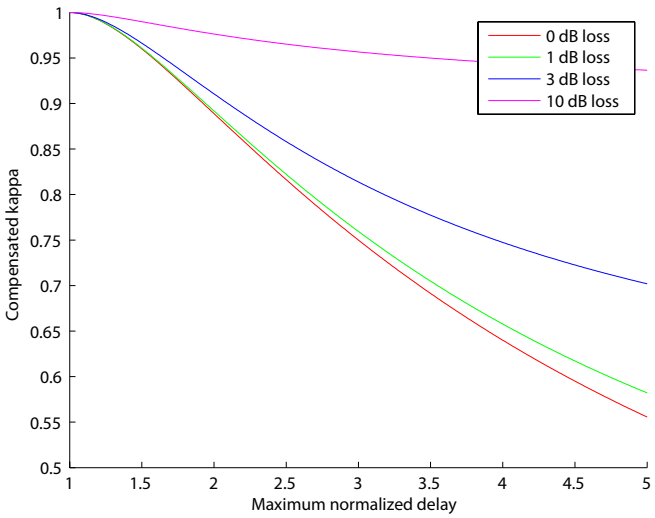

Figure 2.2: Compensated κs as function of loss per round trip and maximum delay

derived from Equation 2.2. The new kappa that now compensates for the ring loss is simply determined by:

$$\kappa_{compensated} = 1 - c_{loss}^2 \tag{2.4}$$

The method described is only usable when losses are not too high (lower than 1 dB). For higher losses, the compensated curve does not match the lossless curve anymore, as we can see in Figure 2.3.

Kappa limitation

An ORR consists of a straight waveguide and a circular waveguide next to it (See Figure 2.4). The coupling section is in fact a Directional Coupler (DC). The relation between the fields at the inputs and outputs is given by:

$$\begin{bmatrix} E_4 \\ E_2 \end{bmatrix} = \begin{bmatrix} \sqrt{1-\kappa} & -j\sqrt{\kappa} \\ -j\sqrt{\kappa} & \sqrt{1-\kappa} \end{bmatrix} \begin{bmatrix} E_3 \\ E_4 \end{bmatrix} \tag{2.5}$$

The value of the power coupling coefficient to the ring κ, which controls the height of the delay response, is limited due to the fabrication process of the optical chip. The best value for a single κ_{dc} of one of the rings on the optical chip currently achieved is 0.465 according to [6], where the ideal value would be 0.5. Since the production process of ORR on the chip is very reproducible, in our model, all values for κ_{dc} for the directional couplers are assumed to be identical. Using a heater on the upper line of

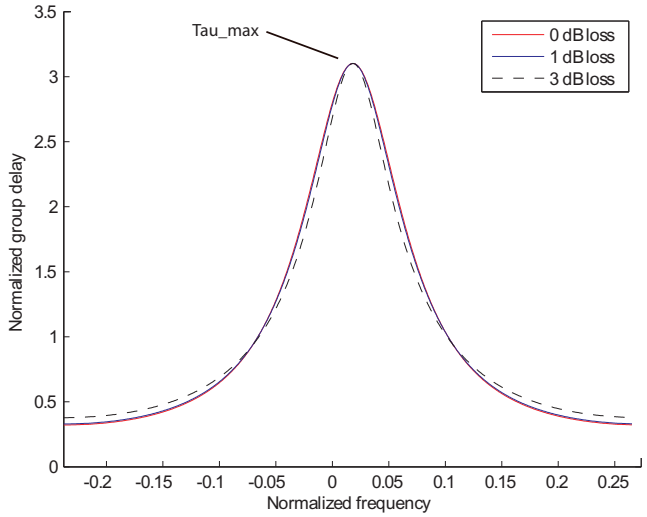

Figure 2.3: Different loss-compensated responses

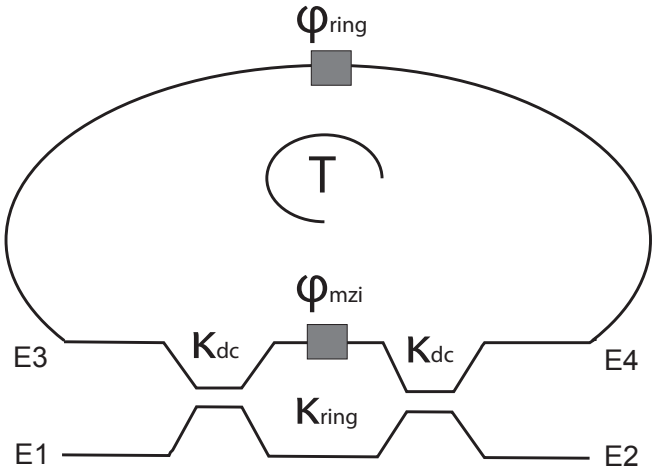

Figure 2.4: Schematic of the ORR with the Mach-Zender interferometer.

the MZI, an additional phase shift ϕ_{mzi} is added to that branch, which will effectively work as a tunable power coupler. The κ now becomes:

$$\kappa = 4\kappa_{dc}(1 - \kappa_{dc}) \cos^2(\phi_{mzi}/2) \tag{2.6}$$

Using equation 2.6 and the fact that the maximum value κ_{max} for κ is reached when the phase shift ϕ equals 0 degrees, we can calculate the maximum value for κ to be 0.9951, with a minimum value of 0 for $\phi_{mzi} = \pi/2$.

Phi compensation

Because a change of κ has a direct influence on the resonance frequency, a compensation is needed to correct this effect. Equation 2.7 is used for this compensation. κ_{max} is the value previously calculated, and is set to previously determined maximum value for κ.

$$\phi_{ring} = \frac{\phi_{mzi}}{2} \tag{2.7}$$

where ϕ_{mzi} can be found by rewriting Equation 2.6.

Kappa conversion

The relation between kappa and the actual heater response to a certain voltage can be see as a raised cosine function (see Figure 2.5). To properly operate the OBFN controller, the κs are converted according to equation 2.6. The converted values can then be used for applying voltages in a similar fashion as the ϕs. The converted values will be denoted as $\phi_{coupler}$ from now on. When no subscript is used, $\phi = \phi_{ring}$.

2.2.2 Structure

The delay element simulator is built according to the event-based programming model. Event-based programming, or event-driven programming, is a programming paradigm in which the flow of the program is determined by events i.e., sensor outputs or user actions (mouse clicks, key presses) or messages from other programs or threads. When there are no such events, the program simply waits without using any resources. A decrease from 100% CPU usage when using user-event catch loops to less than 10% using the event-based paradigm proves the usefullness of this approach.

In the simulator, there are a few CPU intensive operations:

- Changing the delay by turning the delay button as seen in Figure 2.6.
- Updating the screen with new information and drawing the graphs.

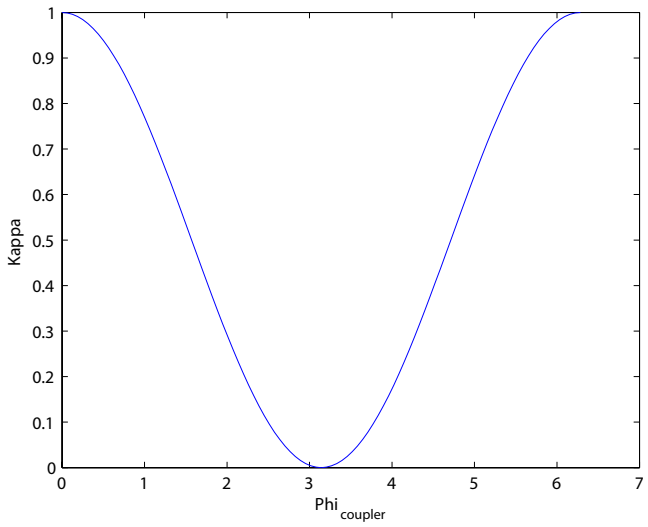

Figure 2.5: Relation between κ and $\phi_{coupler}$

For these two operations, separate events were built. Now, when the user inputs the desired delay, the software will calculate the result, and only then updates the screen. The event name of this event in the current LabVIEW model is called *User Event*.

2.3 delay element Simulator Implementation

Using the design described previously in this chapter, we can start implementing the delay element simulator in LabVIEW. For additional functionality and specific small algorithms we will revert to MatLab. LabVIEW and Matlab work seamlessly together, and is thus a good combination of a visual programming environment, combined with a solid text-based programming language. This section describes implementation details about the GUI, the approximation algorithm, and the dataflow.

2.3.1 Graphical user interface

The first thing one sees when working with a simulator is the graphical user interface (GUI). Although GUI's are an interesting topic of research by themselves, we have aimed at developing an easy to use interface just by using some common sense. This means building an interface with a minimum of clutter, a logical work flow, and one that required no unnecessary scrolling in order to maximize a clear overview. A screenshot of the interface can be seen in Figure 2.6.

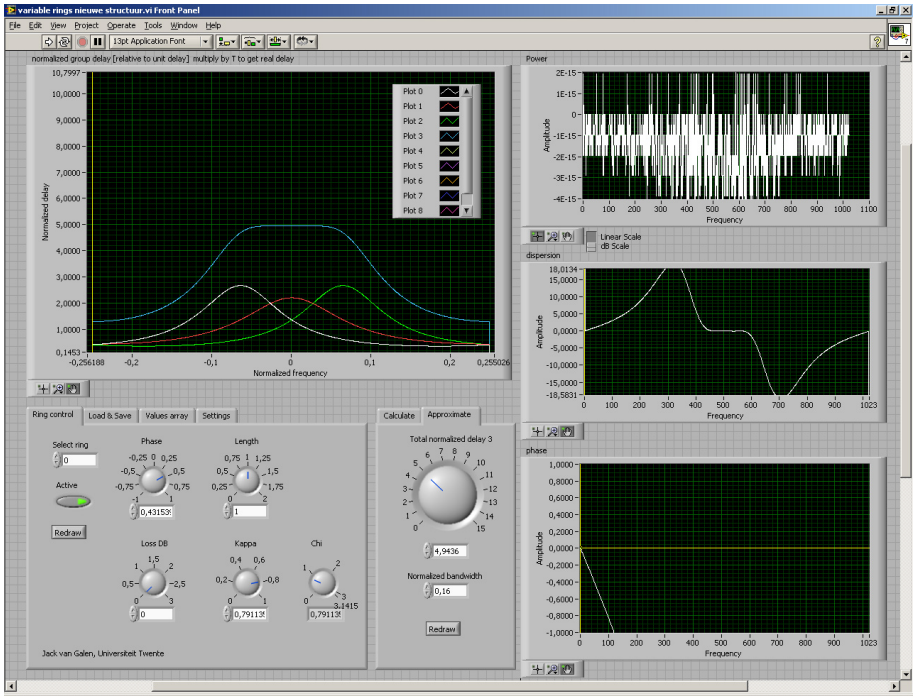

Figure 2.6: The OBFN Simulator at startup

2.3.2 An approximation algorithm

An analytical solution to acquire the ring parameter values κ and ϕ according to the mathematical model presented earlier is not possible, since the number of unknowns in the model of the delay element is greater than the number of equations. An estimation is necessary, but requires a lot of brute force calculations to be done. To overcome the burden of calculating optimal ring parameters on the fly, an approximation algorithm is used in the form of a NLP solver, that precalculates proper estimations [6]. This section will first describe the general theory of an NLP solver, followed by the implementation of such a solver in this specific case.

NLP solver

A NLP is a problem that can be stated as follows: there is one scalar-valued function f, of several variables (x here is a vector), that we want to minimize subject to one or more other functions that serve to limit or define the values of these variables. f is called the *objective function* or *cost function*, while the other functions are called the *constraints*. Of course, the minimization function could be replaced by a maximization function. Formally, we have:

$$\min_{x \in X} f(x)$$

where

$$f : R^n \rightarrow R$$
$$X \subseteq R^n$$

Basically, several solutions of each parameter within a range of possibly suitable values are tried. The algorithm then calculates the costs (or error) using the objective function, and depending of the result of the error in comparison with previous results, the solver tries a different possible solution. This process repeats itself until a large portion of the parameter values within range have been evaluated within the boundaries. Depending on the complexity of the problem to solve, this could take a while. When done, the NLP solver returns the parameter values for which the evaluation of the cost function was minimal.

One of the greatest challenges in NLP is that some problems have *local optima*, that is, solutions that satisfy the requirements of the constraint functions. Algorithms that propose to overcome this difficulty are called *Global optimization*. Global optimization would prologue the necessary time to precompute solutions, and is therefore not applied. Good initial values should be guessed in order to prevent halting in a sub-optimal state.

MMSE

A Minimum Mean Sqared Error (MMSE) estimator describes the approach which minimizes the mean square error. An example an error function based on this technique is shown in equation 2.8. Note that this function is not used to precalculate the ring settings, but serves merely as an example. In the equation, the τ_{total} represents the combined responses of several ORRs, and the target delay D is subtracted of it. Next, the result is squared, and integrated for all the frequencies between f_{min} and f_{max}, which results in the total error μ. In Figure 2.7, the top part of the combined output response of a DE with 3 ORRs is shown. The ripple is clearly visible. The total error is the square of the sum of the areas of all the shaded areas for the bandwith of interest. In this case, the bandwidth is limited to $B = f_{max} - f_{min}$.

$$\mu = \int_{f_{\min}}^{f_{\max}} (\tau_{total}(f) - D)^2 df \tag{2.8}$$

2.3.3 Normalization

Both the bandwidth and delays are normalized throughout the system, and throughout this thesis. To convert between the normalized values and the fysical values, Equation 2.9 for delays, and Equation 2.10 for bandwidths can be used.

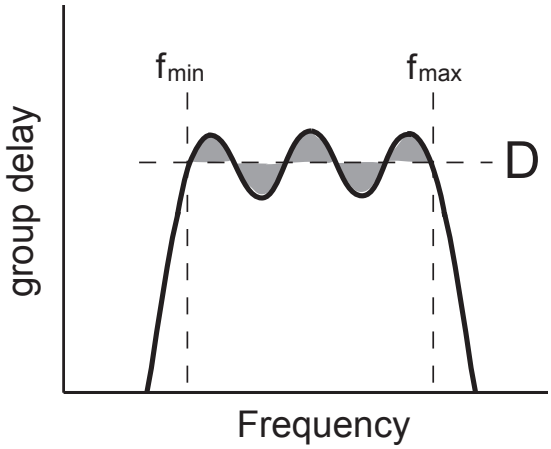

Figure 2.7: Group delay curve for 3 cascaded ORRs. The shaded area denote the parts that are added to the total costs

$$\tau_{norm} = \frac{\tau}{T} \tag{2.9}$$

$$B_{norm} = B \cdot T \tag{2.10}$$

T can be calculated when the FSR is known. The ORRs on the OBFN have a FSR of 14GHz. For the reaminder of this thesis, all mentionings of bandwidths or delays are normalized, unless stated otherwise.

Symmetry

To speed up the process of finding optimal parameters for a given delay and DE configuration, the number of unknowns can be decreased by using symmetry. As can be seen in the output window of Figure 2.6, very nice combined output responses can be achieved by a symmetrical distribution of the individual responses of the rings in a DE. When using three rings, or any other odd number of rings, the ϕ of ring 1 can be set to 0 since it is always in the center, and thus does not have to be calculated. The κ however *does* have to be optimized. The ϕs of rings 2 and 3 are identical, but opposite, so only one of them has to be optimized. Also, the κs of ring 2 and 3 are identical, again meaning that only one of them has to be optimized. The 6 parameters have now thus been reduced to only 3. Of course, for other DE-configurations, the same reasoning applies.

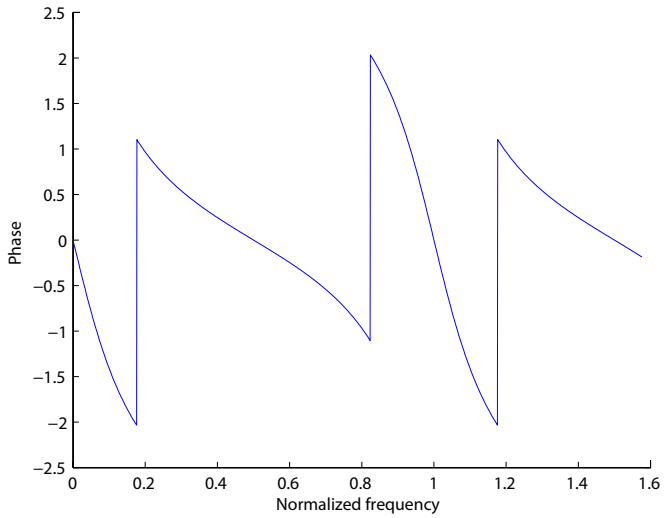

Figure 2.8: Phase response plot with normalized frequency for $\kappa = 0.8$ and $\phi = 0$

NLP-solver implementation

In stead of optimizing the parameters for the delay function directly (Equation 1.2), the phase function is used (see Equation 2.11). Although the use of the delay function to determine the minimum error values for specific parameters would be intuitively appealing and easily comparable with prior research result, the delay error does not play a direct role in the output power of the optical detector. The absolute values of the addition of the complex phase vectors determine the output power. As a third method for determining the error, power functions could be used. Although theoretically optimal, it has some practical disadvantages. Besides that, results show only small differences compared to phase tuning. Therefore phase tuning is used in the approximation algorithm [6]. The proper equation is shown below and is plotted in Figure 2.8.

$$\psi(f) = \arctan\left(\frac{\sin(2\pi fT + \phi)}{\sqrt{1-\kappa} - \cos(2\pi fT + \phi)}\right) - \arctan\left(\frac{\sqrt{1-\kappa}\sin(2\pi fT + \phi)}{1 - \sqrt{1-\kappa}\cos(2\pi fT + \phi)}\right) \tag{2.11}$$

Having an error function:

$$\mu = \sum_n \left(\psi_{total}(f_0 + f_{IF,n}) + 2\pi D(f_0 + f_{IF,n})\right)^2 \tag{2.12}$$

When we repeat the process for a wide range of delays and for a fixed number of

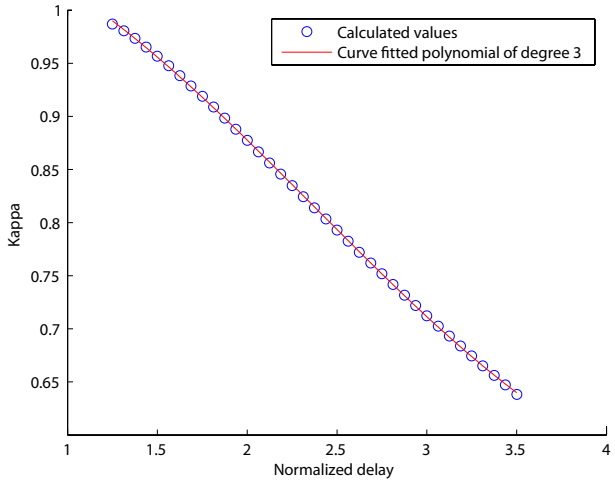

Figure 2.9: The combined result of multiple calculations for κ of a 1-ring delay element. The solid line represents the curve-fitted polynomial of degree 3. Note: only every fifth element of the calculated values is shown to avoid clutter.

rings, the combined results seem to form a more or less smoothly decreasing line (see Figure 2.9). This line is easily traced by a curve fitting function in a mathematical software tool, such as Matlab. As a result, we are left with a curve fitted polynomial function describing the delays versus κ and ϕ for the number of rings we want to process (Figure 2.9). To find the appropriate parameter values for κ and ϕ, all we need to do now is fill in the blanks in the new polynomial.

The polynomials are solely described by their coefficients, with the notion that the degree of the function is the number of coefficients minus 1, and that every term is used only once. The polynomials will however be less accurate when they reach the beginning and end of the range due to the curve fitting procedure. When the required delay is too large, the ripple will become too large, and the final delay is too much off. Of course, the amount of error a system can cope with is application dependent, and thus a suitable suggestion cannot be given in general. Therefore for delay elements having a number of ORR between 1 and 5, error plots have been created. Two of them are shown in Figure 2.10 and Figure 2.11. To prevent the system from curve fitting a function in a range that is not useful at all, only the part with minimal error is used. The range determination heavily depends on the required bandwidth. When larger bandwidths are required, the error will dramatically rise, and the near-errorless range of delays is reduced. An example for a normalized bandwidth $B = 0.16$ is shown in 2.11. Plots such as the ones shown are created for all 5 rings for $B = 0.09$, for which feasible delay ranges are constructed. For now, all minimum delays are $> n$, where n

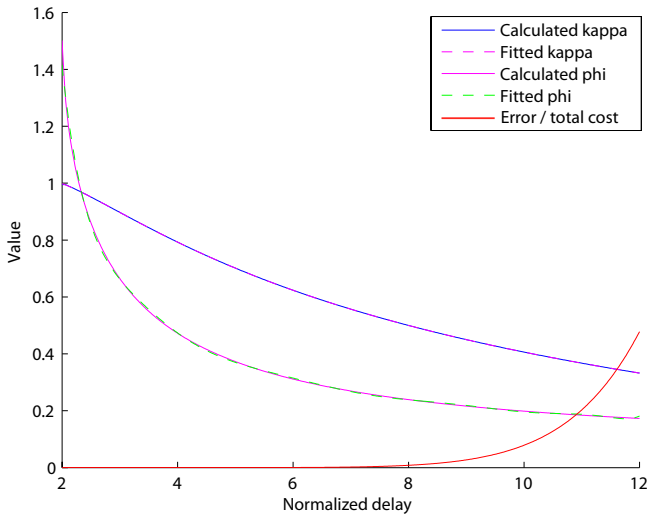

Figure 2.10: Fitted polynomials for a delay element with 2 ORRs for a normalized bandwidth of $B = 0.09$

Number of rings	Minimum delay	Maximum delay
1	1.0	2.0
2	2.0	7.0
3	3.0	14.0
4	4.0	25.0
5	5.0	30.0

Table 2.1: Theoretically feasible delays per delay element for $B = 0.09$

is the number ORRs in the delay element. The results are shown in table 2.1.

Due to the fact the the curve of large delay elements is not fittable any more with low order polynomials and inversed polynomials used in [6], all curves are fitted with a higher order polynomial of degree 10. Research shows that the error is minimal, and better fits are accomplished for delay elements containing more rings. Also, the approach is more generic, and better suits further extension of the number of rings.

The range of bandwidths that needs to be simulated can be adjusted in the precalculation Matlab scripts. For the remainder of this thesis, we will work with a value of 0.09, which coincides with a suitable bandwidth for the system that is being developed. Having a FSR of 14Ghz, the actual bandwidth for which the optimization is done is1.26 GHz.

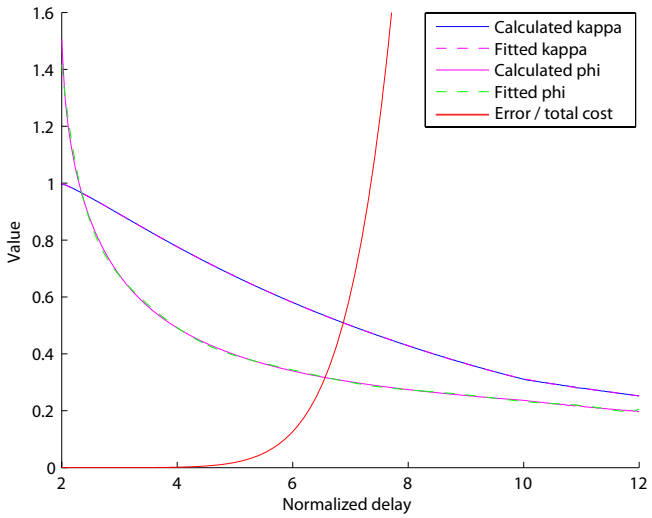

Figure 2.11: Fitted polynomials for a delay element with 2 ORRs for a normalized bandwidth of $B = 0.16$

2.3.4 Small delays

As discussed previously in the part about κ-limitation in Section 2.2.1, the value for κ is limited between 0 and 0.995. In theory, when κ equals 1, all the light is coupled into the ring, and after exactly 1 round trip fully decoupled back into the optical guide. The normalized delay would thus be 1. When κ is tuned to a very small value, only a part of the light is coupled into the ring, where the intensity starts to build up at the resonance frequency. As a result, a infinitesimal small frequency band is delayed for infinitely long, theoretically.

One possibility to achieve delays smaller than one RTT is to set κ at a proper value, and change the phase shift in such a way that the lower parts of the curve are within the bandwidth region of interest, or put differently, shifting away from resonance gradually. However, the NLP-solver persistently finds another set of optimal parameters, where there is a sudden phase shift from on- to off-resonance, and the region of interest is exactly between two resonance peaks (see Figure 2.12).

Optimal parameters have been determined using the same MMSE method as explained before, and again curve fitted for DEs containing 1 to 5 rings. Because of the relatively small degree of the fitted polynomial, a sudden change on the transition from < 1 to ≥ 1 would give rise to serious errors. Therefore, the ring settings calculation method chooses the correct data file containing the parameters according to the required delay. In Figure 2.12, an ORR response is shown with the parameters set for a normalized delay D of 0.5. The region of interest is centered around 0. In

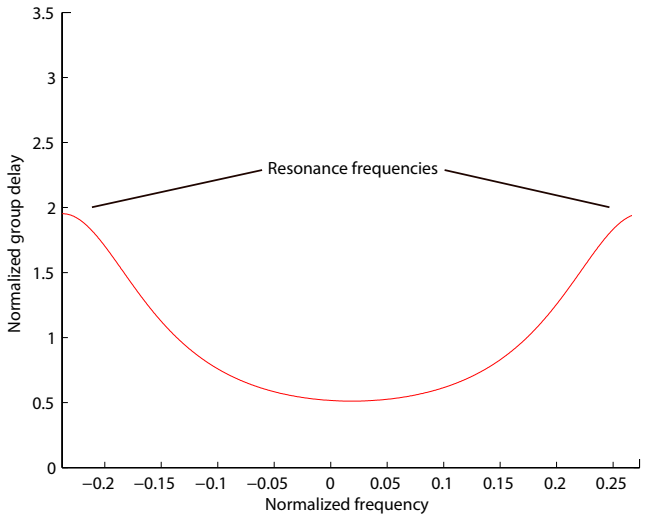

Figure 2.12: Response for a normalized delay of 0.5 for a delay element containing 1 ORR

Figure 2.13, a similar plot is shown, but now a delay element containing 2 ORRs. As we can see, the combined curve is very close to the required delay of 0.5. For a delay element containing only 1 ORR, the smaller-than-1 delays are not achieved by shifting the curve gradually. In stead, ϕ is shifted by π, and the κ is lowered, thereby creating higher peaks to the left and right of our region of interest. As a side effect, the response between the peaks lowers, and thus creating a delay smaller than 1. This is a direct consequence of the optimization process of the NLP-solver. Several attemps to adjust the boundaries of the parameter space did not change the results.

2.3.5 Alternative approaches

Although the methods described in this chapter work perfectly, alternative approaches should be investigated. This section briefly describes an alternative approach that could be further investigated in a future research project.

Other error functions

In stead of the current MMSE method, another measure of error could be provided to the NLP solver. For instance, one could calculate the maximum error in the function for the entire frequency range in stead of the MMSE currently used. Now μ becomes:

$$\mu = \max |\psi_{total}(f_0 + f_{IF,n}) + 2\pi D(f_0 + f_{IF,n})| \tag{2.13}$$

The computational complexity of this way of solving for the unknowns would likely

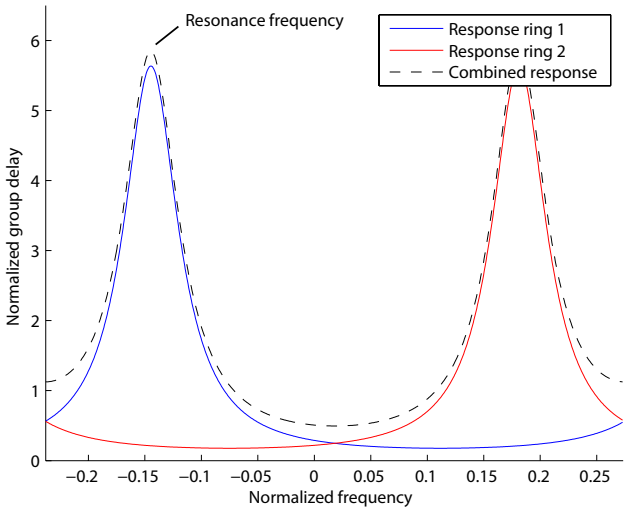

Figure 2.13: Response for a normalized delay of 0.5 for a delay element containing 2 ORRs

be slightly lower since we do not have to square the function and sum it. Besides, one can argue about the meaning of the error and if it suits the problem better than the previous solution. In our system, a high error in some part of the bandwidth is unacceptable, and thus this direct approach would deal with that by punishing high errors immediately for any given frequency.

To test the new error function, the coefficients have been precalculated for a DE containing 5 rings. Recall that because of symmetry, only 5, not 10, parameters have to be optimized. In this case, 3 κs and 2 ϕ. The results of the calculated parameters are shown in Figure 2.14. In spite of what one may have guessed, the calculation took about twice the time it took for the regular objective function, and the results are not smooth. The optimal solutions for different sets of parameters are further apart than when using the previous error function. The NLP solver evidently has a harder time trying to find an optimal solution. The function cannot be reliably curve fitted with a relatively low order polynomial, and thus the road previously chosen will be used.

2.3.6 Matlab scripts and API

For the delay element simulator, delay elements containing up to five rings have been precalculated for a large range of bandwidths and delays, depending on the size of the delay element. Because Matlab is not really suitable for containing large collections of data in a flexible way, a data structure consisting of nested *structs* is used. The diagram (figure 2.15) can be used as a reference model and should make it fairly easy to get the proper coefficients on demand. Since in Matlab, arrays cannot be indexed

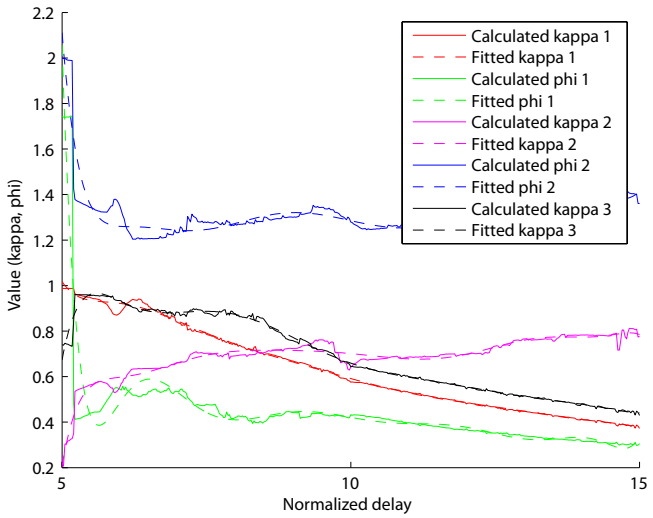

Figure 2.14: The results of the NLP-solver when using the alternative objective function

by a user defined value, a separate array *index* is created. This array contains all the delays that are precalculated. A simple search algorithm will find the proper index i for the required delays. The ring settings data structure at position i returns a set containing two subsets: kappas and phis. For a delay element containing n rings, n coefficient-sets in the form of $[a_n, a_{n-1} \cdots a_2, a_1, a_0]$ are returned. Each set contains the coefficients for creating a polynomial function in the standard form $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$, where a_n represents the n^{th} coefficient.

2.4 Manual

A complete system user manual and Application Programmers Interface (API) Documentation are included in appendix A. The manual is written in such a way that it can be used independently of this report. The API provides more insight in the functional hooks that can be used to extend the simulator, or to be called from other programs.

2.5 Summary and conclusions

When we look at the predetermined requirements, we were able to meet a substantial amount of them. The simulator is programmed in LabVIEW, combined with Matlab for parts of the code. Both programming environments are the de facto standard when it comes to developing simulation tools. Scalability, a very important requirement, is met. The simulator simulates delay elements of basically an infinite number of rings.

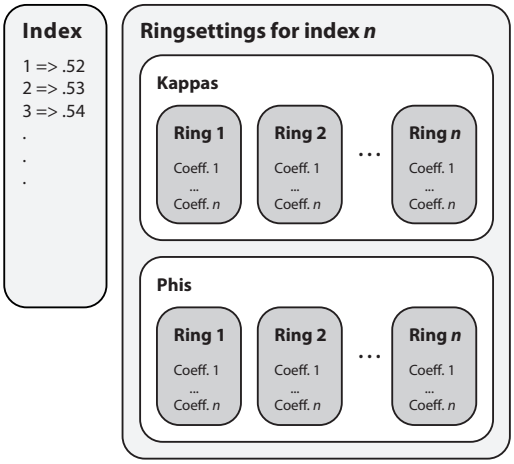

Figure 2.15: A representation of the data structure containing the polynomial coefficients.

Using higher powered computers, the coefficients of the approximation algorithm of larger delay elements can be calculated in less time, bringing the simulation of larger OBFN in the near future to be feasible. The simulator is build using an event-based programming model. The speed acquired by the use of this technique is tremendous, and also keeps the computer available for other tasks when there are no events at hand. Although a bit subjective, we think the GUI is a good example of a simple interface, combined with only the bare necessities for controlling the simulator. Finally, the software has been successfully tested on a the Microsoft Windows operating system, as well as on Apple OS-X. Both operating systems are very well capable of running the simulator.

Chapter 3

Design and Implementation of the OBFN simulator

Using the previously described delay element simulator as a building block, a complete OBFN can be modelled. This chapter describes the design and implementation of such an OBFN simulator. Not only does the software presented here acts as a simulator, but can also be used as a control tool for actively controlling the hardware amplifier board. Both aspects will be discussed in this chapter.

3.1 Requirements

As for the delay element simulator, the OBFN simulator also has several general non-functional requirements. They are:

1. *Maintainable* Although the simulator can be seen as a stand alone application, it would be nice if new features could be added in the near future by others. For that reason, a programming environment should be chosen of which knowledge is widely available.

2. *Scalability* The simulator must be designed to cope with a wide variety of optical chips, now and future versions. This means having the possibility of changing the number of inputs, changing the physical layout of the chip, changing the coaxial delays from the antenna elements to the actual chip, and last but not least changing the different ORR ring parameters like loss and circumference.

3. *Resource usage* The simulator must work fluently even on an every day computer. A proper design of the simulator makes the most out of the available CPU-cycles, and thereby maximizing the speed and responsiveness of the simulator.

4. *Ease of use* The simulator's GUI must be readily usable for anyone who has

knowledge of OBFNs. Clutter and unimportant input elements must be either hidden, or not be shown at all.

5. *Generic* Although the simulator is tailored to the available 8×1 OBFN chip, the simulator itself must be capable of simulating future chips with different types of tuning. For instance, the new liquid crystal version that will become available somewhere in the near future.

6. *Operating system independence* Because of the wide variety of operating systems commonly used nowadays, it would be nice to make use of programming environments that are available on different platforms.

3.2 OBFN simulator design

The OBFN simulator is again for the most part created in LabVIEW. For additional functionality and specific algorithms we will revert to MatLab. Appendix B contains a list of the functions that were written in Matlab, including documentation. This section describes the steps taken to design the simulator, starting with the initial design, the flow of data and the software structure.

3.2.1 UML model

A Unified Modeling Language (UML) model of the core of the OBFN simulator is shown in Figure 3.1. In this layered architecture [7], we can clearly see the aggregation relationships of the separate components that model the optical chip. An OBFN consist of multiple delay elements. These delay elements and their functioning are described in Chapter 2. Each delay element in its turn consists of multiple rings. In theory and for completeness, both the OBFN as well as the delay element class can consists of zero child nodes, where a child node is either a delay element in case of the OBFN, or a ring in case of the delay element. Each delay element is contained by exactly one OBFN, and each ring is contained by exactly one delay element. At the bottom of the figure, the *Path* class is shown. Since a delay element can be part of multiple paths from input to output within the OBFN, there is a *many-to-many* relationship. So each path can consists of zero or more delay elements, and each delay element is contained by zero or more paths. Structuring the OBFN like this corresponds greatly to the real world chip, and gives us some advantages later on.

3.2.2 Dataflow and structure

Figure 3.2 shows the dataflow of the OBFN simulator. Notice that the delay element simulator described in the previous chapter serves as a building block. Only the most

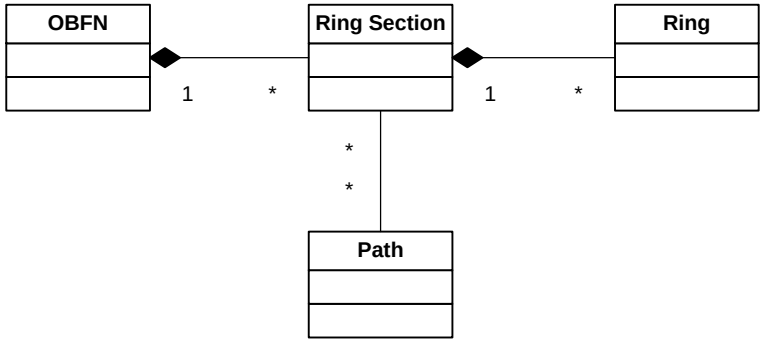

Figure 3.1: UML model of the layered architecture for the OBFN simulator.

important activity is displayed, being a change in the required AOA by the user. The activity is aimed at calculating the required voltages per channel for the amplifier boards, with only an AOA as input. Details about the intermediate steps are described below. The flow of data passes a few stages, mentioned in the following paragraphs.

Calculate $\Delta\tau$

When we assume that the antenna array is not curved, and the AE-spacing is constant, then for a specific AOA, the time between the arrival of the satellite signal $\Delta\tau$ at AE_i and AE_{i+1} is also a constant. This constant can be calculated as follows:

$$\frac{\sin\left(\frac{a\cdot\pi}{180}\right)\cdot d}{c} \tag{3.1}$$

where a is the AOA, d is the AE-spacing, and c is the speed of light, roughly $3{\cdot}10^8 {}^m/_s$. The idea is depicted in Figure 3.3. Of course, the $\Delta\tau$ can be normalized by dividing it by the RTT, which we will use for the remaining part of this thesis.

Calculate total path delays

Because of the fact that coaxial delay offsets are used to provide the means to tune for both positive as negative AOAs, the actual delay that has to be realized by the OBFN itself must be determined. Therefore, a small algorithm is used that determines the highest coaxial delay offset, and uses this offset as a reference point for determining the other path delays. The total path delay is acquired by:

$$\tau_{path\_i} = \max(\tau_{coax}) - \tau_{coax\_i} + (i-1)\cdot\Delta\tau \tag{3.2}$$

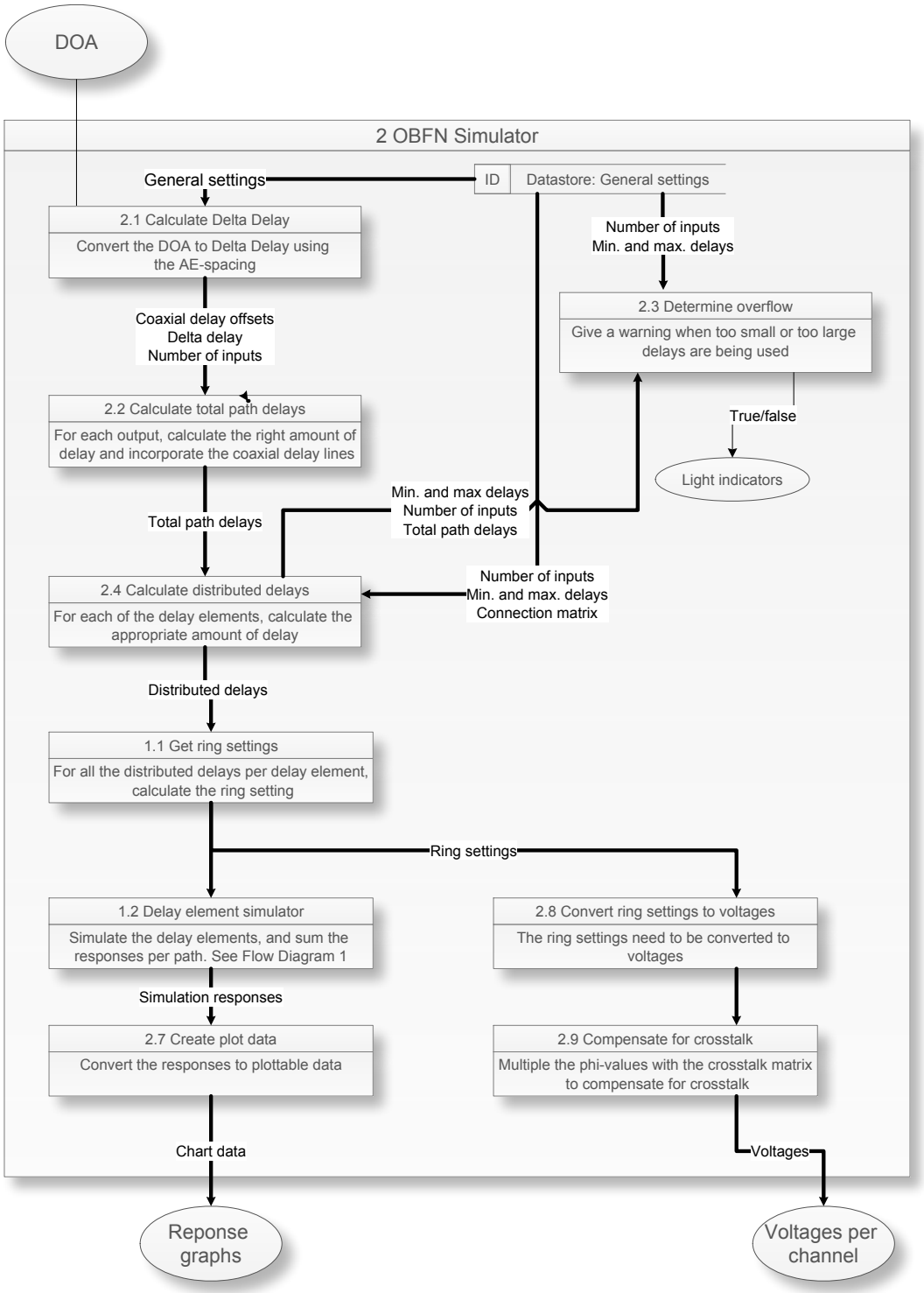

Figure 3.2: Dataflow for the OBFN simulator.

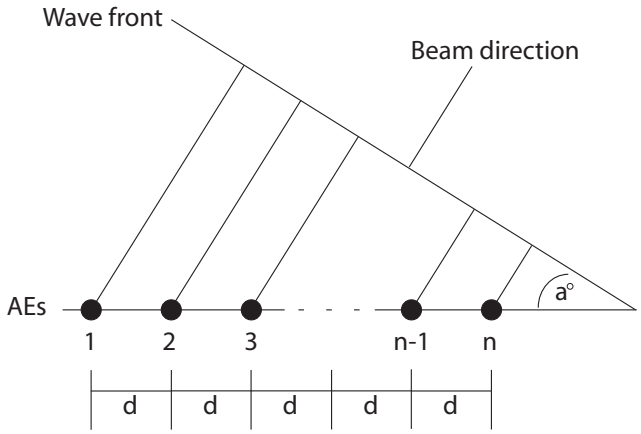

Figure 3.3: A general uniform linear PAA

where i is the number of the input, and $\Delta\tau$ is the required inter arrival time between the AEs. For example, when the maximum normalized coaxial delay is determined to be 3.5, and $\Delta\tau$ is for some specific AOA is determined to be 0.5, the total path delays can be calculated using the above formula. As an example, the total path delays have been calculated for a 4×1-OBFN, see Table 3.1. The results in the second last column are the delays that must be realized by the path to fully compensate all the additional delays caused by the coax cables. For instance, path 2 must realize a normalized delay of 2 with its one ORR. If we sum the fixed coaxial delays with the newly calculated delays, the resulting values, displayed in the last column, indeed show a $\Delta\tau$ of 0.5. More information on how to calculate the coaxial delays is available in Section 3.3.3.

Table 3.1: Calculation of the total path delays

Input i	$Coax_i$	$\max(\tau_{coax}) - \tau_{coax\_i}$	$(i-1) \cdot \Delta\tau$	τ_{path_i}	Total
1	3.5	0	0	0	3.5
2	2	1.5	0.5	2	4
3	1.5	2	1	3	4.5
4	0	3.5	1.5	5	5

Calculate distributed delays

Using the coaxial delay offsets and the $\Delta\tau$, the total amount of delay required per input of the OBFN can be calculated. This is done using the algorithm that will be discussed in more detail in Section 3.3.2.

Determine overflow

Using the delays per delay element, and the information that is made available by the user concerning the minimum and maximum delays for a delay element of a specific length, a warning will be issued. When a warning is given, calculations of the simulator are out of bounds, and cannot be used reliably. The software will however continue to work.

Get ring settings

When the delays for all the delay elements are known, the individual ring settings for each ring within the delay element can be calculated. Note that the number of the step in the flow diagram corresponds to Figure 2.1 in the previous chapter. This step is indeed a reuse of the model and code used for the delay element simulator. A SubVI[1] is created to abstract the inner workings of the previous simulator. For details about this step see Section 2.2.1.

Simulate the delay elements

Again, the SubVI of the delay element simulator is used to simulate the output response when the signal travels all the concatenated ORRs. The resulting responses for all the paths within the OBFN are summed.

Create plot data

The responses coming from the delay elements are being converted for plotting two graphs. The first graph shows the responses from all the inputs. For the second graph, the coaxial delay for each individual input are prepended to the responses, creating the final result. In this plot, the vertical space between successive cumulative path responses should be equal to $\Delta\tau$.

Convert ring settings to voltages

All calculated κs and ϕs have to be converted to voltage levels that can be sent directly to the amplifier boards. The conversion from the ring parameters to voltage levels is pretty straightforward and will be discussed in Chapter 5, where the interaction with the controller board is described.

[1] A subVI is equivalent to a function, subroutine, or method in other programming languages, and useful for encapsulating code that will be reused multiple time. A subVI is also used to develop hierarchical programs.

Compensate for crosstalk

The simulator has the ability to use a crosstalk matrix to compensate for linear crosstalk effects. These effect are caused for the most part thermally, meaning that the heat of heater a has an effect on some other heater b. Again, details about the compensation method will be discussed in Chapter 5.

3.3 OBFN simulator Implementation

3.3.1 Connection matrix

One of the requirements of the OBFN simulator is scalability. In theory, the simulator we have created is only limited to the number of rings within the delay elements for which precalculated parameter coefficients exist. In our case, we are thus limited to OBFNs containing delay elements no larger than five rings.

Each delay element is given a unique ID, denoted by DE_x, where x is a increasing number. For the case of the 8×1-OBFN, we have an exponentially increasing number of rings in each stage as shown in Figure 1.3. To label the delay elements in an orderly fashion, we use a DFS algorithm.

Formally, a DFS is a search that progresses by expanding the first child node of the search tree that appears, thereby going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it has not finished exploring. A depth first search can be performed on many types of graphs. In our case, we have a binary tree, and thus infinite recursion cannot occur. A short formal algorithm in pseudo-code is given in Listing 3.1 [8].

To make things more clear, the 8×1-OBFN has been traversed using the DFS algorithm. The order taken is accordingly to the DFS algorithm, and is displayed in Figure 3.4. This gives us the following 7 delay elements:

- DE_1: Delay line with 1 ring
- DE_2: Delay line with 2 rings
- DE_3: Delay line with 1 ring
- DE_4: Delay line with 4 rings
- DE_5: Delay line with 1 ring
- DE_6: Delay line with 2 rings
- DE_7: Delay line with 1 ring

To represent this tree, we use a matrix. Every row represents a path, and every column represents one of the delay elements. A value within the matrix indicates the number of rings contained that delay element. The values within one column must thus be the same, since this is in reality 1 delay element. When summing a row, the total

Listing 3.1: Pseudo code Depth First Search

```
preorder(node v){
        visit(v);
  for each child w of v
        preorder(w);
}

dfs(vertex v){
        visit(v);
        for each neighbor w of v
        if w is unvisited{
                dfs(w);
                add edge vw to tree T
        }
}
```

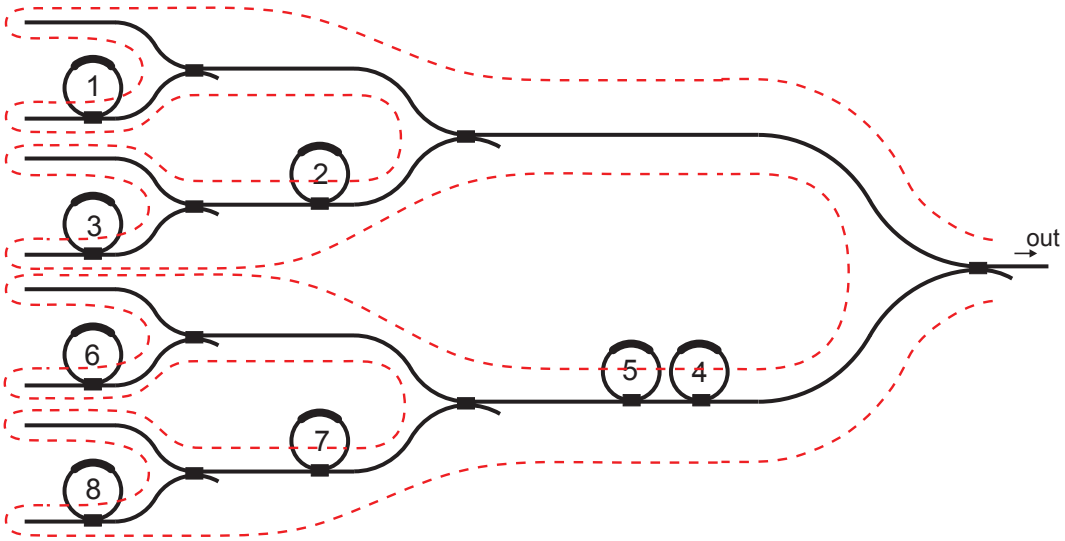

Figure 3.4: DFS walk of the 8×1-OBFN

number of rings in a path is known. Counting the non-empty positions in a column results in having the number of times a delay element is shared between paths. As a concrete example, take a look at Table 3.2.

Table 3.2: Connection matrix 8x1 OBFN

	DE_1	DE_2	DE_3	DE_4	DE_5	DE_6	DE_7
Input 1	-	-	-	-	-	-	-
Input 2	1	-	-	-	-	-	-
Input 3	-	2	-	-	-	-	-
Input 4	-	2	1	-	-	-	-
Input 5	-	-	-	4	-	-	-
Input 6	-	-	-	4	1	-	-
Input 7	-	-	-	4	-	2	-
Input 8	-	-	-	4	-	2	1

3.3.2 Delay distribution within the OBFN

For the simulation, a path instance is created containing multiple delay element instances. The creation of the delay element instances is done according to the value on position i, j in the connection matrix, where the i^{th} row is the representation of the path, and the value of position i, j the length of the delay element. The parameters for each delay element are obtained using the same methods as described in Chapter 2. The input signal is fed through all the newly created instances, where the results are added. When examining the connection matrix, multiple non-empty values in a column indicate a shared delay element among different paths. For example, DE_4 is being used by four paths. The delay of the fifth path solely depends on DE_4, which means that the value of DE_4 becomes a constant when the target delay for the fifth path is known. The required $\Delta\tau$ delay difference between the fifth and sixth path thus have to be realized by just one ring. To calculate the distribution of delays while keeping the shared paths in mind, an overflow algorithm is created. The code is given in listing 3.2.

We made use of the fixed structure of the binary tree structure and the DFS algorithm to create the connection matrix. One of the useful properties of the connection matrix when using this approach is that the number of rings in each row of the matrix is decreasing from left to right, from which we can benefit now. Note that this also means that when using other kinds of topologies, the overflow algorithm might need some adjustments.

Listing 3.2: Pseudo code overflow algorithm

```
Parameters:
totalpathdelays  = contains the total delays of all paths
minmaxdelays     = matrix of the minimum and maximum
                   delays achievable for delay elements
                   of different lengths
connectionmatrix = the connection matrix
numberofinputs   = the number of inputs of the OBFN

Algorithm:
Initialize a matrix 'delays' with zeros

do for each input i
  delayleftover = totaldelay = totalpathdelays(i)

  do for each delay element j of this path
    if connectionmatrix(i,j) is set and delays(i,j) is 0 then do
      if delayleftover larger than 0
        delayleftover = totaldelay - sum of delays in row i;
        if delayleftover is smaller than maxdelay
           newdelay = delayleftover;
        else
           newdelay = maxdelay;
        end if

        Then force this new delay to all the positions in the
        same column j where the connectionmatrix is not 0

      end if
    end if

    if delayleftover is larger than 0 then
    throw an "overflow error"

  end do
end do
```

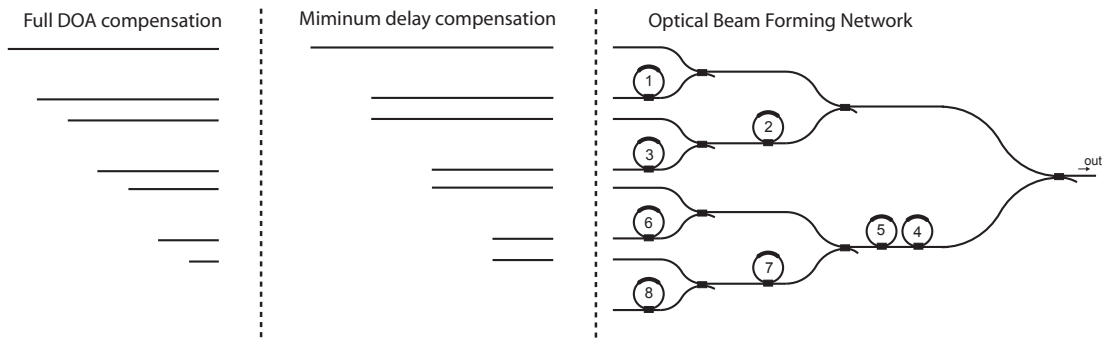

Figure 3.5: Coaxial delay lines prepended to the OBFN. The length of the lines is proportional to the required additional delay.

3.3.3 Dealing with offsets and negative AOA

Due to the fact that the physical realization of the optical chip is based on an asymmetrical binary tree topology, negative AOAs cannot be achieved without further adjustment of the system as a whole, since a negative AOA would mean a negative $\Delta\tau$, and the ORRs can only create positive delays. To solve this problem, extra fixed delay lines have to be prepended to the optical chip. This could be done using coaxial cables. Basically, a situation is now created where the zero-AOA delays are lifted from a normalized 0 to some value x. This way, AOAs from -60 degrees to 60 degrees can be achieved, despite of the physical limitations of the optical chip. In the simulator, the delays of additional coaxial delay lines can be set.

Given the frequency range of the system in Table 1.1, it follows that the antenna element spacing ($\lambda/2$) should be in the order of 1.5cm [9]. When combined with the AOA a ranging from -60 to 60 degrees, the delay between two neighboring elements should be tunable from $-\tau$ to τ. Using Equation 3.1 and given the speed of light in air $c = 3 \cdot 10^8$ and the AE-spacing $d = 0.015m$, we can calculate the tuning range to be roughly $2 \times 40 = 80ps$. For a 8×1-OBFN containing a total of 8 rings as displayed in Figure 3.5, and using a RTT of 80ps (note that this value is *not* related to the tuning range), the coaxial delays are calculated as follows:

Although the normalized delays of each ORR can be below 1 as we have seen in Section 2.3.4, using the coaxial delays we can prevent the ORR to be tuned below a normalized delay of 1 alltogether. In theory, when the MZIs of the ORRs are produced perfectly, the minimum normalized delay is exactly 1, which means a minimum total path delay equal to the number of ORRs within that path. When we have $\Delta\tau = 40ps = RTT/2$, and a minimal path delay in each path of the OBFN of 0, 1, 1, 2, 2, 3, 3,4 RTT respectively, a compensation has to be added that compensates for these minimum delays. This would enable us to receive a broadside signal (AOA =

0) by tuning all the rings to the minimal delay. Added to that, we need a normalized $RTT/2$ between all paths in the case of a maximum negative AOA. The total coaxial offset is now obtained by summing these two values. The results are displayed in Table 3.3. Although we are able to continuously tune a delay element between 0 and some maximum delay, the demands on the total delays of the delay elements are increased by these coaxial offsets, resulting in higher ripple and thus a more distorted signal.

Table 3.3: Additional coaxial delays

Path	Rings	Ring comp.	Max. neg. AOA	Total coax. delay (norm.)
Input 1	0	4	3.5	$4 + 3.5 = 7.5$
Input 2	1	3	3	$3 + 3 = 6$
Input 3	1	3	2.5	$3 + 2.5 = 5.5$
Input 4	2	2	2	$2 + 2 = 4$
Input 5	2	2	1.5	$2 + 1.5 = 3.5$
Input 6	3	1	1	$1 + 1 = 2$
Input 7	3	1	0.5	$1 + 0.5 = 1.5$
Input 8	4	0	0	$0 + 0 = 0$

3.3.4 Connectivity

The simulator can be connected to the hardware implementation to verify the calculated ring setting using the real measurement setup. Two methods are implemented, each having their own advantages. For both methods the commands to set a channel to the appropriate value can be send sequentially (several `wrch` commands), or in bulk (a single `wrchall` command). A switch button is provided to choose one of the two options.

Connect to the Java debug tool through TCP

The first method to connect the board to the simulator is through a TCP internet connection. For this to work, the debug tool needs to be running. See Chapter 4 and Appendix C for more information. By default, the tool is listening on TCP-port 4567 for incoming connections. The LabVIEW simulator can be set to connect to port 4567 on a specific IP-address. The address can be either `localhost`, or a real IP-address of some remote computer on the network or internet. When all set, the simulator tries to make a connection to the debug tool, which handles all requests. Replies from the controller board are sent back to the debug tool, but not to the simulator. In our experience, this way of communicating seems much faster than the next method, but is a bit more complicated to set up.

Direct connection to COM port

This is a more direct approach, but also noticebly slower compared to the method just described. The big advantage is the immediate feedback log within the simulator, and the fact that there is no need for an additional Java program running in the background. The board has to be connected to the same PC where the simulator is running on.

3.3.5 Complexity and upscaling

A real commercial system could consists of as much as 64×64 AEs. For a single row, a 64×1 OBFN could be used. Using the exponentially increasing number of rings in each stage, the maximum number of ORRs in a delay element is $n/2$. For $n = 64$, this means a delay element containing 32 ORRs. Fortunately, as we can see in Table 2.1, the feasible delay as a function of delay element length grows faster than linear, which means smaller delay elements can be used.

The simulator has been built to handle $n \times 1$-OBFN, with n an arbitrary number. Recall that the only limitation for now is the precalculated approximations for the ring settings - for up to 5 rings the coefficients were calculated. When analyzing the wiring in the block diagram of the simulator, it can easily be seen that the computational complexity of the system, or the time it takes to run recalculate a simulation, is roughly proportional to the connection matrix, meaning $O(mn)$.

3.4 Simulation results

When the OBFN simulator is started, all the needed steps to calculate the proper ring settings are performed, resulting in a plot as displayed in Figure 3.6. The settings used to run the simulation are displayed in Table 3.4. In the plot, we can see the needed response of all the paths within an 8×1-OBFN for an arbitrarily choosen $\Delta\tau$ of 0.5, which is equivalent to an AOA of 53 degrees. In Figure 3.7, the results for a negative AOA of -53 degrees is shown. Again, the time delay $\Delta\tau$ between the paths around the normalized frequency of 0 is 0.5.

Part of the OBFN has been reused in [10]. The simulator created there covers the complete path of a real data signal being sent and received by a PAA. The delays for the beam forming process were calculated using the LabVIEW code described in this and the previous chapter. Results show a nice gain, and the transmitted signal could be restored without errors.

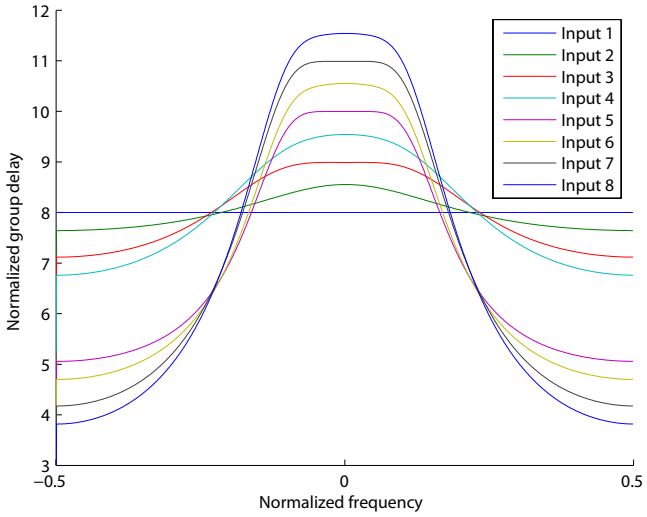

Figure 3.6: Simulation results for an 8×1-OBFN width a $\Delta\tau$ of 0.5 (equivalent to an AOA of 53 degrees

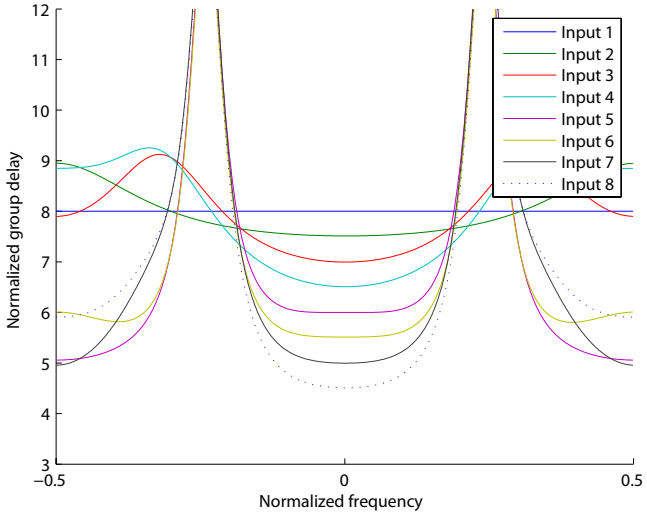

Figure 3.7: Simulation results for an 8×1-OBFN width a $\Delta\tau$ of 0.5 (equivalent to an AOA of -53 degrees

Table 3.4: Simulation settings

Property	Value
Number of inputs	8
Ring loss	.45 dB
RTT	0.08 ns
AE-Spacing	0.015 m
Normalized bandwidth	0.09
Connection Matrix	See Table 3.2

3.5 Manual

A user manual and API Documentation are included in appendix B. The manual contains the steps that need to be taken in order to properly setup the simulator. The API provides more insight in the functional hooks that can be used to extend the simulator, or to be called from other programs.

3.6 Summary and conclusions

A fully functional OBFN simulator has been built that meets the predetermined requirements. The simulator is again built in LabVIEW, and uses the simulator from the previous chapters in its core. With this simulator, OBFNs of different sizes and layouts concerning the number of rings used in each stage can easily be simulated. In theory, the simulator can be extended to larger OBFNs just by adding extra precalculated coefficients.

Chapter 4

Design and Implementation of the Microcontroller Software

Creating a dedicated software program running on the ARM chip itself creates more flexibility towards the controlling aspect of the OBFN system as a whole. The PC would serve as just a user interface for settings some parameters and the desired AOA. This way, the user interface part can easily be ported to for instance a mini PC for direct integration with the rest of the OBFN system.

4.1 Overview of the control system

The beam forming system is controlled by a control system. This control system consists of a combination of already existing hardware [4] and partly new developed software. A schematic of the control system that has been used during this project is shown in Figure 4.1. The control system takes care of applying voltages to the heaters on the optical chip and consists of a Printed Circuit Board (PCB) containing a general purpose microcontroller and one or more amplifier boards containing 32 amplifiers each. The microcontroller can be programmed to control the output voltage of each of the 32 channels. It contains instructions to set a specific value to a channel of a Digital to Analog Convertor (DAC) on one of the amplifier boards. The outputs of the DACs are then fed into amplifiers that boost the values to appropriate voltage levels to power the heater elements.

The microcontroller used is a Rowley CrossFire LPC2138 equipped with a LPC2138 ARM7 Reduced Instruction Set Computer (RISC) microprocessor from NXP. The programming environment for this microcontroller is called Rowley CrossWorks Studio, where the actual flashing of the microcontroller is done using a Universal Serial Bus (USB) port. After the flashing process, the microcontroller can be accessed via a virtual Recommended Standard 232 (RS232) or COM port of a PC. In our case, the virtual COM port runs over a standard USB port. Instructions on how to flash the

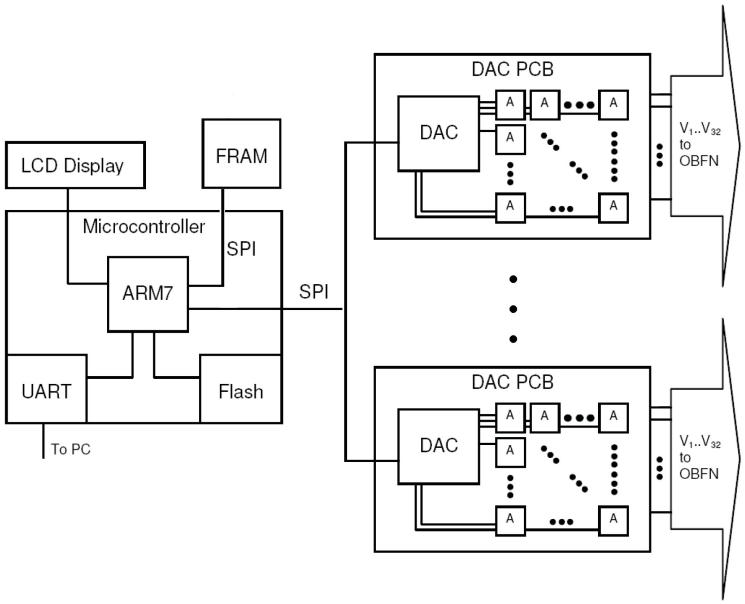

Figure 4.1: Architecture of the control system with on the left the microcontroller, and on the right the amplifier boards.

microcontroller are provided in Appendix C.

4.2 Controller software - PC

To be able to use the microcontroller, commands have to be sent to it by a PC. This section describes the tools that have been used to interact with the microprocessor for controlling and debug purposes.

4.2.1 Current software

The software we have used is not built from scratch, but is built upon an already existing framework. This framework allows for Serial Peripheral Interface (SPI) communication between the microcontroller, and the DAC. Also, methods for receiving commands from the COM-port from the simulator were already available. Although working quite well, the system would sometimes halt. This issue could be traced back to a memory leak in the code, which eventually caused a memory overflow. Because all communication stops when the code running on the microcontroller crashes, no further info could be given to the user.

To control the chip, a special tool has been developed in [12] that allows for separate tuning of all the available channels on the chip. A screenshot of the system in action is shown in Figure 4.2. Basically, the tool sends the same commands to the

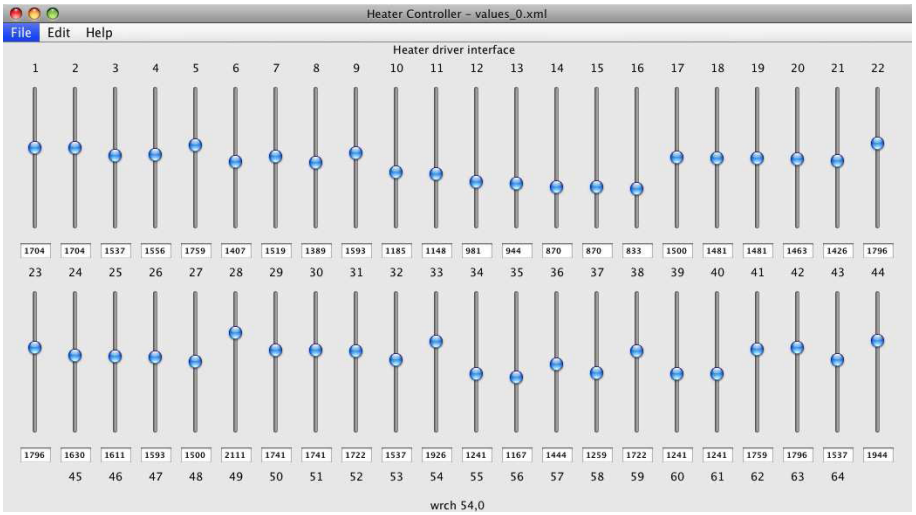

Figure 4.2: The slider tool (created by [12]) is able to control the voltage for each channel of the OBFN individually

microcontroller as the simulator described in the previous chapter. A small change in syntax has been made to better generalize the command structure, on which we will elaborate later on. Furthermore, some minor adjustments were made that makes it now possible to save settings to a file at any desired location and under any name. Finally, when loading a file, a single `reset` command is sent to the microcontroller to assure that the slider settings coincide with the actual voltages that are being applied.

One problem in the existing system is the fact that commands from the COM-port are arriving too fast. This causes the microcontroller the choke, thereby dropping commands and leaving the entire OBFN-system in an unknown state. That is, the sliders on the screen show different levels than the OBFN is actually set to at that moment. All these minor problems were taken care of, as we will see in the following sections.

4.2.2 Debug tool

To verify the correctness of the microcontroller code, parts of it have been tested on a development PC. Other parts were tested using a very simple tool, based on the slider tool shown in Figure 4.2. The GUI implementation was replaced by a simple Command Line Interface (CLI), which could easily be created thanks to the Model-View-Controller (MVC) design of the slider tool. In a MVC-design, there is a strict separation between the User Interface (UI) and the rest of the code. The debug tool allows us to send arbitrary commands to the microcontroller, similar to a standard Hyper Terminal connection, but offers additional functionality. For instance, the possibility to create test-cases and adding a proper delay between the commands that are

sent. Finally, the debug tool can be used to receive commands from the LabVIEW simulator, and send them to the connected hardware controller board. Not only do we obtain higher speeds, but also better logging capabilities of the output of the microcontroller. Apparently, LabVIEW only collects the beginning of large responses, despite of any buffer settings. Information on how to use the debug tool can be found in Appendix C.

4.2.3 Configuration

Both the slider tool as well as the debug tool have several configuration options. Below there is a short list of available settings that can be added or changed in the *settings.xml* located in the appropriate directory of either tool.

- *Nr_of_bars* (Slider tool only) The number of bars to display. The number of bars, and thereby channels is unlimited.
- *COM* The COM-port to connect to. Note that in general, a COM-port can be opened only once. Combined usage of for instance the simulator and the slider tool is therefore not possible.
- *commandparamsseparator* The symbol that separates a command from the parameters. These values must be changed when using older versions of the microcontroller software. All the current microcontrollers have been loaded with the newest version of the software, so the default settings do not have to be changed.
- *paramsassignmentsymbol* The symbol that is used for the assignment of a parameter key to a parameter value.
- *paramsseparator* The symbol used to separate parameters from each other
- *intercommanddelay* (Debug tool only) The time the controller software running on the PC will wait before sending a new command to the micro controller. Since at this moment there is only one way communication, the micro controller must have enough processing time to handle all the incoming requests. When set too low, an overload of commands can cause errors or unexpected results. The value is the time to wait in milliseconds.
- *resetdelay* (Debug tool only) One special command that requires some time to finish is the reset command, setting 0 volt on all available channels. The amount of delay to wait can be entered here. The value is the time to wait in milliseconds.

As an example, the command *wrchall:12=1200,13=1300,14=1400* sets the values of channels 12, 13 and 14 to 1200, 1300 and 1400 centivolts respectively. the ':' separates the command name from the parameters. The parameters are separated by a ',' and the key-value pairs are separated by an '=' symbol. In Listing 4.1 an example XML-file is shown for the debug tool.

Listing 4.1: Example configuration file

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Settings for console debug tool</comment>
        <entry key="COM">2</entry>
        <entry key="commandparamsseparator">:</entry>
        <entry key="paramsassignmentsymbol">=</entry>
        <entry key="paramsseparator">,</entry>
        <entry key="resetdelay">500</entry>
        <entry key="intercommanddelay">200</entry>
</properties>
```

4.3 Controller software - microcontroller

Apart from the software on the PC, some software must run on the microcontroller to control the amplifiers. This section describes several scenarios to gradually port the calculation process now done by the simulator to the microcontroller itself.

4.3.1 Implementation scenarios

There are several possibilites on where to make the separation of the functionality and responsibilities of the software on the PC and the microcontroller. Three scenarios have been evaluated. All have both advantages and disadvantages over the others. A schematic of the separation of responsibilities is shown in Figure 4.3.

1. Keep the microcontroller as simple as possible and only write values to channels
2. Store the crosstalk matrix and other chip characteristics on the chip
3. Implement the whole system on the chip, only AOAs have to be provided

The first option is the simplest, but also the slowest due to the communication speed. However, since we are in an experimental phase, this is not an issue at this moment. All the calculations of the voltage values for the individual channels are calculated by the simulator, and subsequently sent to the microcontroller. The microcontroller then simply activates the channel.

The second scenario can be interesting when using multiple chips of the same type. The hardware board has knowledge of the optical chip's characteristics, and only needs

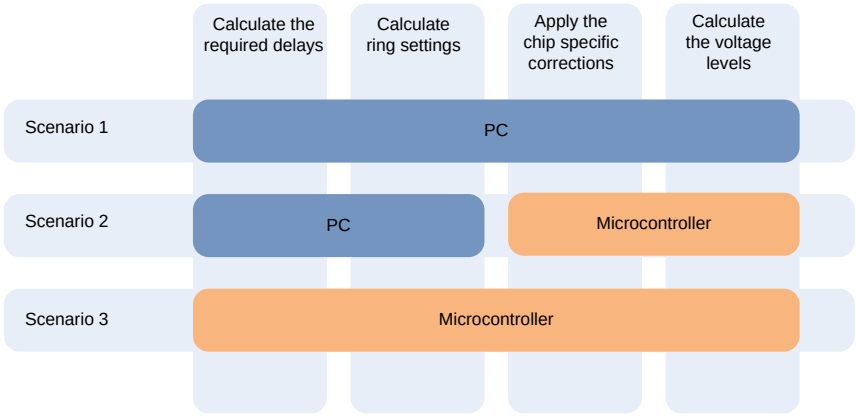

Figure 4.3: The three scenarios showing the responsibilities of the PC and microcontroller given a AOA.

to know the κs and ϕs for each channel to tune the chip correctly. Basically, the conversion of ring settings to voltages is moved from the PC to the microcontroller.

The last option is to create a full implementation of the entire calculation process for the chip. By sending only a AOA, the microcontroller calculates the voltage levels, thereby tuning the chip. This last option does however requires a lot of processing power, and a lot of memory to store the coefficients for the approximation algorithm as described in Section 2.3. Also, additional information such as the AE-spacing must be provided.

In our situation, only the first option has been used extensively. However, a start has been made for the implementation of scenarios 2 and 3. Although not ready-to-use yet, it should provide a good starting point for expanding the computational capabilities of the chip, and aiming for a more stand-alone version of the complete system. A full API-documentation for the functions that have been implemented are listed in Appendix C.

4.3.2 Command parser

A very simple mechanism has been build into the microcontroller code that accepts messages from the COM-port, splits them to *command* and *parameters*, and then splits *parameters* to *parameter1* ... *parameterx*. Using separate functions for handling all the command makes it easy to create combined commands, and to add functions for processing new types of commands later on.

4.3.3 Hardware-software communication

For now, the microcontroller support communication over a COM-port. In [12] a suggestion is done to use D2XX for communication. D2XX drivers allow direct access to the USB device through a Dynamic link library (DLL). Application software can access the USB device through a series of DLL function calls. The benefits of using D2XX would be faster data transmission.

At the moment, for controlling 16 channels, we need commands like:

> wrchall:1=1234, 2=1234, 3=1234, 4=1234, 5=1234, 6=1234, 7=1234, 8=1234, 9=1234, 10=1234, 11=1234, 12=1234, 13=1234, 14=1234, 15=1234, 16=1234

with a total length of 126 bytes. Using a COM speed of 115200 bits (14400 bytes) per second means being able to sent roughly 110 commands of this type per second. Keeping in mind that there is also some processing time involved for processing the commands on the chip, the communication speed is really not the issue here. Also, when focusing on gradually moving the responsibilities from a PC to the microcontroller, the level of communication would further decrease, thereby reducing the need for speed even more.

A big improvement would be to create a simple response parser like the one used in the microcontroller code that handles messages coming from the microcontroller. According to the type of return message, appropriate action can be taken or new commands can be sent. Going one step further is to create a reliable two-way communication channel, preferably over TCP using a new version of the microcontroller that is a little faster, has more memory, and can implement a TCP stack. A fully reliable two-way communication channel can be setup between any PC and the microcontroller using conventional networks. A good candidate would be the NXP

4.3.4 Floating point operations

The microcontroller that was used does not have a Floating Point Unit (FPU) for processing floating point values like 2.1283. Floating point values are needed for scenarios 2 and 3 as previously described. However, using external libraries, a software implementation can be used that mimics the operations of the missing FPU. To enable this feature, some specific compiler options must be added to the compiler within the CrossStudio development environment. The steps that need to be taken are explained in Appendix C.

4.4 Summary

The software now running on the microcontroller has improved stability, and is altered for further extension. Three scenarios were described, of which the first is used during the rest of this thesis. Handles and functions for implementing the other two scenarios have partly been implemented. API-documentation is provided in the Appendix.

Chapter 5

Measurements

As mentioned earlier in Chapter 3, the OBFN-simulator is also capable of controlling the real OBFN system by sending commands to the controller board. This chapter describes measurements that were performed to properly initialize the simulator for the specific OBFN under test. After that, measurements done to verify the correctness of the simulations are described. For all tests, the 8x1 SMART optical chip was used.

5.1 System overview

Due to the fact that we are working with high frequency optical waves, a small change of for instance temperature could have a large net effect on the measurement results. For that reason, a lot of measures have been taken to minimize the influence of fluctuations in the room temperature and fluctuations due to the heaters on the chip itself. The optical chip is mounted on a thick copper plate that is kept to 30 degrees centigrade using a Peltier element. A water cooled plate beneath guides away all the heat to a remote location where a fan cools the water. The system itself is placed on a very stable table. Finally, to prevent airflow in the room to change the local temperature, the entire system is enclosed by a styrofoam box. A photograph showing the inside of this box is shown in Figure 5.1. On the left, we see some modulators. The optical chip is positioned on the right.

5.1.1 Measurement setup

All measurements were performed using the same setup. To be able to compare new result with previous ones, the measurement setup used to calibrate the chip and determine the crosstalk is taken from [12]. The schematics of the setup are shown in Figure 5.2. The laser is connected to the current source (Curr) and a Temperature Controller (TEC). The optical chip acting as the Device Under Test (DUT) is connected to the controller board and a temperature stabilizer. The controller board is

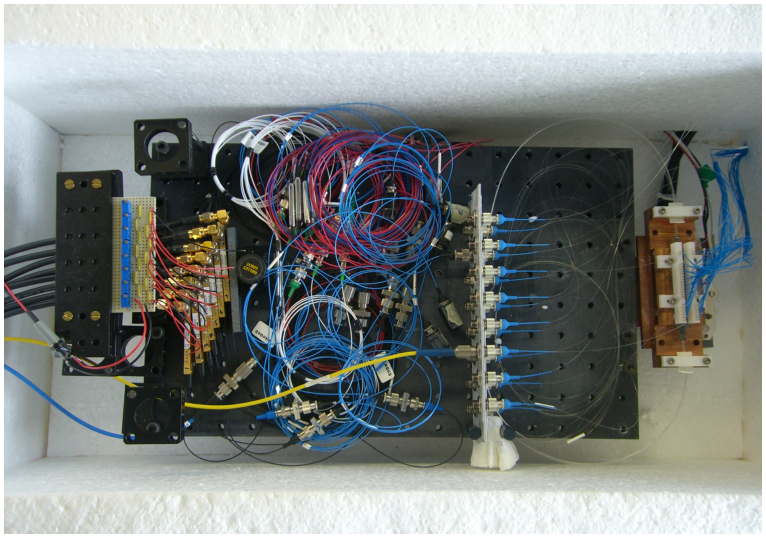

Figure 5.1: The inside of the styrofoam box

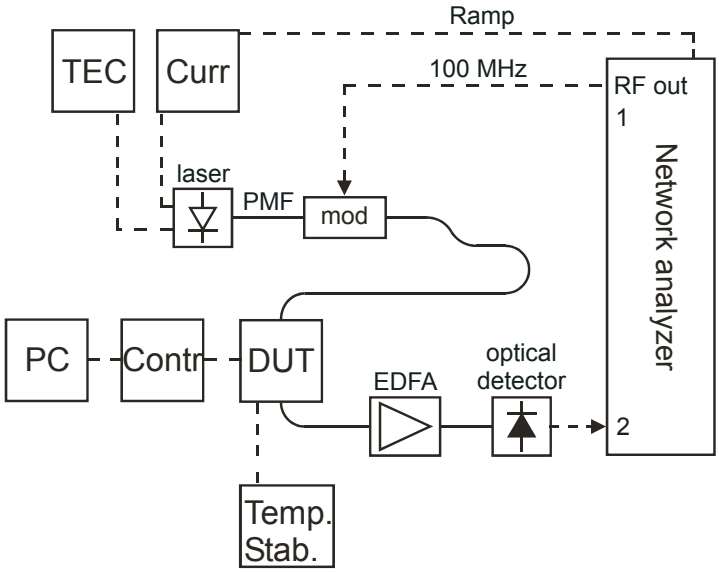

Figure 5.2: Measurement setup. TEC = Temperature Controller for the Laser. Curr = Current controller for the laser. Its current is sweeping, controlled by the Network Analyzer. DUT = Device under test, the optical chip in the photo between the laser and the Contr block. Mod = modulator. PC is the computer, interfacing the control system (Contr). EDFA = Erbium Doped Fiber Amplifier. Electrical wires are represented by dotted lines and optical wires by solid lines.

in its turn connected to an ordinary PC to get its instructions. The modulator (mod) superimposes a RF signal of 100Mhz onto the optical carrier and is then fed into the optical chip. The signal as it is leaving the optical chip is amplified using an Erbium Doped Fiber Amplifier (EDFA), after which it is detected using an optical detector and returned to the network analyzer. The network analyzer shows the time domain of each current sweep on the x-axis, which we will explain later, and the received power and phase shift as two separate traces on the y-axis.

Measurements are done using a laser current ramping technique. The laser current is ramping between two values, much like a sawtooth wave. As an effect of the current ramping, the laser frequency changes gradually over a certain bandwidth. The network analyzer measures a time window that is synchronized with the ramping, thereby showing the power and phase response at each frequency. Multiple FSRs are included within the frequency range. As a result, the network analyzer shows one plot containing dips due to the round trip losses in the ring at their resonance frequencies, and another showing the phase shift of the original signal.

5.1.2 Optical chip labelling

All the rings, heaters, inputs and outputs have been labeled by a unique number. All input labels start with *in*, the output labels with *out* and the rings with *r*. The heater channels do not have a prefix, and are referred to by only their number. These numbers correspond to the sliders of the slider tool as discussed in Chapter 4. The branch labbeled OSBF leads to the Optical Side Band Filter (OSBF), and has not been used. The complete labelling is shown in Figure 5.3. The dotted box surrounding the top left of the OBFN illustrates the part of the chip that was used during all measurements described in this chapter. The light from *out6* was connected to the EDFA, and from there to the optical detector.

5.2 Stability and Voltage Levels

For the measurements to succeed, it is very important that we are working with a stable system. Fluctuations of the output responses of the ORRs during long measurements will make those measurements unreliable. Two types of measurements have been done: an overall stability test of one ORR, and another test to verify the output voltages of the controller board with the values that were sent to it.

5.2.1 System stability

Stability tests can be very time consuming. All the measurements described in this chapter do not take longer than an hour, so we are interested in knowing the level of

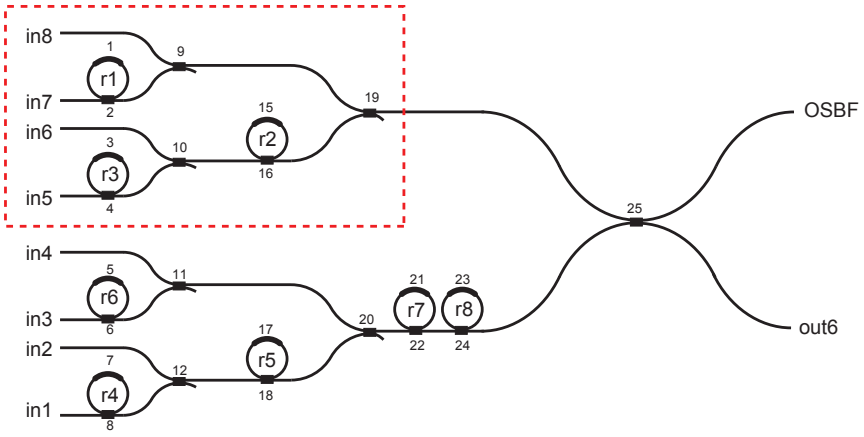

Figure 5.3: Labelling of all heaters, inputs and outputs of the optical chip.

stability in the course of that period. For exactly one hour, at a regular interval of 2 minutes, the output response of a single ORR *(r1)* was measured and saved. During the measurements, nothing of the setup was touched or moved, and all measurements were performed with the blinds shut to prevent direct sunlight, and windows closed for minimizing draft.

The current ramping was set to a ramp time of $t = 0.624708$ seconds, which roughly means having a coverage of $t/T * FSR = 32Ghz$, with an FSR of 14GHz.

The results of all these responses are shown in Figure 5.4. As we can see, the curves are almost exactly aligned, meaning that, for at least the measurement period, the system can be labeled as stable. To further investigate the stability, the fluctuations of the resonance frequency, and the changes of the maximum delay during the hour were examined. In Figure 5.5, we see that the resonance frequency drift is almost negligible, and does also not show any trend. The stability of the maximum delay (the height of the peak) is shown in Figure 5.6. Although the fluctuations are small, a clear trend emerges. The maximum slowly climbs from 0.1142 to 0.1157, creating an additional delay of 0.0015ns in the course of an hour. During the rest of the measurements, it is assumed that this small change does not significantly influences the final results.

In the past, there were difficulties in obtaining this level of stability. A light breeze would affect the system enormously. That is also the reason why it is wrapped in a styrofoam box. When measuring the output response of the next channel, some optical fibers have to be rewired. This rewiring means opening the box, which can influence the measurements. To minimize this effect, all three channels *(in5, in6 and in7)* were connected at once, and a selection of the channel under test was done by setting the combiners appropriately (see Table 5.1), and the proper coax cable was connected to the signal generator. The combiners were adjusted so that all other paths were

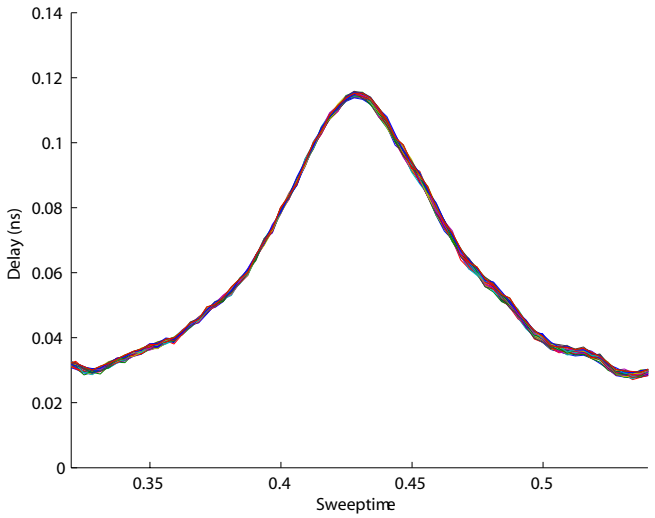

Figure 5.4: Overall stability of one ring (r1), measured at a 2 minute interval for 1 hour

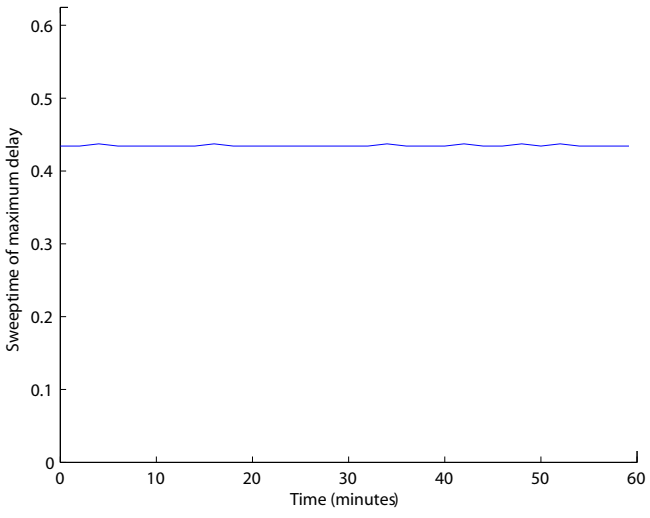

Figure 5.5: Resonance frequency drift of one ring (r1), measured at a 2 minute interval for 1 hour

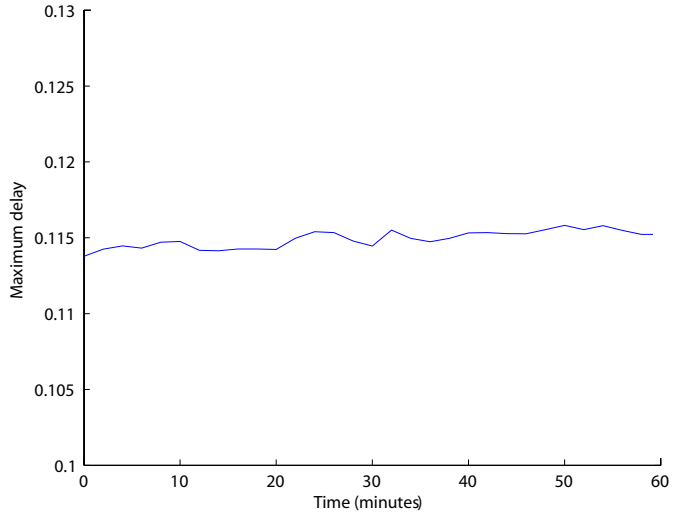

Figure 5.6: Maximum delay drift of one ring (r1), measured at a 2 minute interval for 1 hour

Ring	Heater	1	3	8	9	15	18
1 (coax 4, channel 7)		1065		19.68			
2 (coax 1, channel 6)			13.55		18.55	18.39	18.23
3 (coax 2, channel 5)			13.55			18.39	18.23

Table 5.1: Voltage levels used for measuring output responses by switching between coaxial cables outside the styrofoam box.

effectively disconnected, and could not interfere. Using this approach, the box did not have to be openened, which would in theory eliminate the impact of it on the systems stability. However, as further investigation showed, the instability was not caused by the system itself, but by heaters that were still set to a high voltage level, thus creating crosstalk. This situation was caused by a defective *reset* function that should have set all heaters to 0 volt. Unfortunately, this didn't always happen, leaving the system in an unknown state. The reset function was repaired, and further measurements showed that the system was indeed very stable. This means that future measurements do not have to be performed with the box closed at all times. Optical fibres can be rewired within the box, which will give better results, and eliminates any interference effects.

5.2.2 Voltage levels

The voltage levels that are calculated in the simulator, are sent to the microprocessor, which subsequently drives the DAC, after which the voltages are amplified. Because the system is very sensitive to small voltage changes (in the order of tenths of volts), the voltages must be stable in any case. Measurements for two channels *(in7 and in6)* of the 4×1 subset of the full 8×1-OBFN show that the voltages are hardly (only 0.001 volt) affected by voltages ranging from 0 to 30 volt applied to the other channels. We can, in this particular case, safely conclude that a voltage applied on one channel is not affected by a voltage applied on another.

As we will see later on, the relation between ϕ and voltage is captured by a linear equation in V^2, measured at two points. It is therefore important to know how the actual voltage relates to the calculated voltage. For the same two channels as above *(in7 and in6)*, this relation is determined. Figures 5.7 and 5.8 show the responses of the amplifier boards measured by a voltage meter. The voltage source has been set such that the output voltage of channel i when set to 30.00 volts was exactly 30.00 volt. Both the absolute difference and relative difference in voltage level have been plotted. We see that for low voltages, the output differs relatively much from the calculated voltage. Low voltages should thus be avoided as much as possible. Furthermore, we see that the measured voltage show an upward trend. This could be corrected for directly by a multiplier constant, either in the simulator or in the microprocessor. Examples of compensated voltages are also shown in the two figures, where a multiplier constant of 1.0131 has been chosen for both channels. The maximum absolute voltage difference now drops to less than 0.1 volt.

5.3 Chip characterization

To properly convert calculated ring settings to voltages, the properties of the individual ORRs must be known. This section describes the measurements performed to get the relation between ring parameter and voltages for the individual rings. During one day, the chip has been re-characterized a few times to see if any changes occurred. No significant changes were observed.

5.3.1 Kappa-calibration

For the simulator to calculate the proper voltage levels for a particular value of κ, the relation between kappa and voltage must be determined. We use κ to control the height of the group delay peak, where κ usually lies anywhere between 0.5 and 1, Lower values of κ create excessive delays which are undesirable. The value of κ is converted to $\phi_{coupler}$ using Equation 2.6 since, in theory, it is proportional to V^2.

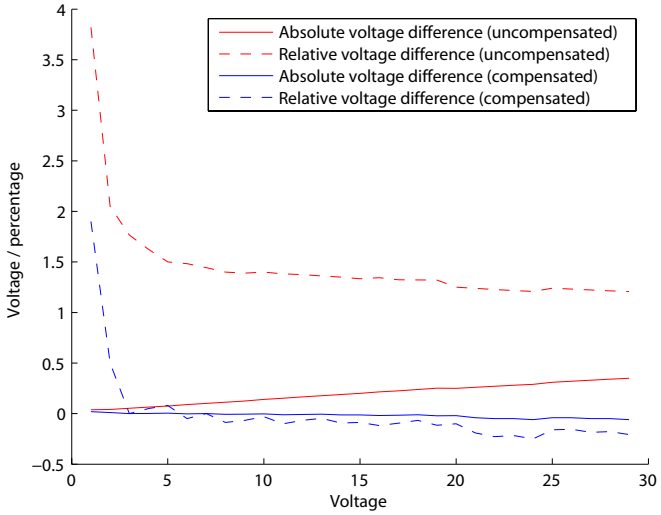

Figure 5.7: Voltage differences for channel 1

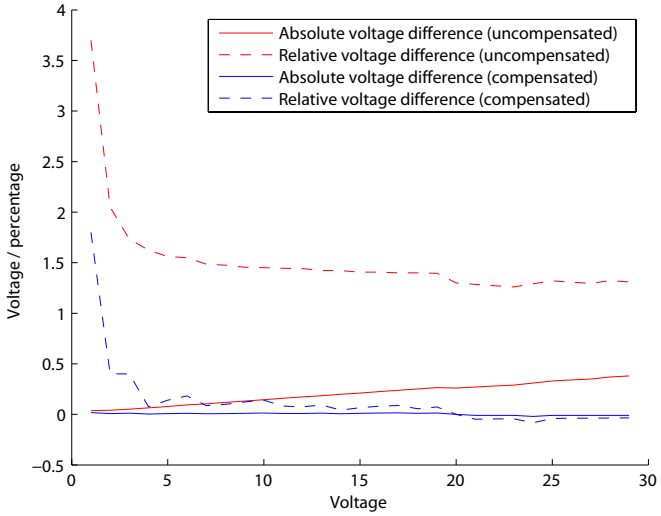

Figure 5.8: Voltage differences for channel 2

κ	τ_{norm}	$\tau(10^{-10}s)$	$\Delta\Phi^\circ$	$\Delta\Phi^\circ + offset$	V^2	$\phi_{coupler}$
0,98	1,3	1,04	-3,744	16,856	6,61	0,28
0,96	1,5	1,2	-4,32	16,28	8,06	0,40
0,94	1,6	1,28	-4,608	15,992	8,71	0,49
0,92	1,8	1,44	-5,184	15,416	9,4	0,57
0,9	1,9	1,52	-5,472	15,128	9,75	0,64
0,8	2,6	2,08	-7,488	13,112	11,29	0,93
0,7	3,4	2,72	-9,792	10,808	12,26	1,16
0,6	4,5	3,6	-12,96	7,64	13,1	1,37
0,5	5,9	4,72	-16,99	3,608	13,8	1,57
0,4	8	6,4	-23,04	-2,44	14,5	1,77
0,3	11,7	9,36	-33,696	-13,096	15,16	1,98

Table 5.2: Example measurements for determining the κ-voltage relationship of ring r1 (channel 2)

For the first 3 ORRs *(r1, r2 and r3)*, the values have been measured using the setup described earlier. The results of one set of measurements are shown in Table 5.2. First, using Equation 2.1, κ is converted to the maximum normalized delay. The delay is converted to a real delay in nano seconds by multiplying it by $0.08 \cdot 10^{-9}$. Using a signal frequency of $100 \cdot 10^6$ MHz with T $= 1/100 \cdot 10^6$ s, the expected signal phase shift is calculated by $\tau \cdot 360/T$. Using the output of the network analyzer, the voltage was adjusted to match the expected phase response. Finally, the $\phi_{coupler}$ equivalent of κ is calculated. The results of this process are shown in Figure 5.9. Note that the phase shift detected by the network analyzer shows an offset. This offset was determined using the method that will be described in Section 5.5.1. This offset tends to fluctuate, and should be repeated when doing comparisons of measurements. At this moment, the cause of the fluctuations is unknown, and should be further investigated.

The relations are *almost* linear in V^2, but because the slightest change of κ has a large effect on the actual group delay, especially for large delays, only the best possible fit is good enough. To determine what the impact will be on the loss of precision, an error measure is determined. The error value is calculated by taking the absolute difference between the lineair approximation and the measured value. The maximum error based on measurements of the first three rings *(r1, r2 and r3)* is 0.0512 rad. The small curvature of the measured slope has not been further examined, and for now, the resulting marginal error is not taken into account in further calculations.

Notice that the slope of the characterization of the third ring is negative. This is because the natural delay (the delay when 0 volt was applied) caused by the ring was too high. Because of this, the characterization was done in the range of π to 2π

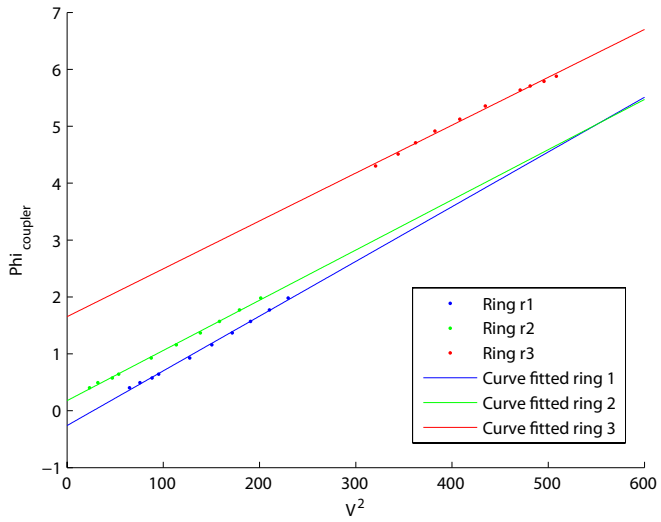

Figure 5.9: Result for determining the values of kappa in relation to the applied voltage levels.

Ring number	1	2	3
a $[rad/V^2]$	0.0096158	0.0088222	0.0084136
b $[rad]$	-0.26061	0.17725	1.6531

Table 5.3: Coefficients for the κ-voltage curve fit

of the tunable MZI (see Figure 2.5). Changing the voltage gradually gave a certain range where a delay could be achieved from the physical minimum, being slightly more than one round trip time, to infinite. This range is subsequently used. This should however not pose any problems, since each ORR is characterized individually, and specific calculations of the voltage levels for κ within the simulator is also done on a per-ring basis.

The plots have been curve-fitted with a linear function, resulting in a formula of the form:

$$\phi_{coupler} = a \cdot V^2 + b \tag{5.1}$$

The values of a and b for the first three rings are shown in Table 5.3.

5.3.2 Phi-calibration

In order to calculate the voltages, some fixed values depending on the optical chip need to be measured. The values $V_{\phi 2\pi}$ and $\Delta V_{\phi ref}$ for the set of ORRs are shown in Table

Ring number	1	2	3
$\Delta V_{\phi 2\pi}$	24.35	25.97	24.83
$\Delta V_{\phi ref}$	0	23.23	3.87
a $[rad/V^2]$	0.010596663	0.009315864	0.010190925
b $[rad]$ 0	5.027146946	0.15262847	

Table 5.4: Coefficients for the ϕ-voltage curve fit

5.4, where:

- $V_{\phi 2\pi}$: The voltage needed to obtain a phase shift of 1 FSR or 2π.
- $\Delta V_{\phi ref}$: Reference voltage needed to obtain a pre-set reference phase. This is to align the resonance frequencies of the rings.

The values for $V_{\phi 2\pi}$ are obtained at a κ_{ring} of 0.7. The voltage for the particular κ for the ring under test is determined by using the data from the from the kappa-calibration in the previous section. All other phase- and coupling heaters should be set to 0 volt for no crosstalk. This does mean however that the calibration process is slightly different than the one that is used in [12], where the focus was on calibrating at the equal *voltage levels* for κ.

With the two measured values, a linear function ϕ_{ring} as a function of V^2 is created as follows:

$$\phi_{ring} = a \cdot V_{\phi}^2 - \phi_{offset} \tag{5.2}$$

where

$$a = \frac{2\pi}{(\Delta V_{\phi 2\pi})^2} \tag{5.3}$$

and

$$b = \Delta V_{ref}^2 \cdot a \tag{5.4}$$

The equation for the output voltage V_{out} for a no-crosstalk situation now becomes:

$$V_{out} = \sqrt{(\phi_{desired} + b)/a} \tag{5.5}$$

The method described can be used in the case there is no crosstalk. We will see later on that the system does suffer from crosstalk, which needs to be corrected for. The crosstalk does not have an effect on the characterization. The values a and b can be entered in the simulator, since they are chip-specific. See the manual in Appendix B for more information on how to properly enter the a and b values.

5.4 Crosstalk

The tuning of the optical chip is done by applying heat to the specific parts of the chip to cause some change in the behavior of the ORRs. This tuning process of the optical chip suffers from crosstalk. Crosstalk is the unwanted effect that tuning of one ring has on another. For example, when tuning ring *r1* for ϕ, the heat that is created has some influence on the rings surrounding it. The smaller the distance, the greater the effect. This effect can be both positive and negative, meaning that other rings experience a phase addition, or phase subtraction. The positive effect is caused by some extra heat provided to one ring by another. The other effect is explained by [12] as being an electrical crosstalk effect. In both cases, compensation is necessary. The system we will use to compensate for crosstalk will be able to deal with basically any form of crosstalk, provided that the effects are linear. In that case, we can use a simple matrix multiplication to compensate the crosstalk effects.

The matrix would be a square matrix of size $n \times n$, where n is the number of heater elements of the chip that need compensation. The diagonal of the matrix is filled with the self-values: the effect of the heater that belongs to the ring without any crosstalk. All the other $n \times n - n$ values are the crosstalk factors. They can be either negative due to thermal crosstalk, or both positive and negative due to electrical crosstalk, depending on the polarity of the voltages.

In our measurements, a total of 3 rings needed to be tuned, having a total of 6 heaters. Fortunately, a large part of the crosstalk effects can be ignored. Firstly, the effect of any heater on a MZI can be ignored. The two branches of the MZI are so close together, that the spreading heat causes both of them to warm up roughly the same amount. The resulting phase difference is not influenced by this. Visual determination using the network analyzer confirms that there is no effect. This means that half of the crosstalk matrix values can be left blank.

Secondly, because of the large distance between ring *r2* and rings *r1* and *r3* (see Figure 5.3), the effects of the heat are almost unnoticeable. Compensation requires only change of a few hundredths of volts. For the sake of simplicity, these small effects will be ignored.

For our proof of concept, the effects of the κ-heaters on the other heaters were not taken into account. Note that they *do* have an impact on the final output response, and should be considered in following projects.

5.4.1 Measurement execution

The influence of each ϕ-heater element to another is measured using the steps below.

1. Characterize all ORRs on the optical chip once ($V_{\phi 2\pi}$ and V_{ref})

2. Set all rings to 0 volt.
3. Pick a ring i.
4. Tune ring i to 2π. Do not take offsets into account.
5. Pick a ring j (the crosstalker).
6. Tune ring j to 2π.
7. The phase of ring i now changes due to the change of ring j, and is independent of other rings (which are at 0 volt).
8. Compensate the voltage of ring i from V_a^2 to V_b^2 so it is back to its original position of 2π using the slider panel.
9. Convert the compensation voltage to a proper value using Equation 5.6.
10. Store the value on the (i, j) position in the matrix.

The conversion in step 9 for the effect from j on i is done using the following equation (the index of the proper ring property of Table 5.4 is shown between the parenthesis):

$$A(i,j) = \frac{a_i \cdot \Delta V^2(i)}{\Delta V^2_{\phi 2\pi}(j)} \tag{5.6}$$

where $\Delta V^2 = V_a^2 - V_b^2$.

Incorporating the crosstalk effect on Equation 5.2, now becomes:

$$\vec{\phi}_{desired} = \mathrm{A} \cdot \vec{V}^2_{out} - \vec{\phi}_{offset} \tag{5.7}$$

And the output voltage:

$$\vec{V}^2_{out} = \mathrm{A}^{-1} \cdot (\vec{\phi}_{desired} + \vec{\phi}_{offset}) \tag{5.8}$$

5.4.2 Results

Our measurements have been limited to a 4×1 subset of the full 8×1-OBFN. Only channels *in5*, *in6* and *in7* were used, containing in total 3 ORRs, and thus having 6 heater elements. The crosstalk matrix has been determined and is shown in Table 5.5. Diagonally, the a values appear, which were previously calculated during the ϕ- and κ-calibration process. A value in position (i, j) means the effect of j on i. Doing a left-multiplication with a column vector of uncorrected values results in having a column vector with the corrected values.

	1	3	5	2	4	16
1	0.010596663	0	0.000618814	0	0	0
3	0.000187465	0.009315864	0	0	0	0
15	0.000468741	0	0.010190925	0	0	0
2	0	0	0	0.0096158	0	0
4	0	0	0	0	0.0088222	0
16	0	0	0	0	0	0.0084136

Table 5.5: Crosstalk matrix. The numbers on the top and on the left denote the heater numbers.

5.5 Delay measurements

To investigate if the simulator with all its calculations is working as expected, the simulated results should match the output responses of the OBFN network. Using the same system setup as before, several measurements have been performed.

5.5.1 Determining group delay offsets

When a κ is tuned at a very low value, the delay starts to go to infinity at the resonance frequency. The delay to the immediate left and right of this peak is almost 0, which is confirmed by both the simulations and measurements. When we tune κ down a little more, the peak completely disappears in the output windows of the network analyzer. This phase shift is used as a reference value to calculate the actual group delay in nano seconds. This phase-shift can be determined for every output, which can be used for aligning the output responses. We noticed that by physically moving the optical fibers, the phase offsets changed quite dramatically, so any rewiring of the cables to different inputs or outputs of the optical chip has to be done with great care.

5.5.2 Single ORR

As a proof of concept, a measurement has been done for a single ORR *(r1)*, and thus 2 heaters. The results are shown in Figure 5.10. For a set of AOAs between 10 and 60 degrees, measurements have been done. To compare the results of the simulation and the measured values, all values are first denormalized to ns. The differences between AOA i and $i-1$ for both the simulation, as well as the measurements are shown in Table 5.6. T($a^\circ - b^\circ$) means the delay difference in ns between an AOA of a degrees and b degrees. With respect to the resonance frequency, we see a small shift of the resonance frequency to the right for larger delays, presumably caused by the lack off crosstalk compensation for the κ-heaters as mentioned earlier in Section 5.4.

	T($60° - 50°$)	T($50° - 40°$)	T($40° - 30°$)	T($30° - 20°$)
Simulation (ns)	0.0050	0.0062	0.0071	0.0079
Measurement (ns)	0.0059	0.0061	0.0068	0.0077
Absolute difference (ns)	0.0009	0.0001	0.0003	0.0002

Table 5.6:

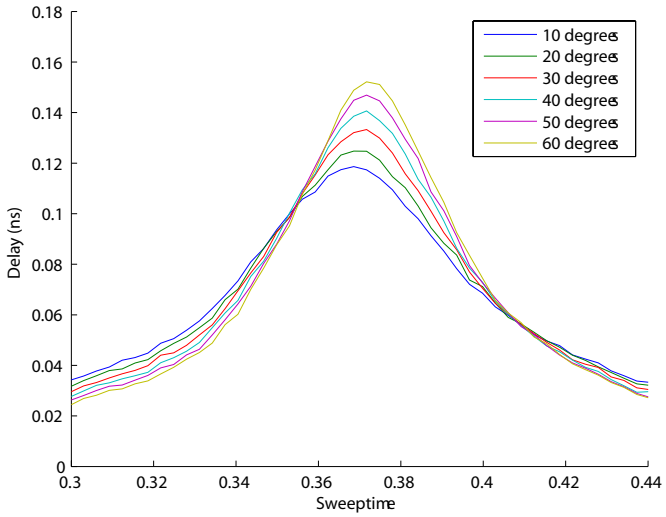

Figure 5.10: Single ORR response for several AOAs

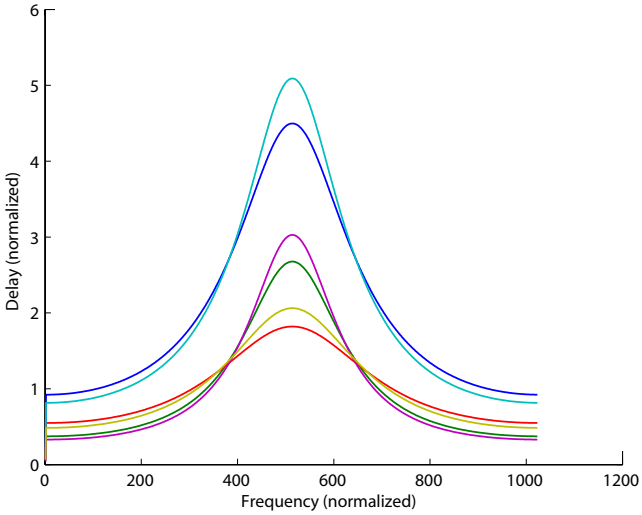

Figure 5.11: The response of a simulation of a 4x1 subset of the 8x1-OBFN containing 3 ORRs (r1, r2 and r3). The AOA has been set to 30 and 60 degrees, equavalent to a $\Delta\tau$ of 0,36 and 0,62 respectively.

5.5.3 4x1-OBFN

As a final measurement, the results of the simulator were tested on a 4×1-OBFN. The response of the OBFN for input in5, in6, and in7 have been measured. The results are shown in Figure 5.11 (simulated) and 5.12 (measured). Although the results seem quite good, again there is an offset to the left that is increasing with decreasing angles. This effect is most likely caused by the lack of correction of the κ-heaters on other rings. These measurements have not been performed yet. The effect of heaters 2, 4 and 16 on heaters 1, 3, and 15 are filled in with zeros at this moment.

5.6 Summary and conclusions

In this chapter, measurement setup, execution and results have been discussed. When the OBFN is properly characterized, the simulator seems to work very well for the tested 4×1-OBFN. The method used could prove to be usable for larger systems. Although the software system is ready to compensate all linear crosstalk effects, only half of them have been entered in the crosstalk matrix. Some measurements still have to be done, hopefully resulting in a perfect alignment of the resonance frequencies in the measured output responses. For our measurements, the stability of the system itself was sufficient. The rewiring of the optical cables caused the offset phase shift of the input signal to be altered. Whenever cables are rewired, phase offsets should be

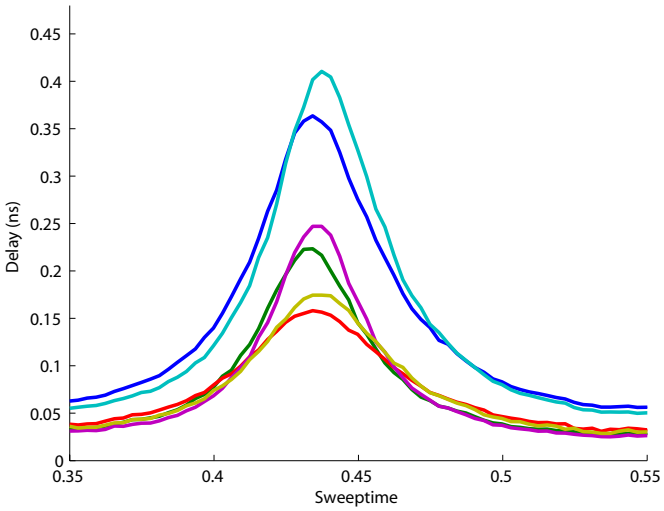

Figure 5.12: The response of measurements on a 4x1 subset of the 8x1 containing 3 ORRs (r1, r2 and r3). The AOA has been set to 30 and 60 degrees, equavalent to a $\Delta\tau$ of 0,36 and 0,62 respectively. The crosstalk correction has been included.

redetermined.

Chapter 6

Conclusions and Futher Research

This final chapter presents some conclusions and directions for further research and new that questions and ideas arose while doing this project.

6.1 Conclusions

The main reseach goal as stated in Section 1.3.1 was the creation of a maintainable and scalable software control system that can automatically tune all the parameters of an OBFN given only the direction of the incoming beam.

To achieve the goal, two simulators were written in LabVIEW to see if the underlying calculations would work in theory. The first of the two simulators is specifically designed to simulate the group delay response of delay elements with a variable amount of rings. The settings for the rings were aquired by using an approximation algorithm with precalculated values. The effects of a change of κ on ϕ, and the loss compensation by a change of κ have been incorporated. The end result is a scalable simulator capable of simulating delay elements containing a variable amount of rings.

The second simulator was an additional layer around the code of the first simulator, thereby creating a tool that can simulate an entire OBFN. The distribution of the delays accross the rings and the calculation of the voltages is all done with this simulator. The connection to the previously designed amplifier board makes it possible to apply these calculated voltages to the actual lab setup.

Finally, as a proof of concept, the simulator has been tested in a lab environment to see if the apprach taken could work, and would be a feasible candidate for further research. The first measurements using the voltages calculated by the control system look very promising. Also, the system is capable (with very small adjustments) of tuning future chip designs or using other tuning methods than thermo-optical.

To keep this report as generally applicable as possible, no specific applications were kept in mind when performing simulations or measurements.

6.2 Further research

During this research project, some interesting questions arose, and things came to mind that could possibly improve the system as a whole.

- Negative AOA are now fully handled by additional coaxial delay lines. These additional lines however put a higher demand on the delay that must be achieved by each ORR. Perhaps a more symmetrical OBFN design would solve this problem. Since more rings create a higher level of tuning complexity, some trade-off must be found.

- The delays are now calculated for a flat linear PAA. In real life, PAAs are often curved or have some irregular shape. The simulator could be expanded to cope with grid PAAs. The problem of handling curved surfaces could likely be solved in the simulator by adding an extra SubVI between the input of the AOA and the calculations of the required delay per path.

- Some of the crosstalk effects have not yet been measured. These effect do however contribute to the shift of the resonance frequency, and should thus be included in the crosstalk matrix.

- Although the crosstalk correction matrix works, it would be better to have some form of thermal feedback from each heater directly by the use of integrated thermomethers. Perhaps the resistence of the heaters on the current chip could be used for that purpose. By characterizing the optical effects of each tuning element for all temperatures, the heater-and-feedback combination would be responsible for achieving the desired optical effects. The crosstalk matrix and the relation from voltage to optical effect can then be eliminated, leaving an easier to calibrate and tune system, especially when the OBFN grows in size.

- The communication with the microprocessor is currently one-way. Possible problems due to timing are now solved by adjustable delays between the commands that are sent. Of course, this is only a temporary solution. Better is to have a two way reliable communication channel with the chip giving feedback. This could be implemented using a simple response parser in the Java debug tool code.

Bibliography

[1] [Online]. Available: http://www.lionixbv.nl/

[2] L. Zhuang, "Time-delay properties of optical ring resonators," Master's thesis, Universtiy of Twente, May 2005.

[3] C. Roeloffzen, "Passband flattened binary-tree structured add-drop multiplexers using sion waveguide technology," Ph.D. dissertation, University of Twente, 2002.

[4] M. Ruiter, "Design of a system for driving heaters on optical ring resonators," Master's thesis, University of Twente, 2006.

[5] W. D. Shoaff, "How to write a master's thesis in computer science," *Department of Computer Sciences, Florida Institute of Technology*, 2001.

[6] R. Blokpoel, "Staggered delay tuning algorithms for ring resonators in optical beam forming networks," Master's thesis, University of Twente, 2007.

[7] T. C. Lethbridge and R. Laganiere, *Object Oriented Software Engineering - Practical Software Development using UML and Java*, 2nd ed. McGraw Hill, 2001.

[8] S. Baase and A. V. Gelder, *Computer Algorithms*, Unknown, Ed. Addison Wesley, 2000.

[9] A. Meijkerink, "Functional design of the demonstrator chipset," deliverable for the FlySmart Project.

[10] M. Tijmes, "Simulation of a ring resonator-based optical beamformer system for phased array receive antennas," Master's thesis, University of Twente, 2009.

[11] T. Vrijmoeth, "Implementation of a heater-driving system," Master's thesis, University of Twente, 2006.

[12] J. van 't Klooster, "Context, design and implementation of a control system for ring resonator-based optical beam forming networks," MSc. Thesis, University of Twente, October 2008.

Appendix A

Delay Element Simulator Documentation

A.1 General information

The LabVIEW delay element simulator has been built using LabVIEW 8.5 on a Microsoft Windows XP Professional operating system. The simulator depends on several Matlab routines that were developed in Matlab 7r14 , so a proper installation of Matlab is also required. For LabVIEW to find the proper Matlab routines, Matlab should have a path reference (*file menu* → *set path*) pointing to the directories *simulator_delayelement* and *coefficients_lookup_table*. Due to caching of code, changes in MatLab code are not immediately effective in LabVIEW. The best way to circumvent problems related to cached code, is to completely restart LabVIEW.

A.2 Manual

This section will describe the delay element simulator from a users perspective.

A.2.1 Tour of the interface

A complete overview of the interface is given in figure A.1. Going clockwise starting at the top left, we see the output window showing the normalized group delay versus frequency, the power versus frequency, the dispersion versus frequency and finally the phase versus frequency. Note that all frequency axes are normalized. The main output window on the top left shows the group delay responses for each ring of the delay element individually, and also a combined group delay.

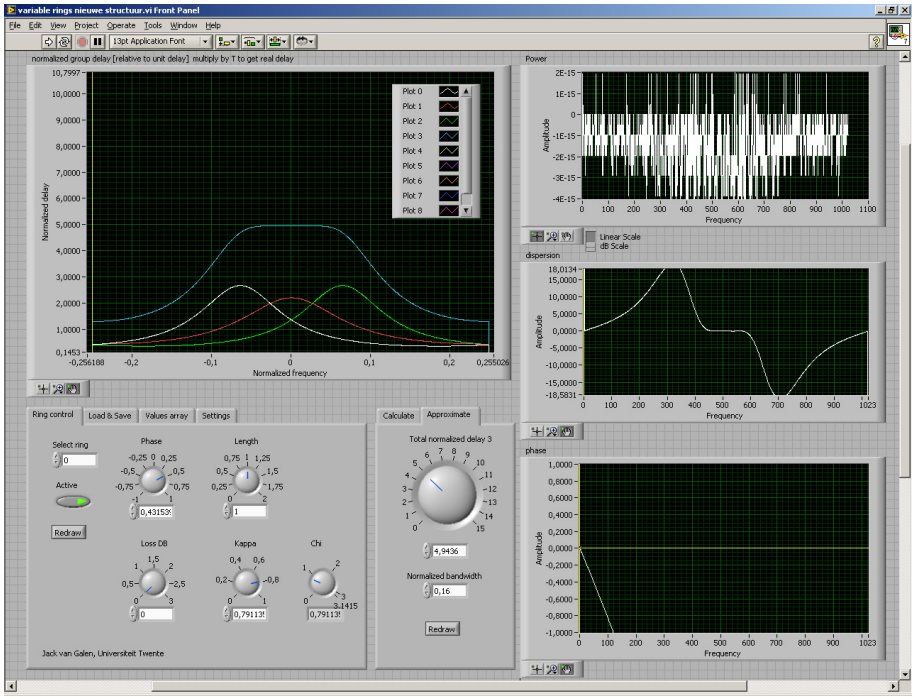

Figure A.1: Screenshot of the delay element simulator at startup

Tab Ring Control

The bottom left contains two control panels. On the left, a panel having 4 tabs is visible and has a combined input / output function. The first tab block is used to apply settings for each ring. By selecting a ring, the knobs are turned to the current settings. Turning a knob has an immediate effect on the output windows. The second tab block is used for automatically calculating the ring setting by using the MMSE calculation, and the approximation algorithm, both described in chapter 2.

Tab Load & Save

The *Load and save* tab shows two buttons which allows you to save and restore ring settings. After opening a previously saved file, all the ring settings will be restored. Also, the number of rings in the *Settings* tab will be adjusted according to the number of rings the settings were saved for.

Tab Values array

The *Values array* tab contains the values of each ring that are used for drawing the group delay output response in the top left window. Values can be individually changed here if needed. Changes are immediately processed. The chart array data is just for error checking purposes and can be ignored during normal operation.

Tab Settings

Several default settings are available for the delay element simulator. LabVIEW offers the possibility to store default values for all input fields, which can be used to store custom settings for later use. The settings are:

- *Number of rings*: The number of rings to simulate
- *Round trips*: The number of round trips used to model an ORR. A higher value will result in a better response at the cost of speed.
- *Input amplitude*: A constant 1 signal to model the input signal
- *Output amplitude*: The output amplitude with respect to the input amplitude. When losses are applied the output amplitude drops.
- *Output intensity*: The magnitude of the output amplitude
- *Equivalent Coupling Bends*: Adds extra coupling because of bends within the ORR
- *Actual Ring RTT* The round trip time of each ring in nanoseconds.
- *Center Frequency* The frequency in THz on which the main output window is centralized.

Tab Calculate

Using the MMSE method described in chapter 2, a real-time calculation is done to get the best ring settings possible for an optimal combined output group delay response for a given bandwidth and normalized delay. When a delay element is simulated that has a large number of rings, the calculation time could become large on older computers.

Tab Approximate

For the reason of large calculation times, an approximation algorithm was used (also described in chapter 2). Again, a normalized delay and normalized bandwidth serve as an input for getting an approximation of the ring settings. Changes to the delay are processed near real-time, after which the main window is updated to show the effect.

A.2.2 Usage examples

Example: we want to see the response of a delay element containing 2 rings and save the ring settings to a file.

1. Click the LabVIEW *play* button to run the simulator (top left corner)
2. Click the *settings* tab (lower left corner) and change the *Number of rings* to 2.
3. Click the *Approximate* tab (lower middle part) and choose a delay. The curve will change accordingly.
4. Current ring-settings can be saved for later use in the *Load & Save* tab. Other settings can be saved using LabVIEW's own save functionality in the *Edit* menu.

A.3 Pre-calculation scripts and API-documentation

The coefficients that are used to calculate the ring settings for a specific delay are pre-calculated using some fully detached scripts. The case of the 2-ring scripts is described here. The scripts for more rings are comparable. Although Matlab can more or less be used as an Object Oriented Programming (OOP) environment, the scripts are basic procedural scripts for the sake of easily testing and changing code.

The precalculation program has several files that contain the following functions.

calcall

This scripts can be run by setting the path to the appropriate directory, and issue the command *calcall*. Within the script, the bandwidth range and the delay range can be specified. The script will start to calculate all the settings for the whole bandwidth range, and save the results to *allresults2.mat*. Next, an appropriate structure as shown in figure 2.15 is saved in a file named *ringsettings_2_aboveone.mat*. This file is required for the lookup table which we will discuss later.

Parameters

- No parameters, just run *calcall*.

Return values

- File *allresults2.mat*: file containing the entire workspace
- File *ringsettings_2_aboveone.mat*: file containing the processed coefficients along some other useful info. See the documentation of the *calculatedOptimizedCoefficients_2rings* function for details.

calculatedOptimizedCoefficients_2rings

A function to calculate coefficients according to the phase-optimization function. Given a bandwidth, parameters for the whole delay range will be optimized.

Parameters

- *bandwidth*: The bandwidth to optimized for
- *fromdelay*: The starting point of the delay range (normalized)
- *todelay*: The end point of the delay range (normalized)

Return values

- *res1*: The coefficients for the first curve fitted function (in this case the κ)
- *res2*: The coefficients for the second curve fitted function (in this case the ϕ)
- *Z1*: The raw optimization results for the first parameter (κ)
- *Z2*: The raw optimization results for the second parameter (ϕ)
- *xas*: The x-axis used. This servers basically as an index for the delay range
- B: The bandwidth used
- E: Error values from the NLP solver for the whole delay range

phase_2opti

Sets some options and then calls Matlab's *fmincon* function to start the NLP solver. As an objective function to be minimized, $phasefun2(x)$ is used.

Parameters

- *points*: Number of points to check within the frequency range
- *bandwidth*: Normalized bandwidth
- *height*: The target delay
- *varargin*: Values used as a starting point for the NLP solver

Return values

- *coefficients*: The optimal coefficients for the given bandwidth and delay
- *error*: The error comparable to the ripple error.

phasefun2

The actual function to be optimized for a set of unknowns.

Parameters

- x Denotes an array of parameters to solve for

Return values

- mu The error for this function for the parameters tested

Appendix B

OBFN Simulator Documentation

B.1 General information

The LabVIEW OBFN simulator has been built using LabVIEW 8.5 on a Microsoft Windows XP Professional operating system. The simulator depends on several Matlab routines that were developed in Matlab 7r14 , so a proper installation of Matlab is also required. For LabVIEW to find the proper Matlab routines, Matlab should have a path reference (*file menu* → *set path*) pointing to the directories *simulator_OBFN* and *coefficients_lookup_table*. Due to caching of code, changes in MatLab code are not immediately effective in LabVIEW. The best way to circumvent problems related to cached code, is to completely restart LabVIEW.

B.2 Manual

This section will describe the OBFN simulator from a user's perspective.

B.2.1 Tour of the interface

A complete overview of the system interface is shown in Figure B.1. Unfortunately, the GUI does not fit on a standard screen resolution of 1280×1024, which makes scrolling necessary. We see three main parts. The top part consists of tab containing connection settings. The middle part takes care of all the settings related to the OBFN itself. Lastly, the bottom part serves purely to display the results. Please be aware of the caching of Matlab functions that LabVIEW performs. For substantial changes in settings the simulator has to be restarted. If you experience any errors, press the *Make current values default* option in the *Edit* menu and restart LabVIEW. Things should be working again now.

Lets discuss all the separate parts one by one, starting with the middle part, going down to the bottom part, and finally the top part.

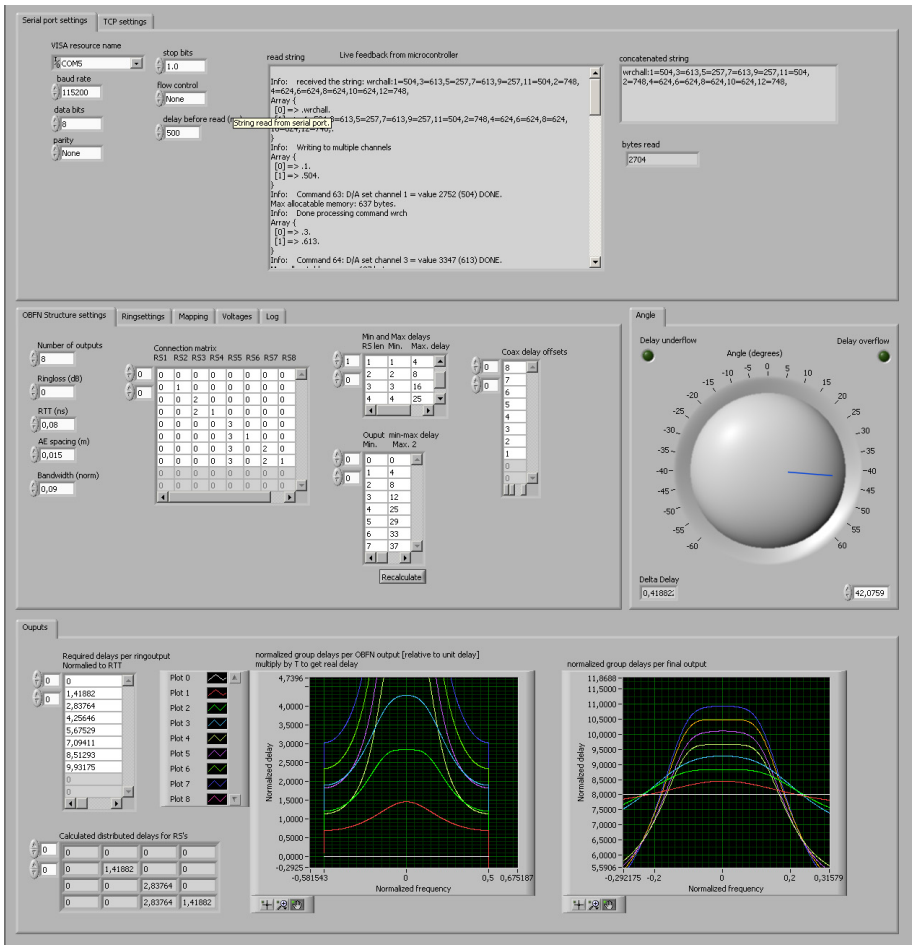

Figure B.1: Screenshot of the OBFN simulator at startup

Tab OBFN Structure settings

Multiple settings can be adjusted to fit the desired OBFN simulation. On the left, the number of inputs, the general ring loss in dB, the RTT in ns, the AE spacing in m, and the normalized bandwidth can be set. These general settings will be used in all the simulated ORRs.

Next in line is the connection matrix. Details on how to fill in the connection matrix are written down extensively in Section 3.3.1. Make sure that the actual size of the matrix in LabVIEW is adjusted to the connection matrix exactly, and that no extra empty rows or columns are active.

To the left of the connection matrix, there is a small matrix called *Min. and Max. delays*. This matrix contains the minimum and maximum delays for delay elements containing a specific number of rings, and is used to calculate the *Input min-max delays* when pressing the *Recalculate* button.

On the right of the tab, the coaxial delay offsets can be set. These are the offsets as discussed in Section 3.3.3 that compensate for the missing rings when tuning for angles smaller than 0 degrees.

Tab Ring settings

The ring settings tabs contains all the calculated ring settings for all of the rings in the entire OBFN. For now, *Active* and *Length* are always 1. On the right of the ring settings, two arrays of offsets are visible. The phase offsets for either ϕ and $\phi_{coupler}$ as discussed in Chapter 2 must be set here. These values will be used to calculate the proper voltages to send to the controller board.

Tab Mapping

The LabVIEW simulator labels each of the heater elements in a first come, first serve fashion. This could result in wrong commands being sent to the controller board. For this reason, a mapping can be applied to the heater elements used in the simulator. The first column represents the number used in the simulator, while the second column denotes the channel number used in the hardware setup.

Tab Voltages

The *Voltages* tab contains the crosstalk matrix as discussed in Chapter 5. The matrix has to be filled in manually. The simulator uses the crosstalk matrix to calculate the final output voltages displayed on the right side of the tab. These are the values in volts that will be sent to the proper channel (as set in the *Mapping* tab). Next to the final voltages array, a few switches and buttons are visible. When the *Enable hardware*

Property	Value
Baud rate	115200
Data bits	8
Parity	None
Stop bits	1
Flow control	None
Delay before read	500

Table B.1: COM-port settings in for the OBFN simulator

write switch is active, commands are sent to the controller board. When disabled, only the new voltages are calculated, but no commands are sent. One can choose between two *write types*. One uses a single combined command, and is thus faster. The other uses single commands for each voltage to be set. The latter can be used for debugging purposes. The last switch on the tab determines which connection type to use. Either directly via COM, or indirectly using the Console Debug Tool. For the fastest results, the combination *wrchall* and *TCP* must be used.

Tab Angle

The *Angle* tab contains a knob to simulate the angle of the incoming satellite signals. Note that this simulator is currently suitable for a one dimensional antenna array, and thus one angle suffices. The angle can be set between -60 and 60 degrees, corresponding to the system specifications as shown in Table 1.1. In the lower left, the normalized delay difference $\Delta\tau$ between the antenna elements is shown. The two indicators in the top corners warn the user when the delays for all the ring settings are out of range, using the *Min. and Max. delays* matrix on the *OBFN structure settings* tab. When one of the lights turns on, the systems operates outside the safe zone, and results can be unexpected.

Tab Serial port settings

To properly communicate with the controller board directly, the COM settings must be right. In Windows XP, the COM-port of the controller board can be determined by right clicking *My Computer* → *Properties* → *Hardware* → *Device Manager* → *Ports*. The other settings are shown in Table B.1. When communicating with the controller board directly using COM, the results are displayed in the feedback panel. The command that has been sent is shown in the *Concatenated string* panel.

Tab TCP settings

The second way to communicate with the hardware board is via a TCP connection to the console debug tool. Only two parameters have to be known, provided that the settings of the console debug tool are correct, and that it is running. The first parameter is the *Host name*. This can be either a host name as the name suggests, but can also be an Internet Protocol (IP)-ad

Tab Inputs

When all settings are done, and the *Run continuously* button has been pressed, the lower section of the simulator displays the results when changing the *Angle*. On the left, the array *Required delays per input* shows the total delay of each path within the OBFN. Below that, the distributed delays for all the delay elements are displayed. The matrix has a similar structure as the connection matrix.

The two graphs display the results of the actual simulation using the distributed delays. The left graph shows the output response of all the paths. The right graphs also shows all the input responses, but now the coaxial offsets are taken into account. This graph gives a good indication of the correctness of all the algorithms used in the calculation process. When results seem wrong, the look-up tables containing the coefficients for the approximation algorithms must be verified. Also, sometimes a restart of both LabVIEW and Matlab works wonders.

B.2.2 Usage example 1: setup a new OBFN simulation

Since the OBFN simulator is a bit more complex than the delay element simulator, special care has to be taken to properly set up the system. We will show you how to change the settings of the simulator from a 2×1 OBFN to a 4×1 OBFN.

Setting up the environment

First, the environment has to be set up properly. For more information about this process read Section B.1 of this appendix.

Change the OBFN structure settings

The following settings have to be adjusted in order to simulate a new and different OBFN.

1. *Number of inputs*: change the value to 4
2. *Connection matrix*: adjust the connection matrix to accommodate 4 inputs. For details, see Section 3.3.1.

3. (Optionally) *Min and Max delays*: if needed, change the *Min. and Max. delays* matrix.

4. *Coax delay offsets*: adjust the coaxial delay offsets to proper values

5. *Mapping*: if the simulator is used to control the controller board, the mapping has to be adjusted fit the heater numbers to the real channels.

6. *Crosstalk matrix*: finally, adjust the size of the crosstalk matrix to $2n \times 2n$, where n is the number of rings used in the entire OBFN.

Run the simulation

When all the steps above are done, the simulator is ready to be used. Press the *Run Continuously* button in the LabVIEW tool bar. Try to change the angle by turning the big knob, en see the results adjust almost instantly. When an error occurs, there is probably some miscommunication between LabVIEW and Matlab. Try to restart both programs and rerun the simulation.

Saving the new settings

All settings can be saved by stopping the simulation by pressing the red *Abort* button, followed by *Edit menu* → *Make Current Values Default*. The next time when the simulator is loaded, the settings will be restored.

B.3 API-documentation

This section will give a description of all the important public Matlab functions that can be used. Most of the functions that are used by the simulator have already been discussed in Appendix A.

calculateDelays

This functions calculates the proper delays for all the delay elements, given the total path delays of all paths.

Parameters

- *totalpathdelays*: The total path delays for all paths.
- *minmaxdelays*: The Min. Max. delays matrix holding the minimum and maximum delays values for delay elements of a specific length.
- *connectionmatrix*: The connection matrix representing the OBFN structure.

- *numberofinputs*: The number of paths in the OBFN.

Return values

- *delays*: An array containing the individual delays for all the delay elements in the OBFN. In combination with the connection matrix, this gives sufficient information to calculate the ring settings parameters.

Appendix C

Hardware Controller Documentation

C.1 General information

This appendix provides additional information for a proper installation and setup of the hardware controller.

C.2 Virtual COM-port driver

To let Java communicate with the COM-port, an operating specific package needs to be installed. For Microsoft Windows, instructions are given below:

1. Get the Java Runtime Environment from http://java.sun.com. If you would also like to edit and recompile some of the software, on of the development kits is needed that includes the Java compiler*javac*, for example the Java SDK.

2. Two files located on the CD that accompanies this thesis needs to be copied to the proper Java directories. Note that if you are installing an SDK, more than one Java directories will be created. Check your Windows path global variable settings to see which directory is actually used. Then copy the files from the CD *JDK118-javaxcomm* directory as follows:

   (a) Copy *win32com.dll* to the *bin* directory (e.g. c:/jdk1.6/bin)

   (b) Copy *com.jar* to the *lib* directory (e.g. c:/jdk1.6/lib)

   (c) Copy *avax.comm.properties* to the *lib* directory (e.g. c:/jdk1.6/lib)

3. You should be able to use the COM port now. If the steps above did not work, make sure you have copied the files the proper Java directory, and reboot your system.

C.3 Using the Slider tool

The slider tool can be started by navigating to the *Slider tool* directory on the CD, and double clicking click the slider.jar file. The jar file can be copied to any location, and is fully stand-alone.

C.4 Using the Debug tool

The debug tool can be started by navigating to *Debug tool* and double clicking the debugtool.jar file. The jar file can be copied to any location, and is fully stand-alone.

C.5 Flashing the micro controller

To flash the ARM7 micro controller, the CrossWorks studio application is used. A manual can be found in the *Manuals* directory on the CD, labeled *Starting the micro controller environment.doc*. When using a specific micro controller for the first time on a computer, a license needs to be installed. For all micro controllers that were in possession during the writing of this thesis, licenses were requested, and can be found in Appendix F.

C.6 Floating point operations

The micro controller does not have hardware floating point support. However, a software library can be included during compilation, that enabled a transparent use of floating point values anyway. A separate section in the *Starting the micro controller environment.doc* document described the exact settings that need to be set in order to make use of this functionality.

Appendix D

OBFN layout

Diagram of the board for reference purposes.

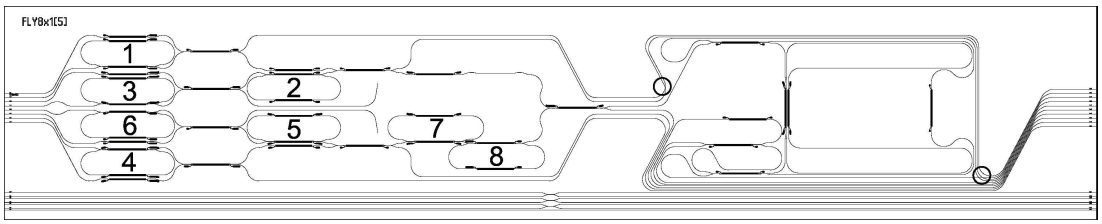

Figure D.1: 8×1 FlySMART chip layout

Appendix E

Ring, channel and waveguide data

Below an enumeration is given of the voltage sign per ring, their channels for ϕ and κ tuning, and the used input waveguide.

1. +, ϕ channel 1, κ channel 2, input waveguide 7.
2. -, ϕ channel 15, κ channel 16, input waveguide 5.
3. +, ϕ channel 3, κ channel 4, input waveguide 5.
4. +, ϕ channel 7, κ channel 8, input waveguide 2.
5. -, ϕ channel 17, κ channel 18, input waveguide 3.
6. +, ϕ channel 5, κ channel 6, input waveguide 4.
7. -, ϕ channel 21, κ channel 22, input waveguide 2.
8. -, ϕ channel 23, κ channel 24, input waveguide 2.

Appendix F

Rowley Crossfire Licenses

#020025:
0MO60-9JQ6G-IAJOX-08WL2-FC0SU-BC3CN-KGFB2-PKL5S-J0AFQ-UUZ7M+
M5664-QMXZY-UKUI0-94H0Z-SOG5Z-YIK5X-MAQYT-RGYZP-CM3FP-6CZMU

#020042
0F8AA-U8B43-KRKJK-46CY0-T9UAY-ZANEB-G1MQJ-IAGOI-BLHKN-PZGAN+
XO3TV-06T2F-IT4XE-3B14P-PX49P-ZQHKP-LS6NK-D0BPC-P8TQA-OWBBE

#020007
08VXK-NGCBS-BAF14-W78ZV-CW95V-E2EQI-TAZDS-9WQQ5-KKQ9X-AU4ZM+
KG7NG-ZWB5J-HJDHD-4BXTK-BEYML-F9OS9-FGI3P-OK4GR-Q8S64-HA4HW

#020011
0C4SZ-CKSBR-9TO4P-8HA4U-KC3CX-CNC48-B8065-P1HHG-I6XM3-FYQT7+
R1HB5-HNUYQ-46TVV-WXM2U-K3GOT-4ISCW-38EL2-OXAJO-6328M-QJ1OY

#02003B
0HU9N-IEP3B-20JOA-T97PE-0BZGS-AUGM5-P5H7P-R2H6K-SM61J-EMI6H+
PKYVA-VSQ9E-KWPST-N017O-E4EAF-6L1G7-3BZU5-5AG5K-28CSG-81AP1

#02002B
0Y5CP-9F5ZB-WGMF8-PK9B7-6Z7V2-M7WCT-I100Y-X8UIO-9ZNY0-GA171+
DHGKM-1LJDL-1CDM1-53JZK-P0IA5-8XXE2-9L0FM-BLUSZ-JR436-MDSG8

#02002A
0VI2C-FUBA5-9845R-CI265-38MKS-A7GNB-A30DI-BIQMC-V19GF-RAFUV+
JV03X-F3C2B-HIWQQ-YTDOX-T9KCL-ENG7F-DAVNG-THI4I-MLK8P-N22I8