Acknowledgments

First and foremost, I would like to thank my supervisor, Mats Høvin, for his first class support and advice. I could not have hoped for a better guidance. I would also like to thank my fellow students and staff, who helped me enjoy all these years at the university. I will not try to list names, in order not to omit anyone.

Additionally, I would like to thank all my friends, especially my coffee break pals for valuable recreation time and my family for love and support through approximately 2.73 decades. Finally, I would like to extend my deepest gratitude to Silje for her grammar knowledge and love.
Contents

1 Introduction 7

2 The Delta Sigma Modulator 9
  2.1 Short DSM History ........................................... 10
  2.2 DSM Basics .................................................. 11
    2.2.1 Quantization ............................................. 11
    2.2.2 Quantization Noise ...................................... 12
    2.2.3 Oversampling ........................................... 14
    2.2.4 Noise Shaping ........................................... 16
    2.2.5 Higher-Order Noise Shaping .............................. 18
    2.2.6 The Linear Model (LM) ................................ 19
    2.2.7 Noise- and Signal Transfer Functions (NTF and STF) 20
    2.2.8 Why Use Single-Bit DSMs? ............................... 22
  2.3 Stability of Higher Order Single-Bit DSM ........................ 24
    2.3.1 Stability Analysis by the Linear Model ............... 25
    2.3.2 Common Stability Criteria for Higher Order DSM ........ 27
  2.4 A Proposed Characterization and Verification of NTFs Stability and Quality ........................................ 29
    2.4.1 The Proposed Method .................................... 29
    2.4.2 Some Properties of the Method ........................... 31
    2.4.3 Stability Restrictions and SNR Performance of Chebychev Filter Design Techniques ................................. 33
  2.5 Today's NTF Design Methods .................................. 35
    2.5.1 DSM Topologies and Chosen Simulation Setting ........ 37
    2.5.2 Classical Chebychev NTF Design Example ............... 38
  2.6 Motivation for a Heuristic Search for NTFs .................... 39
    2.6.1 No Iron Clad Design Methodology is Available .......... 39
    2.6.2 Flexibility of Parameters ................................ 39
    2.6.3 DSMs for Special Purpose ................................ 40
    2.6.4 Signal Generators ....................................... 40
    2.6.5 Adaptive NTF Implementation ............................. 40
    2.6.6 Power Consumption ...................................... 41

3 The Search 42
  3.1 Some Definitions ........................................... 44
  3.2 The Fitness Function - Describes the Search Space .......... 45
    3.2.1 The Coding of the Input to the Fitness Function ....... 46
    3.2.2 Fitness Measurement - The output ....................... 47
CONTENTS

3.3 Constraining the Search Space ........................................... 48
  3.3.1 Constraining by the Fitness Function Coding ................. 48
  3.3.2 Local Constraining Inside the Fitness Function ........... 51
  3.3.3 Summarize .......................................................... 52
3.4 Determination of the Fitness Measurement .......................... 53
  3.4.1 Some Consideration of the Fitness Measurement ............. 53
  3.4.2 Using an SNR Measurement as the Fitness .................... 53
  3.4.3 Smooth Search Spaces ............................................ 55
    3.4.3.1 Length of Input Signal or Amount of fit-points ....... 56
    3.4.3.2 Mean of Several Different Input Signals .............. 57
    3.4.3.3 Concept on Dynamic Amplitude .......................... 58
    3.4.3.4 Smooth Search Space Conclusions ........................ 58
3.5 The Heuristic Search Algorithms .................................... 59
  3.5.1 Basic of Heuristic Search Algorithms ........................ 59
  3.5.2 Random Search (RS) .............................................. 60
  3.5.3 Hill-Climbing (HC) .............................................. 61
  3.5.4 Genetic Algorithm (GA) ....................................... 62
  3.5.5 Hybrid GA ...................................................... 63
  3.5.6 Random Restart Hill Climb (rrHC) ............................... 64
  3.5.7 Best of Random Restart Hill Climb (brrHC) .................. 65
  3.5.8 Dynamical Constrained Random Restart HC (dcrHC) ........... 66
  3.5.9 Search Algorithm Conclusion ................................ 67

4 The Results of Searches .................................................... 68
  4.1 Comparison Between Searched Designs and Peak Classical Designs 68
    4.1.1 Maximum SNR for an OSR of 64 and Modulator Order 5 .... 68
    4.1.2 A Search for an NTF With High maxU Tolerance ........... 71
  4.2 Discussions of the Search Results ................................ 72

5 Conclusion ........................................................................... 73

6 Further Work ....................................................................... 75

A Source Code .......................................................................... 79
  A.1 Source Code for DSM engines (mex code for matlab) ........ 79
    A.1.1 mexGen.c ....................................................... 79
    A.1.2 mexEfo.c ....................................................... 80
    A.1.3 genCheck.c ................................................. 81
    A.1.4 efoCheck.c ................................................. 82
  A.2 Source Code for the Proposed Characterization Method ....... 83
    A.2.1 CVntf.m ........................................................... 83
    A.2.2 maxU.m ........................................................... 84
    A.2.3 snrM.m ........................................................... 85
  A.3 Source Code for the Search Algorithms ............................. 86
    A.3.1 newAlians.m (Random Search) ............................... 86
    A.3.2 HillClimb.m .................................................. 86
    A.3.3 brrHC.m ...................................................... 88
    A.3.4 dcrHC.m ...................................................... 89
  A.4 Source Code for a Fitness Function Example ...................... 91
    A.4.1 efoRADZM.m ................................................. 91
## CONTENTS

A.5 Source Code for Some Needed Functions .......................... 92
   A.5.1 zp2ntf.m and ntf2zp.m .................................. 92
   A.5.2 ntf2gen.m and ntf2efo.m .................................. 92
   A.5.3 DSsin.m ..................................................... 92
   A.5.4 rad2ntf.m, rad2zp.m and D$S$zero.m ....................... 93
   A.5.5 snm.m, sy.m and blh.m ................................... 93
   A.5.6 DSfreqNtf.m ............................................... 96
   A.5.7 stabCheck.m ............................................... 97

B Paper Submitted to Norchip 2006 ................................ 99

C Paper To Be Submitted Shortly ................................. 105
CONTENTS

6
Chapter 1

Introduction

Delta-sigma modulation [7] (DSM) is a very popular technique for making high-resolution analog-to-digital and digital-to-analog converters. These oversampled data converters have several advantages over conventional Nyquist-rate converters. The features are achieved by the concept of noise shaping and utilizing more linear coarse quantizers.

As the title suggests, this thesis will present a heuristic search to find stable high order delta-sigma modulators. We will in other words cover two different special fields, heuristic search and delta-sigma modulators.

To achieve stable operation for high-order modulation so called non-monotonic loop-filters are needed. The specification of these loop-filters is normally given by a noise transfer function (NTF).

With the high modulator order we want to decrease the over sampling rate (OSR) while maintaining a high signal-to-noise ratio (SNR) performance. The design challenge is to get an as high as possible SNR while maintaining modulator stability. By a not stable modulator we mean a modulator that exhibits large, although not necessarily unbounded, states and poor SNR compared with the ones predicted.

The design of the loop-filters has no easy formula because of a lack of good design rules. The Linear Model [7] explains the noise shaping behavior but it has been difficult to make good stability criteria [11]. This is especially a big concern for high-order single-bit modulators. Today’s designs must therefore be thoroughly simulated to ensure stability and this makes the optimization of the designs a tedious work.

Today’s design methods can be loosely grouped in the following methods:

1. Using classical filter design techniques like Chebychev and Butterworth to produce the NTFs. The performance and stability are then simulated afterwards. If the stability or SNR performance is not satisfied we must post adjust the filter design parameters and simulate its behavior once more.

2. Another approach is to optimize more arbitrary NTFs while maintaining stability by restricting the design after some proposed stability criteria. But the criteria must be conservative to ensure stability.

Both methods have the potential to share one common disadvantage, the designs
are restricted. It is not guaranteed that filter design techniques make optimal NTF shapes and in method number 2 the criteria must be conservative to ensure stability.

Figure 1.1: Today’s methods can only use a restricted NTF area. Can optimal or interesting NTF exist outside these areas?

This paper presents an automatic design process by letting a search algorithm, guided by the linear model, directly explore the landscapes of the nonlinear world of quantization. Such work has, to the authors’ knowledge, not been presented before.

To enable us two make a thorough comparison between arbitrary NTFs, a characterization method was needed. This is not directly trivial and no methods were found in literature. A proposed method is therefore presented. The idea is formulated in a paper which is submitted to Norchip 2006, see the appendix.

The search algorithms Genetic Algorithm, Hill Climbing and hybridGA, a hybrid of these two, were tried out. A Random Restart Hill Climber with a combination of constraining the zeros and rejecting NTFs with too high out of band (OOG) values, led to an effective method for finding stable high-order DSMs with arbitrary specifications. As we will show the design approach found stable modulators with higher SNR performance compared to classical filter design techniques.

The presented heuristic search for stable high-order DSM was successful in our view. It was found to be an aggressive way to find high SNR performance given certain parameter restrictions. The design approach is therefore reproduced in a paper which will be submitted to a conference shortly.

The chapters in this thesis are grouped as follows. The delta-sigma basics and the motivation for using search are presented in chapter 2. Chapter 3 treats the search basics and describes the construction of both the fitness function and the search algorithm. Chapter 4 covers search results and compare these with classical filter design technique solutions. Chapter 5 gives the conclusion. The appendix at the end contains source codes and the two mentioned papers.
Chapter 2

The Delta Sigma Modulator

One of the most common elements in modern electronics is dataconverters. Their function is to translate from analog to digital values or vice versa. The most intuitive approach is the traditional Nyquist-rate converters but it can be a problematic design to realize because of its difficult component matching implementation. For instance the realization of a 16-bit Nyquist converter with a low reference voltage is difficult, since it will be nearly impossible to make precise enough resistors to satisfy the 16-bit resolution in VLSI. These implementations must be trimmed and are dependent on high complicated precise analog circuitry, i.e. a design with an expensive and restricted implementation.

A remedy for this, especially for application that needs high-resolution for relatively low-frequency signals, is the use of the delta-sigma modulator (DSM).

\[ \text{Figure 2.1: Delta-sigma modulation} \]

The basic concept of delta-sigma modulation is to use feedback for improving effective resolution of a coarse quantizer. In this way it aims to predict and correct the next quantization error. A simplified DSM operation can be seen in figure 2.1. The input is fed to the DSM which outputs a coarse oversampled quantized representation of the input, in this example single-bit. The bit-stream can then be digital or analog filtered for the wanted purpose.

The DSM is both used for digital to analog converters (DAC) and analog to digital converters (ADC). The concept is the same, only that input and output in figure 2.1 is respectively analog and digital for ADCs and digital and analog for DACs.

Though the DSM principle is elegant it can involve a tedious learning curve. This chapter will try to explain the basic concepts of the DSM operation. First we must explain the basic operation of quantization and how we can gain resolution. Then we will go in further details about the operation and analysis of
the delta-sigma modulator. We will especially discuss the problem of modulator stability. Here we will also present a proposed method of characterizing different modulators against each other. At the end of the chapter we will explain what motivated for a heuristic search to find stable high-order DSM.

Let us start with a short history.

2.1 Short DSM History

The first description of the use of a feedback for improving a coarse quantizer was given in a patent by Cutler [1] filed in 1954. This system was further analyzed and improved by Spang and Schultheiss [2] in 1962 and resulted in a system called error feedback only. Also a so-called Delta modulator was proposed in 1952 by Jager [3]. But both these systems had serious practical implementation problems (as ADC).

The name delta-sigma modulator was introduced by Inosa, Yasuda and Murakami in 1962. They proposed to add the loop filter to the front end of a delta modulator inside the loop and called it the delta-sigma modulator [4]. This method overcame the implementation problems.

The name $\Delta \Sigma$ was intended to reflect the fact that the system first took a difference ($\Delta$) and then integrated ($\Sigma$). There are some that rather want to call it $\Sigma \Delta$ to reflect the fact that the system can be represented as the cascade of integrators ($\Sigma$) and a $\Delta$-modulator. Both names are in active use, so a researcher looking for literature has to be careful incorporating both names in the search. This thesis uses $\Delta \Sigma$ since this was the first name the author learned and also in respect of the inventors.

Since the introduction of the delta-sigma modulator in 1962 several improvements have been proposed of this first order DSM. Most important was a higher-order modification suggested by Ritchie [5]. But as several researchers reported later the suggested higher-order modulators were not a trivial task. Before we can begin to talk about this issue some background information is needed. We will begin with a fundamental element, the quantizer.
2.2 DSM Basics

2.2.1 Quantization

Quantization can be defined as rounding or truncating a value to the nearest reference value. In other words it is the process of converting or mapping from continuous values of information to a finite number of discrete values as illustrated by figure 2.2. Quantization of amplitude and sampling in time are at the heart of all digital modulators.

The *Nyquist Shannon sampling theorem* [9] says that periodic sampling at more than twice the signal bandwidth need not introduce distortion. But quantization does. The primary objective in designing modulators is to limit this distortion, the quantization error or noise\(^1\).

We will see three ways to gain resolution.

1. Increasing the *number of bit-levels* - resolution in amplitude
2. Increasing the *oversampling rate* (OSR) - resolution in time
3. If the OSR is higher than two, the possibility for *noise shaping*

![Resolution in time and amplitude](image)

Figure 2.2: Resolution in time and amplitude for a 3-bit quantizer (left) versus a 4-bit quantizer with twice the sampling rate

Traditional Nyquist-converters use only method number 1 to gain resolution while the DSM gain its advantages utilizing method number 2 and 3. Though 1 is the most straightforward we will see that the combination of 2 and 3 has several advantages. Both number 1 and 2 are quite intuitive to understand. More accurate samples and more samples bring us closer to the original signal as illustrated by figure 2.2.

But to understand number 3 is not that straightforward. The concept builds on the fact that we know that the samples will be filtered by a reconstruction filter. Instead of trying to find the most precise samples we try to find the sample sequence which has an average value that corresponds best to the average value of the original signal.

To measure the effect of the different ways to gain resolution some theory is needed. Let’s begin by defining the *quantization noise*.

---

\(^1\) Quantization error refers to the error for each samples while the quantization noise refers to the effect of several errors.
2.2.2 Quantization Noise

The quantizer error is defined as the quantizer output minus the quantizer input

\[ e = q(x) - x \] (2.1)

where \( q \) is the quantizer function. Figure 2.3 represents a 3-bits quantizer function for a simple ramp input. In this example the quantizer input range is 2 and the level spacing is given by

\[ \Delta = \frac{\text{input range}}{2^b - 1} = \frac{2}{7} \]

where \( b \) is the number of bit-levels. The error is completely defined by the input. But if the input changes randomly or is sufficiently busy and is restricted inside the input range, the error has equal probability of lying anywhere in the range \( \pm \frac{\Delta}{2} \) as illustrated by figure 2.4. This noise can then be seen as white.

![Quantizer Input Output Error](image1)

**Figure 2.3:** The effect of a 3-bit quantizer for a simple ramp input

![Quantizer Input Output Error](image2)

**Figure 2.4:** The effect of a 3-bit quantizer for a random input

*White noise* [9] is a noise signal with a flat power spectral density. In other words, the signal’s power spectral density has equal power in any band, at any center frequency. It should therefore be a total random signal with no periodicity.

Signal-to-noise ratio (SNR) is defined as the power ratio between a signal \( S \), the meaningful information, and the background noise \( N \).

\[ \text{SNR} = \frac{S_{\text{pow}}}{N_{\text{pow}}} = \left( \frac{S_{\text{rms}}^2}{N_{\text{rms}}^2} \right) = \left( \frac{S_{\text{rms}}}{N_{\text{rms}}} \right)^2 \] (2.2)
CHAPTER 2. THE DELTA SIGMA MODULATOR

The subscript \( \text{pow} \) refers to the average power while \( \text{rms} \) refers to the root mean square value of the signal, or loosely, the average amplitude. The relation between \( \text{rms} \) and power in equation (2.2) is connected to how we compute power for electrical systems. The relation is given due to Ohm's Law [10] as

\[
\text{Power} = \frac{\text{Voltage}_{\text{rms}}^2}{\text{Resistance}}
\]

Since we are interested in the ratio between signal and noise we can omit the resistance since it will not affect the ratio in equation (2.2). It is also normal to set the resistance to unity so that the relation simply becomes \( P = V_{\text{rms}}^2 \).

Because signals often have a very wide dynamic range SNRs are usually expressed in terms of the logarithmic decibel scale. In decibels the SNR is 10 times the logarithm of the power ratio

\[
\text{SNR(dB)} = 10 \log_{10} \left( \frac{S_{\text{pow}}}{N_{\text{pow}}} \right)
\]

If the noise and signals are given in \( \text{rms amplitude} \), the SNR is given by

\[
\text{SNR(dB)} = 10 \log_{10} \left( \frac{S_{\text{rms}}}{N_{\text{rms}}} \right)^2 = 20 \log_{10} \left( \frac{S_{\text{rms}}}{N_{\text{rms}}} \right)
\]

Assuming a uniform distribution of input signal values, the quantization noise is a uniformly-distributed random signal with, loosely speaking, a peak-to-peak “amplitude\(^2\)” of one quantization level, making the maximal possible amplitude ratio approximately \( 2^b/1 \). The system’s maximum dynamic range can then loosely be determined to be

\[
\text{DR(dB)} = \text{SNR(dB)} = 20 \log_{10}(2^b) \approx b \cdot 6.02 \text{ dB}
\]

This is the origin of statements like “16-bit audio has a dynamic range of 96 dB.” Each extra quantization bit reduces the level of the quantization noise by roughly 6 dB. A system’s capable resolution is in literature often given by the number of bits using the relation of equation (2.5) to give a feeling of the level of resolution.

\(^2\)Amplitude is a term used normally about sinusoids; here we use it for the signals range.
2.2.3 Oversampling

Increasing the number of bit levels is not the only way to gain resolution when we quantize a signal. To show the gained effect of oversampling, some analysis is needed. Let us calculate the theoretical average noise power. For this we use root mean square (RMS) value which for a general signal is given by

\[ x_{rms} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} x_i^2} \]  

(2.6)

or for continuous functions

\[ f_{rms} = \sqrt{\frac{1}{T_2 - T_1} \int_{T_1}^{T_2} |f(t)|^2 \, dt} \]  

(2.7)

Since we assume a uniform noise distribution it’s enough to use one interval, from \(-\Delta/2\) to \(\Delta/2\), to calculate the average error power. Due to the relation in equation (2.2) between power and amplitude our theoretical average error power will become the square of the RMS value, i.e., the mean square value. If we treat the quantization error \(e\) as having equal probability of lying anywhere in the range \(\pm \Delta/2\) we can group the errors in increasing values and define it as a linear function \(f(x) = x\) as figure 2.5 illustrates. If the signal is sufficiently random figure 2.5b will converge to a straight line as the length increases.

![Random signal](image1.png)  

(a) Random signal

![Sorted random signal](image2.png)  

(b) Sorted random signal

Figure 2.5: Effect of sorting a random signal of length 1000

Regarding the noise as uniformly distributed gives us the opportunity to express the average noise as the function \(f(e) = e\). Using (2.7) we can then express the average noise power as

\[ N_{pow} = N_{rms}^2 = \frac{1}{\Delta} \int_{-\Delta/2}^{\Delta/2} e^2 \, de = \frac{\Delta^2}{12} \]  

(2.8)

If we now assume that all of its power folds into the positive frequency band \(0 \leq f \leq f_s/2\), and we consider the quantization noise as white, the spectral
CHAPTER 2. THE DELTA SIGMA MODULATOR

The density of the sampled noise is then given by

\[ N(f) = N_{rms} \sqrt{\frac{2}{f_s}} = N_{rms} \sqrt{\frac{2}{1/T}} = N_{rms} \sqrt{2T} \]  

Equation (2.9)

Here \( f_s \) is the sampling frequency and \( T = 1/f_s \) is the sampling period.

We can use equation (2.9) to see what happens when we introduce oversampling. Oversampling is the process of sampling a signal with a sampling frequency higher than twice the Nyquist frequency. The oversampling ratio (OSR) is defined as the ratio of the sampling frequency \( f_s \) to the Nyquist frequency \( 2f_{pb} \) and is given by

\[ \text{OSR} = \frac{f_s}{2f_{pb}} = \frac{1}{2f_{pb}T} \]  

Equation (2.10)

If we continue to assume white noise distribution over the spectrum we will gain SNR since the noise is spread in a wider spectrum, as illustrated in figure 2.6. The total noise is unchanged but we can now (low-pass) filter out more of the noise since there is more noise outside the pass-band, and hence gain SNR. To be specific the noise power that falls into the pass-band is now given by

\[ N_{pb,\text{pow}} = \int_{-f_{pb}}^{f_{pb}} N_{rms}(f) \, df = N_{rms}^2 2f_{pb}T = \frac{N_{rms}^2}{\text{OSR}} \Rightarrow N_{pb,\text{rms}} = \frac{N_{rms}}{\sqrt{\text{OSR}}} \]

Thus the oversampling reduces the in-band noise from ordinary Nyquist sampling by the square root of the oversampling ratio. i.e. a doubling of the sampling frequency decreases the in-band noise by \( 20 \log_{10} \sqrt{2} \approx 3\text{dB} \) or increases the resolution by half a bit (2.5). But there is more resolution to gain.

Since we have introduced oversampling, if proper filtering is implemented, a part of the spectrum will not influence the down-sampled result. If we could make the converters produce more noise in the part of the spectrum that is filtered away, and less inside the pass-band, it is possible to obtain a result with less noise than averaging the noise over the entire band of the converter as shown in figure 2.6b. This technique is called noise shaping.
2.2.4 Noise Shaping

To gain more resolution when oversampling we can use the delta-sigma ($\Delta \Sigma$) modulator (DSM). In short the DSM is a system that feeds back the error so that the average output signal tracks the average input. To give a first intuitive feel, a simple first order noise shaping structure is given in figure 2.7.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2_7.png}
\caption{First order error feedback only loop}
\end{figure}

With the equivalent circuit of the linear model we can do some analysis. Here we have replaced the quantizer with an addition of error $e$ which represents the error the quantizer introduces to the system. This analytical technique is called the linear model which we will discuss in more detail later in chapter 2.2.6.

By simple block calculation we see that the output of the modulator in figure 2.7 is

$$y_i = x_i + (e_i - e_{i-1})$$

(2.11)

Thus this circuit differentiates the quantization error, making the modulation error the first difference of the quantization error, while leaving the signal unchanged.

By again assuming sufficiently busy input, so that the error is uncorrelated with the signal, the spectral density of modulation noise, that is $n_i = e_i - e_{i-1}$, can be expressed by a discrete fourier-transform [9] as

$$N(f) = E(f) \left| 1 - e^{-j\omega T} \right|$$

$$f \in [0, 2\pi]$$

$$= E(f) \left| e^{-j\pi f} \frac{2j \sin(\frac{\omega T}{2})}{\pi} \right|$$

(2.12)

where $\omega = 2\pi f$. If we plot the noise shaped spectral density of equation (2.12) against that of equation (2.9) we see why we call it noise shaping, see figure 2.8.

The noise is shaped so that more lies in the upper parts of the frequency response and less inside the pass-band. The new total noise power in the signal band is

$$N_{pb pow} = \int_{f_p}^{f_{pb}} |N(f)|^2 df \approx f_{pb}^2 \pi^2 \frac{N_{rms}^2}{3} (2f_{pb}T)^3$$
Figure 2.8: Spectral density effect of first order noise shaping

And its rms value becomes

\[ N_{pb\text{rms}} \approx N_{rms} \frac{\pi}{\sqrt{3}} (2 f_0 T)^{3/2} \frac{2.10}{3} = N_{rms} \frac{\pi}{\sqrt{3}} (OSR)^{-3/2} \]

Each doubling of the oversampling ratio of this system reduces the noise by 9dB and provides 1.5 bits of increased resolution. Note that this improvement is dependent that the modulator’s output is low-pass filtered so that the out-of-band noise is eliminated.

Basic of noise shaping: shape the noise outside the pass-band. The total amount of noise is unchanged but it gives us the possibility to effectively filter out more of the unwanted noise and hence increase the resolution. A natural question arises, can we increase the amount of noise shaping?
2.2.5 Higher-Order Noise Shaping

The objective of using improved noise shaping is to reduce the total noise in the pass-band. A first order DSM like the one given in figure 2.7 subtracts the previous error. Higher order prediction should give better results than this first-order prediction. There are many ways to achieve second order noise shaping. Let's look at an example which is similar to the previous first order modulator. From figure 2.9 it is easy to see that the output becomes

\[ y_i = x_i + (e_i - 2e_{i-1} + e_{i-2}) \]

![Figure 2.9: Second order error feedback only loop](image)

Now the modulation noise is the second difference of the quantization error. The spectral density of the noise then, by a discrete Fourier-transform, becomes

\[ N(f) = E(f)(1 - e^{-j\omega T})^2 \]

And in a similar way as for the last case, we get for busy signals

\[ |N(f)| = 4N_{rms}\sqrt{2T}\sin^2\left(\frac{\omega T}{2}\right) \]

that gives the following rms noise in the signal band

\[ N_{phrms} \approx N_{rms}\frac{\pi^2}{\sqrt{3}}(2f_0 T)^{5/2} f_0^{5/2} \approx N_{rms}\frac{\pi^2}{\sqrt{3}}\text{OSR}^{-5/2} \]

This gives a noise attenuating of 15 dB for every doubling of the oversampling rate, providing approximately 2.5 extra bits of resolution.

Further increase of higher order predictions by adding more feedback loops can easily be done. And it can be shown [7, Chapter 1.2.3] that in theory the noise will fall \(3(2L - 1)\) decibels for every doubling of the sampling rate, providing \((L - \frac{1}{2})\) extra bits of resolution, where \(L\) is the amount of feedback loops. But as you will see in chapter 2.3 this is not possible in practice since increasing the order is not only beneficial for the DSM behavior.

Note that these derivations of properties of the delta sigma modulator depend on the assumption that the quantization noise is white. But this is far away from the general case. There are several problems concerning this. Especially pattern noise for low order modulators (< 3) and stability for high order modulators (> 2). Since this thesis mainly deals with higher order modulators the details concerning pattern noise will be omitted.

For further noise analysis see [7, Chapter 2 and 3].
2.2.6 The Linear Model (LM)

Although it is possible to gain some intuitive feel of DSM operation, the easiest way is by mathematical analysis of the linear model. We will now introduce a more general block diagram for delta-sigma modulators from [7, Chapter 4.2] in figure 2.10. This model is sufficiently general to encompass the majority of single-quantizer modulator topologies [11]. The modulator is split into two parts. One two input linear part and the nonlinear quantizer part. Again to do analysis we replace the quantizer by an addition of a noise signal $e$.

![Block diagram of a single-quantizer DSM](image)

Figure 2.10: General block diagram of a single-quantizer DSM

The linear block, which we call the loop-filter, has arbitrary transfer functions (see chapter 2.2.7) from its two inputs $U$ and $V$. This is done so that the diagram is usable for many topologies. The inputs transfer functions have been labeled for convenience as

$$L_0(z) = \frac{G(z)}{H(z)}$$
$$L_1(z) = \frac{[H(z) - 1]}{H(z)}$$

With these functions we can write the output of the linear block as the following sum of the inputs

$$Y(z) = L_0(z)U(z) + L_1(z)V(z)$$

By defining the error signal $e$ as $E(z) = Y(z) - V(z)$ we can rearrange the equation to give a formula for the output of the modulator in terms of its input and the error signal.

$$V(z) = G(z)U(z) + H(z)E(z)$$

Equation (2.13) captures the essence of noise shaping. The modulator output consists of independently filtered signal and noise components. This makes it possible with a proper loop-filter to, spectrally separate the input signal from the noise introduced by quantization, i.e. noise shaping. We will now rename $G$ and $H$ to respectively signal transfer function (STF) and noise transfer function (NTF).

$^a_z$ refers to the $z$-transform, see [9]
2.2.7 Noise- and Signal Transfer Functions (NTF and STF)

A transfer function [9] is a mathematical representation of the relation between the input and output of linear time-invariant systems. The transfer function TF is commonly used in the analysis of single-input single-output circuits. It is defined by

\[ TF = \frac{Output}{Input} \]

The introduced linear model gives us the opportunity to get an expression for the signal transfer function (STF) and the noise transfer function (NTF). The transfer function is only defined when there is one input and one output. Our linear model has two inputs, modulator input and noise addition. The trick is to set \( U(z) = 0 \) when computing the NTF and set \( E(z) = 0 \) when computing the STF. From equation (2.13) we then get

\[ STF(z) = \frac{V(z)}{U(z)} \quad \text{and} \quad NTF(z) = \frac{V(z)}{E(z)} \]

This gives us a method to analytically measure the theoretical noise attenuation a given loop-filter has. The noise attenuation in the pass-band can be calculated through the frequency response [9] of the NTF, i.e. \( |NTF(e^{j\omega})| \). You will see that the NTF is the heart of high-order DSM design.

The only restriction on the NTF, besides modulator stability which we will discuss later in chapter 2.3, is causality. This is that the NTF must satisfy the realizable condition\(^4\) \( ntf(0) = NTF(\infty) = 1 \). In practice this means that we must have at least one delay in the feedback loop [8, 4.1].

The NTF should act as a high pass filter for the ordinary delta-sigma modulators. Delta-sigma bandpass modulators, which also exist, have other NTF needs. The general desired ideal magnitude response of the NTF is

\[ |NTF(z)| = \begin{cases} 0 & \text{inside the passband} \\ 1 & \text{outside the passband} \end{cases} \]

This is the ideal need and is of course not possible to realize.

The simplest form for NTF is an all zero filter, i.e. a finite impulse response (FIR) filter [9].

\[ NTF(z) = a_0 + a_1 z^{-1} + \cdots + a_n z^{-n} \quad (2.14) \]

But to create the rectangular non-monotonic filters that will be discussed later, we need infinite impulse response (IIR) filters [9].

\[ NTF(z) = \frac{a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}}{b_0 + b_1 z^{-1} + \cdots + b_n z^{-n}} \quad (2.15) \]

Note that both forms satisfy the causality restriction when we let \( a_0 = 1 \) for (2.14) and \( a_0 = b_0 \) for (2.15).

We can also represent a filter transfer functions as a pole-zero plot in the z-plane [9]. The magnitude response of the filter can quickly be understood based on the location of the poles and zeros. The unit circle represents the frequency axes. In loose terms, the gain and attenuation are given by the poles and the

\(^4\)\( ntf() \) refers to the impulse response and \( NTF() \) to the z-transform, see [9]
zeros distance to the unit circle. The poles contribute to a gain increase and the zeros to an attenuation. In figure 2.11 the frequency response and the z-plane locations of the poles and zeros are given for two fourth-order non-monotonic NTFs, i.e. of the form of equation (2.15). Note how the frequency responses in figure 2.11 diverge from the first order monotonic modulator in figure 2.8.

![Frequency Response](image1)

![Z-plane Plot](image2)

Figure 2.11: Frequency response and Z-plane plot for two different fourth-order NTFs, (a) and (b) is a so-called Chebychev design while (c) and (d) is a Butterworth design. Note how the different zero-placements in the z-plane affect the frequency response.

When a good performing NTF is found it is just a matter of transforming the NTF to filter coefficients for the chosen topology or architecture. The DSM behavior is principally the same for one NTF even though topology and architecture widely diverse. It determines both the theoretical performance and the stability of the underlying modulator [11]. The only restriction is that the actual implementation can introduce coefficient errors that will naturally distort the result somewhat [7, Chapter 5 and 11].
2.2.8 Why Use Single-Bit DSMs?

As mentioned in the beginning of chapter 2 there are many implementation restrictions involved with ordinary Nyquist converters. And as we have seen there are several ways to achieve a high dynamic range. We can move or trade the resolution from the amplitude domain to the time domain. I.e. by increasing the OSR and utilizing noise shaping we can decrease the bit resolution while maintaining the total system resolution.

Surprisingly, with a sufficiently large OSR a two-level bit-stream can gain the same resolution as a Nyquist sampled $b$-bits signal. In fact for low frequency applications, there are more analog circuitry restrictions for Nyquist converters than for single-bit DSM converters. This is because they can utilize noise shaping and the two-level quantization.

Two level quantizations avoid the need for matched level spacing since they have inherently linear specifications. A displacement of the two levels only introduce a change of quantization range and dc offset, neither of which need to be critical and are therefore remarkably insensitive to analog component imperfections. The fundamental form is always preserved as illustrated in figure 2.12.

![Figure 2.12: Effect of a displaced level for two-level quantization](image)

The use of noise shaped two level oversampled converters leads to robust and cheap digital-to-analog and analog-to-digital converters (DAC/ADC). Instead of relying on precise electric behavior we spectrally separate the signal from the noise by shaping the spectrum of the quantization noise in such a way that very little noise lies in the signal band of interest. This takes extensive use of digital signal processing (DSP) which again takes advantage of VLSI (fast digital circuits). This is cheaper than precise analog circuits.

Because of the high OSR needs, the DSMs are best fitted for relative low-frequency signals (digital audio, telephony). This is because high frequency circuits have problematic implementation and efficiency also drops when the switching frequency becomes too high. But application emerges while faster technologies become available, such as video and radar. The use of high-order...
noise shaping can also alleviate somewhat the need for high OSR.

We will not go in detail about the difficulties of the hardware implementation of the DSMs. These are highly topology dependent and not of interest for a heuristic search to find generally good NTFs. But an important point should be noted. High-order single-bit delta sigma modulators have several advantages compared to other modulator strategies. The disadvantages mainly concern the difficult loop filter designs and stability problems [7, p. 166], which are exactly the aim for this thesis.
2.3 Stability of Higher Order Single-Bit DSM

The driving force of using high-order DSM is simply the gained NTF suppression of high-order loop-filters. I.e. the gained amount of noise shaping.

\[
\text{NTF}(z) = (1 - z^{-1})^n
\]  \hspace{1cm} (2.16)

We can gain exceedingly high SNR when \(n\) is large, even at low OSRs. But this level of performance is not achievable in practice because a modulator with an NTF in the form of equation (2.16) is not stable for single-bit modulators with orders greater than 2 [7]. By not stable we mean that the modulator exhibits large, although not necessarily unbounded, states and poor SNR compared with the ones predicted. This is illustrated by figure 2.13. Both modulators are monotonic and hence have the form of equation (2.16).

![Diagram of stability properties of second- and third-order monotone DSMs with the same input. The second order is stable while the third-order is unstable.](image)

Instability in practice is characterized by a low oscillation frequency, producing long alternating strings of 1’s and 0’s as can be seen in the unstable bit-stream output of the third order DSM in figure 2.13. This will be used later to detect instability in chapter 3.3.2. Another typical characterization is that the input to the quantizer overloads, i.e. the quantizer input, goes outside the quantizers input range.

It is appropriate here to note that multi-level DSM has more stable properties [8, Chapter 4.2.2].

For a number of years it was believed that modulators of order 3 and greater were automatically unstable. This was until Ritchie [5] and Lee [6] showed that high-order modulators could be stabilized. Since then high-order modulators have been a quest for many designers [12][13]. There are several papers that try to gain insight to the stability problem, but none of them provide a complete solution to it. The most common analysis is through the linear model.
2.3.1 Stability Analysis by the Linear Model

Simple digital signal processing analysis predicts that a causal linear system where the poles are inside the unit circle are so called **BIBO stable** - bounded input ensures bounded output [9]. But this simple rule vanishes for the DSM since the quantizer makes the system non-linear.

If we apply the linear model concepts we flaw the analysis but it will increase our understanding of the stability problem.

As seen in chapter 2.2.6 can we derive the following formula for the modulator output in terms of its input and error signal:

\[ Y(z) = \text{STF}(z)U(z) + \text{NTF}(z)E(z) \]

This makes us capable of focusing on the NTF by setting STF(z) to unity. Let us now model the quantizer as variable gain \( k_q \), the *quantizer gain* [19]. See figure 2.14.

The quantizer gain represents the slope of the quantizer. As the input of the quantizer varies, the gain \( k_q \) varies according to the input as

\[ k_q = \frac{q_{\text{output}}}{q_{\text{input}}} \]  

(2.17)

Thus the gain varies from \( \infty \) to zero when the input goes from zero to \( \pm \infty \) as seen in figure 2.15. As long as we are inside the quantizer input range, the gain will be \( \geq 1 \).

With this model, it can be shown [7, Chapter 4.2.1] that the modified NTF will now become

\[ \hat{\text{NTF}}(z) = \frac{\text{NTF}(z)}{k_q + \text{NTF}(z)(1 - k_q)} \]

I.e. the quantizer gain changes our loop-filter and gives us a different pole zero location. Unfortunately it can give rise to poles outside the unit circle and hence violate the bounded input bounded output (BIBO) stability rule [9]. This is known as Root Locus input bounded output (BIBO) stability rule [9]. This Since \( k_q \) is signal dependent we cannot fix this value and account for this effect. But the analysis helps us understand the causes for the problem of instability.
When the input to the quantizer is large, $k_q$ will fall. The effect of the pole locations is NTF and topology dependent. But as many conclude the general problem is that if the $k_q$ gets too small we will eventually experience poles outside the unit circle [7][8]. Hence root locus analysis indicates a reduced stability for decreasing values of $k_q$.

This is somewhat a difficult analysis. But intuitively it can be understood that, when the input to the quantizer is large, $k_q$ will fall and this in turn will result in even larger quantizer inputs. I.e. this will lead to runaway states and hence instability.

Intuitively this can also be understood by the following. If the input to the quantizer gets too large, the system will in turn get a problem correcting the error. The error will grow almost boundlessly. The feedback loop dominates the output and the SNR decreases.
2.3.2 Common Stability Criteria for Higher Order DSM

As we have seen in the previous chapter the root locus suggests that in order for a modulator to be stable, the input to the quantizer must not be allowed to become too large. Since the input, assuming STF(\(z\)) = 1, to the quantizer is given by

\[
Y(Z) = U(Z) + (\text{NTF}(z) - 1) E(z)
\]

a modulator's stability is dependent on both the maximum allowed input level and the feedback level gain of the NTF. This requirement leads to the conclusion that stability is dependent that the gain of a NTF(\(z\)) must not be too large and that there is a restriction on the maximum allowed input value.

Since an NTF of the form \(\text{NTF}(z) = (1 - z^{-1})^n\) has a peak gain of \(|\text{NTF}(-1)| = 2^n\), we understand why higher-order modulators with simple NTFs are unstable. In order to lower the out-of-band gain (OBG) we need NTFs with more rectangular filter specification. This is in literature called non-monotonic transfer functions [7, Chapter 1.2.3.6].

What we want is a transfer function that has an as high as possible noise attenuation in the pass-band while maintaining low enough OBG to sustain stability. We can use different filter techniques like Chebychev and Butterworth to design NTFs with these characterizations. A typical NTF frequency response is given in figure 2.16.

![Figure 2.16: Frequency response of a typical non-monotonic NTF design](image)

We will talk about the aggressiveness of the NTF. Loosely this defines how aggressively we have pushed the stable states to their limits. I.e. an aggressive NTF has high SNR and OBG values but a smaller maximal stable input (\(max_U\)) range. While a not so aggressive NTF has lower SNR performance but a larger \(max_U\) range. Figure 2.17 plots the \(max_U\) range and estimated SNR performance as a function of OBG values. The NTFs were produced by the Chebychev filter design technique and is of order five and has an OSR of 64.

To increase the aggressiveness of the NTF can be seen as increasing the SNR performance to a point where high OBG begins to make distorting effects. When we increase the OBG prior to instability we get higher SNR performance.
CHAPTER 2. THE DELTA SIGMA MODULATOR

But the price is an often undesirable decrease in the stable input range. This trade off between maximum allowed stable input range and the allowed OBG, limits the achievable SNR for the given order and OSR. This is well illustrated by figure 2.17. Chapter 2.4 will discuss different methods of producing these NTFs.

Note that root locus only indicates that high OBG will cause instability. It does not tell us how an optimal NTF should look like. For instance if we want to give an OBG bound, that is the maximal allowed OBG value for given parameters, how should we measure this OBG or which norm should we use for this bound? If we can have ripples in the OBG or if it should be as flat as possible, are questions where the scientific knowledge of today has yet not found sufficient answers [11].

There are several researchers that have suggested the use of ad hoc criteria for stability. These are mostly OBG bounds for the NTF [8, Chapter 4.2.1], but all are of the empirical\(^5\) ones and do not provide a complete solution to the stability problems. They are either too conservative or apply only to specific modulators [11]. Therefore one has to rely on simulation to verify stability and they can therefore not be used as strict guides towards optimal NTFs.

These bounds have a reasonable application as a rule-of-thumbs for stability for NTFs with flat OBG properties, such as those made from Butterworth and Chebychev filter techniques. But not necessarily for arbitrary NTFs as those our heuristic search will find.

There also exist no verification standards or methods of stability in literature. “Is this modulator stable?” is a prudent question. It is of course difficult to ensure stability for all signals in the signal-band since it is impossible to exhaustively simulate all the possible inputs that may be applied. But with no standardized way to measure stability and quality makes it difficult to compare different and arbitrary NTFs. Therefore we would like to propose the following characterization method for higher order DSM.

---

\(^5\) Based on experimental data, not on a theory.

Figure 2.17: Maximally allowed input range and SNR as a function of OBG. Note that the OBG increase is an effect of the increase in noise attenuation.
2.4 A Proposed Characterization and Verification of NTFs Stability and Quality

In literature NTF solutions often give little information about the stability characterization and performance. The solutions are often only specified by a single SNR value, an example fit plot for a sinusoidal input signal and perhaps the maximum allowed input amplitude ($max_U$).

The information given is not adequate to understand the whole performance of the DSM solution. It would be profitable with a standardized characterization and verification method which can determine the properties of a modulator in a wider extent and in this way approach a complete characterization of the NTF performance.

This is highly relevant when one wants to compare different arbitrary NTFs. Not only NTFs which are produced by common filter design techniques like Chebychev and Butterworth. Which is the best is a natural question. But this simple question has a problematic answer because of the lack of good standardized characterization methods.

2.4.1 The Proposed Method

To measure the performance of an NTF is a somewhat problematic quest since it's not possible to give a straightforward measurement of the stability and quality. The quantizer ruins for the analysis and it is not possible to test a modulator for every possible input sequence, since there are an infinite number of different combinations.

But if we assume that a given modulator input sinusoid, given by amplitude and frequency, reveals some of the modulator properties, we can systematically use this assumption to make a characterization of the system. We will see that in this way it is possible to converge to a stability characterization without the need of an infinite number of input signals. The following is our proposal.

Proposal When a higher order modulator is to be verified and characterized, the $max_U$ should be swept over the signal-band of interest. Not only will this plot tell us how the maximum stable input threshold varies over the signal-band, but its variance will also give a better understanding of the expected input range and hence help us to give a more strict maximum allowed input range. This global maximum allowed input range should then be used to make a similar sweep for the simulated SNR distortion (SINAD).

SINAD [15] was chosen as our noise floor estimate since it includes the distortion. Though higher-order modulators suffer less from distortion than low order modulation it is still our interest to minimize this noise element.

The characterization of two NTF designs from figure 2.11 on page 21 are given in figure 2.18. Both fourth-order designs have an OBG of approximately 1.5 and share the approximately same $max_u$ range of 0.6 over the pass-band. But the Chebychev design benefits from the spread zeros in the pass-band which give it a higher SNR performance (99.5dB versus 88dB). The 128-points characterization figures in this chapter, that is the $max_U$ and the SINAD sweeps, took approximately three minutes to compute on a normal desktop computer.
Figure 2.18: Characterization of the two NTF designs with an OSR of 64

Algorithm 1 Pseudo Code for the proposed method

```plaintext
freqs = [0 0.1 .. 0.9 1]  % Normalized pass-band frequencies
for every f in freqs
    maxu(f) = estimate_maxu(f)
end
maxu_global = min(maxu(freqs))  % min or more fancy variance estimate
snr_sweep = estimate_Snr(freqz, maxu_global)
```

The maximum stable input limit sweep reflects in our opinion more of the stability properties of a modulator. Several simulations have shown that if the modulator is stable for the sinusoidal $x = a \sin(f)$ it is not necessarily stable for amplitudes less than $a$. But this sweep and its curve variance can help us to give a reasonable sharper upper limit for the maximum allowed input range.
2.4.2 Some Properties of the Method

It’s important to note that these plots and their NTF characterizations in this Chapter is produced with the Chebychev\(^6\) filter design technique. So these characterization are only valid for Chebychev or all-flat filters as those Chebychev and Butterworth make.

The \(\max_U\) sweep reveals some stochastic properties of the modulators stability as you can see in figure 2.19. Note also that a more aggressive\(^7\) NTF has a higher variance in the \(\max_U\) sweep. It is difficult to determine a sharp upper limit for the \(\max_U\) of these aggressive NTFs without our proposed method.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{maxU_sweep.png}
\caption{The effect of high OBG for the \(\max_U\) sweep}
\end{figure}

Note that many designs have a ripple in the SINAD sweep from \(f = 0\) to \(\frac{1}{2}\). This occurs for designs with low OBG (< 1.6) and is an effect of the third harmonic component of the input signal, see figure 2.20. There is therefore a connection between the shape of the ripple and the frequency pass-band response to the NTF as illustrated by figure 2.21. Also note that all the \(\max_U\) sweep shapes coincide well with the \(\max_U\) characterization of the aggressiveness shapes in figure 2.19.

This method can also be used to automate the process of designing NTFs. A function can take an NTF as input and return the \(\max_U\) and the estimated SNR performance. A program can then use this function to automatically adjust an NTF to the designers needed specifications. This is partly what we will do when a search for good NTFs is presented later.

Our implementation of instability detection looks for long strings of 1’s or 0’s in the output bit-stream from the modulators. An important question is naturally how long the limit for stable strings should be. First the number 16 was chosen and seemed to work fine. But when NTFs with low OBG were tested it was found that higher limits must be chosen. This is illustrated by figure 2.22.

\(^6\)The NTFs where in fact produced with a procedure from “The Delta-Sigma Toolbox 7.0” [29]. This was chosen since it gives us an easy access to the OBG variable. But the performance is equivalent as those Chebychev makes.

\(^7\)An aggressive NTF is in literature referred to NTF with a high OBG
It was found that 32 is almost a magical limit. SNR estimates of bit-streams that had a tolerance of higher than 32 were often unstable. This can be due to the restriction of the float resolution in the simulation environment but this must be further studied. We have used a limit of 32 and it was found to be satisfying for our aims.
2.4.3 Stability Restrictions and SNR Performance of Chebyshev Filter Design Techniques

The proposed method from chapter 2.4.1 gives us the possibility to do the following studies of different Chebyshev designs. The OSR is set to 64 for both figures. The SNR increases when the OSR is increased but the $\max_U$ properties are almost the same.

Figure 2.23: $\max_U$ versus OBG and modulator order. Here the OSR was set to 64, but this parameter has little effect on this plot.
Figure 2.24: SNR versus OBG and modulator orders for an OSR of 64

Note that from figure 2.23 we must have an OBG higher than 1.1 to benefit from the high modulator orders. From figure 2.24 we see that with these OBG levels we already are below an \( \text{max}_U \) range of 0.9. We therefore conclude that Chebychev is not that well suited to make high-order NTFs with high \( \text{max}_U \) tolerance (> 0.85).
2.5 Today’s NTF Design Methods

Today there exist no optimal methods to design high-order modulators since, as we have seen in the previous chapter, there exist no analytical means which can guide us to an optimal NTF. The root locus analysis and others [8, Chapter 4.2] indicate that OBG must be bounded and that the modulators which are not stable for input $x$ have a higher chance of being stable for a down scaled signal $a x$ where $a < 1$. But the only way to verify stability and get a reasonable SNR measurement is by simulation.

Most of today’s methods consist of first creating an NTF, by some common filter design methods like Chebychev and Butterworth, and simulate its properties afterwards. Then post-adjust the NTF parameters if the stability or SNR performance are not satisfactory. This method is the most common and can be seen as a flow-chart procedure. A simple flow chart illustration of this method is given in figure 2.25. The steps are taken from [7, Chapter 4.4.1].

![Flowchart of NTF design procedure](image)  

**Figure 2.25:** Simple flowchart of a NTF design procedure

Another approach is to optimize more arbitrary NTFs while maintaining stability by restricting the design according to some (proposed) stability criteria. Both linear optimization techniques (Simplex) and Genetic Algorithms (GA) are tried out in this manner. There exists also an NTF design that focuses on the poles root locus movement inside the unit circle, but this stability comfort is an SNR performance trade-off. The methods can be grouped in the following categories.

1. Flow-chart procedure combined with filter design techniques [16][17].

2. Optimizing NTFs with different techniques while constraining the NTF by some stability criteria [18][19][20].

Both of the above methods have the potential to share one common disadvantage, the designs are restricted. When using method number 1 the NTFs are
restricted by how the filter techniques produce filters. And if we use number 2, the constraining must be conservative to ensure stability. A large part of the area outside these restricted areas are sure to be unstable, but can it contain optimal NTFs or other solutions of interests?

![Diagram](image)

Figure 2.26: The methods can only use a restricted NTF area
2.5.1 DSM Topologies and Chosen Simulation Setting

There are many ways to build delta sigma converters. The choice of loop topology and modulator implementation is DAC/ADC dependent. A Digital to Analog converter (DAC) uses digital logic. It can even be done in software. Analog to digital converter (ADC) is naturally highly dependent on analog circuitry. You can find several topologies in [7] for both ADCs and DACs.

It may be surprising that both ADC and DAC can make use of the same fundamental single-bit DSM. This is because both systems gain advantages using the inherently linear two-level or single-bit quantizer.

The aim for these single-bit modulators is to make a bit stream whose average value equals the average input value to the converters. For ADC this is often the best way to find a digital representation of the analog signal. In the DAC case the single-bit representation is preferable since a two-level switch (output stage) share in theory the same linear properties as two-level quantizations.

As chapter 2.2.7 noted the performance of an NTF is principally independent of the implementation. So once a high performance NTF is found for given parameters and restrictions, such as order, OSR and maxU range, this can be used for several DSM architectures.

To find theoretically good NTFs is therefore not highly dependent on the chosen implementation. This thesis could therefore choose the simple and computationally fast difference equation [7, Chapter 14.3] as the simulation model. This is a high level model that only determines the numerical result a NTF has. This is the fastest way to simulate the modulator behavior and determine properties as maximum stable input range. A post processing of the simulated output bit-stream, for instance a fast fourier transform (ftf), can then be used to estimate the SNR performance and other qualitative properties.

There are for single stage single-bit modulators two principally different architectures. If the STF is of unity or not, that is if STF(\(z\))=1 for all \(z\). Both models have been tried out and as simulation confirms, the performances between the two are rather similar. There are though special cases where the performances of the two are different. To be specific error feedback only (EFO) was chosen as the STF unity topology and the delta-sigma loop was chosen as the one without STF unity [7, Chapter 10.2.1-2]. These topologies are shown in figure 2.27.

The choice of difference equation as the simulation model also makes it easy to transform from NTF to loop filter coefficients, used by the difference equation, because of the close relation to the transfer function. Other topologies would have made messy calculation to find the loop filter coefficients. The chosen DSM model simplifies the work without losing generality.

![Figure 2.27: The Delta Sigma loop (left) and the error feedback only topology](image)

Figure 2.27: The Delta Sigma loop (left) and the error feedback only topology
2.5.2 Classical Chebychev NTF Design Example

Let us design an NTF with the help of Chebychev filter design. This is a Matlab example by making an inverse Chebychev type 2 of order 4 by the command cheby2. We choose an NTF with order 4, pass-band attenuation of 82.5 dB and normalized cut-off frequency of 1/64. This yields an NTF for a modulator with an OSR of 64 and with an OBG of approximately 1.5. This is the same design that is plotted figure 2.11a and b.

**Algorithm 2** NTF design example in Matlab

\[
[B,A] = \text{cheby2}(4, 82.5, 1/64, 'high'); \text{ % or butter/ellip} \\
B = B/B(1)); \text{ % causality} \\
[z,p] = \text{tf2zp}(B,A,1); \text{ % 3rd parameter also ensure causality}
\]

Now we have a transfer function to an NTF. But we don't know its exact properties. The describing function method of Aardalen and Paulus can be used to predict the SNR for various input amplitudes [14][29]. But to verify stability and measure the modulator behavior can only be done by a simulation. The easiest way is to transform the NTF loop-filter coefficients for the difference equation, which is simple and fast to simulate. Now we can simulate the NTF behavior, verify stability and measure the SNR for different inputs. By some extensive and systematic tests a stability- and quality-image of the NTF can be found as proposed by chapter 2.4.1. This is done in figure 2.15 and as we can see the design has a maximally allowed input amplitude range of about 0.6 and an expected SNR performance of approximately 99.5 dB.
2.6 Motivation for a Heuristic Search for NTFs

In computer science, a heuristic is a technique designed to solve a problem that ignores whether the solution can be proved to be correct or optimal, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem. Heuristics are intended to gain computational performance or conceptual simplicity potentially at the cost of accuracy or precision [21].

The fundamental difference from the method mentioned in 2.4 is that we will not use analysis to design our loop filters which determine the NTF. Today's computers are fast enough to determine, through the difference equation, if a design is stable or not for a given input signal in some fractions of a second. To measure the SNR performance some extra time is needed. We can let this quality measurement be our guide to good or optimal solutions. This is called the objective or fitness function and is the basis for heuristic search. If we take some care constraining the search, we will see that a sufficiently efficient search algorithm can be made for the purpose of finding arbitrary stable NTF. But why should we use heuristic search?

2.6.1 No Iron Clad Design Methodology is Available

Despite the widespread use of higher order DSMs there exists no satisfactory theory of their operation. This is a consequence of the fact that these systems are nonlinear, due to the presence of a discontinuous non-linearity - the quantizer.

The linear model explains the theory behind noise shaping but there exists no total understanding regarding the stability problems of high order DSM. The root locus and other analysis indicates that the OBG must be bounded but not to which extent and how [7][11].

Instead of relying on rule of thumbs we let a computer independently investigate the real world of non-linear DSM operations. And since our search will mainly be independent of any strict empiricism it could find surprising results not likely to accidentally result from a skilled human designer. This can again increase insight about the stability analysis and give rise for new science regarding the understanding of higher order DSM, or it can also to some degree verify today's empirical wisdom.

2.6.2 Flexibility of Parameters

Designing NTFs the traditional way requires some skills. For instance designing DSM that have a high maximum input range (>0.9) is not easy or even possible as seen in figure 2.23. Filter design techniques have restrictions for the purpose of the loop-filter design for the DSM.

You also avoid the tedious flow-scheme. The search automatically finds an aggressive NTF for your demands. If you for instance, from an implementation point of view, need 2 zeros in dc, you can fix these zeros and search for the best combination for your request.

We can also restrict or modify the search for a special set of coefficients or other constraints, that is coefficients of the power of two, spread of coefficients etc. The design process can in this way be more automatic and also more optimal for the given restrictions that the designer is given.
2.6.3 DSMs for Special Purpose

We can arbitrarily weight our fitness measurements so that they will find solutions that are well suited for special purposes. We can for instance specify in the fitness function the system's low-pass filter specifications, and in such a way find a solution that is well suited for this implementation. Chebychev has a maximum attenuated pass-band spectrum. Schreier has optimized zeros for maximum SNR. But how can we gain a modulator which is, in its total workings, optimal for our ears? A psycho acoustic model can be used for this purpose.

2.6.4 Signal Generators

There is always a need for high quality signal generators. If we for instance need the best possible sinus \( a \sin(\theta) \) for a given amplitude \( a \) and frequency \( \theta \), we could intensify the search for modulators after these means and not care about the performance for other signals. This could find DSMs which are specifically good for certain signals but which not necessarily hold for the whole pass-band. There exist today no specific methods for this.

2.6.5 Adaptive NTF Implementation

The ability to automatically find good NTFs for a given signal could also fit for an adaptive DSM solution that systematically adjusts the NTFs aggressiveness according to the running signal. This is highly relevant if one wants to utilize the maximum SNR performance restricted by a modulator order and OSR while still wanting a high \( max_U \) tolerance.

![Graph](image)

Figure 2.28: As our simulation shows, an adaptive NTF solution can give us both the SNR performance from an aggressive NTF and the \( max_U \) tolerance of none aggressive NTF.

High \( max_U \) tolerance comes on the expense of low SNR performance. Figure 2.28 is a comparison between two static NTF peak SNR plot and a simulated...
adapted peak SNR plot. Instead of choosing between a high SNR performance or a high $\max_u$ range, we can use an adaptive solution which offers both advantages and probably also a higher SNR performance which our simulation indicates.

The adaptive NTF curve in 2.28 represents the search achievements when we let the search adapt to sinusiodals. Maybe even more performance can be gained if it can adapt to the actual input signal.

### 2.6.6 Power Consumption

The search can also be weighted in such a way that a design well suited for low power consumption application is selected while ensuring an acceptable SNR performance. To accomplish this a restriction on the spread of coefficients can be used to lower the needed word lengths, and hence keep down the hardware size.

In fact every wanted criteria which is possible to measure can be used to influence the search.
Chapter 3

The Search

In computer science, a search algorithm, broadly speaking, is an algorithm that takes a problem as input and returns a solution to the problem, usually after evaluating a number of possible solutions. The set of all possible solutions to a problem is called the search space.

![Figure 3.1: A smooth and continuous well-defined search space](image)

Figure 3.1 is a simple example of a search space. If we want to maximize or minimize a function it is easily done by well-known optimization techniques when the function is as well defined\(^2\) as \(f\). But when the function is more arbitrary we do not have the same ability to use mathematical analysis to gain optimality. This is because they often are less smooth and that the computational cost of the point evaluations becomes too high.

A similar search space can be accomplished by using an arbitrary function

\[
f(\text{parameters}) = \text{performance}
\]

---

\(^1\) In literature the same concepts are often given different names depending on if the parameters to the function are real-valued or not. If the parameters is real-valued the searches for optimum are called optimization and the algorithm hill-climbing is called gradient descent. This can be confusing so this thesis will adopt names from both categories and only use one name for each concept.

\(^2\) \(f(x, y) = 3(1-x)^2\exp(-(x^2)-(y+1)^2) - 10(\frac{r}{4} - x^3y^5)\exp(-x^2-y^2) - \frac{1}{3}\exp(-(x+1)^2-y^2)\)
CHAPTER 3. THE SEARCH

We will adapt from Genetic Algorithms and call this our fitness function. For instance we can let the parameter be the different ways we can make a cup of coffee and let the performance be the quality of the taste. Here we certainly do not necessarily have the analysis needed to find the optimum solution for the parameters. The function can be seen as a black box or function. The parameters and the out-coming results are known but the interior manner of the operation is not completely known. This is illustrated in figure 3.2 where the three knobs let us adjust the parameters. We can measure the out-coming performance, but the interior operation is hidden from us.

This is somewhat the case for high-order DSM as discussed in chapter 2. Its behavior is not totally understood so that to find optimum NTF cannot be carried out through analysis. But the parameters to the DSM solution, the NTF, is easy to formulate and a quality measurement can also be carried out. Hence the fundamental foundation for exploring different solutions, i.e. the search space, is present. But how can we use this to find good NTF for a high order DSM?

Before a search can be started a suitable search space that matches our task must be defined. The determination of the search space is covered in chapter 3.2, 3.3 and 3.4. Chapter 3.5 will then cover how we systemize the use of the fitness function to accomplish good or nearly optimal solutions. Chapter 4 will then present some results of the search to find stable high order DSMs. This chapter assumes that the reader is familiar with some of the concepts dealing with higher-order DSM which was covered in chapter 2.

Figure 3.2: Black Box, different parameters gives different performance
3.1 Some Definitions

Before we can start to determine the search space and begin discussing search algorithm strategies some expressions are needed. These expressions may feel strange or unfamiliar in the beginning but their meanings will hopefully become clear when the end of this chapter is reached.

Initial Search Some algorithms need an initial search. This is for instance highly relevant for a search for stable high-order DSM. The initial search must find stable solutions that the optimization algorithm can go further on with. This is often just a random search for accepted candidates.

Hit(s) This refers to stable or accepted solutions which the initial search algorithm finds.

Hit-percentage This soon became a highly relevant term when a search for stable higher-order DSM was explored. It's a measure for how big the part of the search space, that contains accepted or stable solutions, is. If the hit-percentage is very low the initial search will use a long time finding accepted start candidates. Correctly carrying out constraining of the search space can increase the hit-percentage. But this is on the expense of the following dependency.

Dependency on Empirical Wisdom Since the DSM analysis is difficult most results are based on experiments, hence it is mostly based on empirical\textsuperscript{3} wisdom. We may want to let the algorithm figure out for itself where the good solutions are, independent of our prejudged mind. This increases the chance for finding surprising solutions not likely to accidentally result from a skilled human designer. It should therefore only be fed with a priori knowledge. But this is often a trade off for the hit-percentage.

Epistasis Interaction between parameters in which one parameter suppresses the action of another parameter. This is the case with the coding of the NTF. Poles and zeros interact and make it difficult to isolate the parameters behavior. It is therefore difficult or impossible to find the optimal location for one parameter at a time.

\textsuperscript{3}Empirical: Based on experimental data, not on a theory.
3.2 The Fitness Function - Describes the Search Space

The fitness function\(^4\) is simply a function that takes a problem's parameters, a candidate, and returns the measured performance - the fitness of the candidate.

The fitness function is the basis of the search and defines the search space, hence it defines every solution to every possible combination of parameters. The function tells the search algorithm how good a solution is and should therefore represent the true quality of the modulator. The simplest way to then find the optimal solution is to iterate through the whole search space. This is known as the *Brute-force* search [22]. But this is often of little practical interest since the time will grow exponentially as the search space dimension increases. The problem concerning the size can be overcome to some extent through different strategies that we will call search algorithms. There are several other considerations to take:

- Are there different ways to code the parameters?
- Are there different ways to measure the performance?
- How can we constrain the search space?
- How can these different implementations of the fitness function affect the search space and hence affect the quality of our search?

The goal with this chapter is to explain how the fitness function determines the search space. Let us start with the coding of the input.

---

\(^4\)The name fitness function is adapted from the field of Genetic Algorithm.
3.2.1 The Coding of the Input to the Fitness Function

The input to the fitness function should be a set of parameters that defines a proposed solution to the task. It should be able to represent all the different ways we can solve the problem. This is, in the case of our high order DSM, the NTF with an appropriate OSR as described in chapter 2.2.7. The NTF determines the loop filter coefficients that again determine the resulting DSM behavior.

The NTF can be expressed or coded in several ways, for instance by zero-pole locations or by a transfer function. The coding is highly relevant to the search since it directly affects the search space, especially the dimension.

An interesting matter is if it can affect the search space in such a way that it makes it more or less convex, differentiable, continuous or affect the search space in any other way. This is highly relevant for the performance of the search. The analysis is beyond this thesis’s scope but we will try to focus on smooth search spaces later on in chapter 3.4.3. In figure 3.3 a comparison between a chaotic and a smooth search space can be seen. It’s clear that the first has a characterization which makes it a less preferable search space.

![Chaotic characterization](image1)

![Smooth characterization](image2)

Figure 3.3: Chaotic versus smooth search spaces

It seemed natural to describe the NTF as zero-pole locations in the z-plane (see figure 2.11) because of its close relationship with the understanding of the frequency response. This has many advantages.

- It’s easy to keep the NTF poles inside the unit circle to sustain BIBO stability and similarly easy to constrain the zero locations on the unit circle in the pass band.
- It’s easier to connect the parameters to the given NTF performance.
- If we had to start with loop-filter coefficients we would have been dependent on messy calculation to control the above.

Further we let the coding of the NTF be a question of constraining the search space. The coding is therefore further determined later in chapter 3.3 that deals with the constraining.
The fitness function is now determined to be

\[ f(NTF_{our}) = \text{performance} \]

Let us now consider how we should measure the output of the fitness function.

### 3.2.2 Fitness Measurement - The output

As the case with our simple coffee example it is easier to describe the input to the function than the output. How should we measure the total quality of a DSM solution? This is not an easy quest since it can be looked upon in different ways as described in chapter 2. It may even be a subjective task. But here also lies one of the advantages of this design approach. We can weight our fitness measurement such as to find a solution well suited for our needs as noted in chapter 2.6.3.

But the choice of the fitness measurement will also determine the characterization of the search space that can heavily affect the performance of the search. Smooth and continuous search spaces are advantageous for most search algorithms while more random and not continuous spaces makes it harder for a practical search, especially local search. Some care should therefore be taken when the fitness measurement method is to be determined.

The combinations of fitness measurements are almost endless and should be determined by the wanted performance. This will be determined later in chapter 3.4. The fitness function is now loosely defined to be

\[ f(NTF_{our}) = \text{desired quality measurement} \]

We now have a defined search space. But it has an infinite size. Before a practical search can be executed it must be constrained to a finite area. Smaller areas are better for our search as long we do not constrain out interesting or optimum areas.
3.3 Constraining the Search Space

A problem with Heuristic search is the exponential growth of complexity or space dimensions with the order. DSM also has a large amount of unstable areas that can be a big problem for the search if not handled with care. This makes it noncontinuous and noisy.

As we will see we can both constrain by the coding of the input function, that is how we code the NTF, and constrain internally in the fitness function. This is respectively to restrict the parameters area and restrict internally inside this area.

Note that one should be careful when constraining with empirical wisdom. It may be sensible but the danger of constraining away optimum solutions exists.

3.3.1 Constraining by the Fitness Function Coding

We are now going to make external constraints. I.e. constrain our infinite space to a finite constrained space. For our simple fitness function $f$ in figure 3.1 this would be to constrain $x, y \in [\text{start, stop}]$. This is directly to decrease the search space area to a finite one. But before we can decide a frame we must decide how to code the zeros and poles of the NTF.

A disadvantage with zero-pole coding of the NTF is that it contains redundant information. We know that zero pole locations must be symmetric to express a real transfer function and hence a realizable filter [9]. So instead of specifying every zero and pole let us only specify the needed information. That is the location of the pairs of poles and zeros. This will exclude redundant parameters from the search space. When one of two in a pair is specified it is simple to determine the other by mirroring it down or up around the real-axis. If the NTF-order is odd we know, from symmetric needs, that one pole and one zero must lay on the real-axis. So here only one coefficient is needed to specify the pole-zero placement since the location is fixed on the real-axis. This symmetry is illustrated in figure 3.4.

![Figure 3.4: The need for symmetry in the z-plane](image)

Another way to constrain is by fixing parameters to specified values. For instance Schreiers optimized zeros [7, Table 4.1] was tried out. Now only the pole-pairs need to be specified since the zeros are fixed. But experiments show
that constraining the zeros on the unit circle inside the pass band works better for our search. This gives more freedom of the zeros placement while ensuring high pass band attenuation. In fact it also increased the hit percentage.

Note again that fixing or constraining the zeros placements in this manner makes use of empirical wisdom and can be seen as a weakness for the independency. If other zero placements are optimal this coding will ignore these solutions and optimum can never be found.

Thus the coding of the zeros is now simply one coefficient $z_i \in [0, 1]$ which represents the zeros placement in normalized frequency, and if the order is odd there is one zero that does not need any specification since it must lie in dc$^5$.

The coding of the pole pairs was further determined to be in radian format

$$p_i = r_i e^{\pm \theta_i}$$

where $r \in [0, 1]$ and $\theta \in [0, \pi]$. This makes it easy to ensure poles inside the unit circle since this is only to ensure that $r < 1$. $\theta \in [0, \pi/2]$ can be chosen if we further want to restrict the poles inside first and forth quadrant. The coding is illustrated in figure 3.5. Remember that the figure and the coding only specify the pole-zeros on and above the real axis. Those above the real axis are mirrored down to yield a symmetric plot and hence a real transfer function.

![Figure 3.5: The coding of pole and zero placement](image)

To summarize our fitness function coding is now determined to be pole-pairs and zero-pairs which represent an NTF, hence NTF$ = ( $pole-pairs, zero-pairs$)$ where the pairs are defined as

$$p_i = r_i e^{\pm j \theta_i}, \quad z_i = e^{\pm j \frac{\lambda_i}{2n}} \quad \text{for } i = 1, 2, \ldots, \frac{n}{2} \text{ if } n \text{ is even}$$

Here $\lambda_i \in [0, 1]$ determines the zeros placement in the pass-band. If $n$ is odd, due to the needed symmetry of a real filter, the last $n$-th pole and zero is respectively fixed to the real axis and dc; hence the last pole and zero is specified to be $p_n = r_i$ and $z_n = 1$. For even orders $\frac{3n}{2}$ coefficients are needed and if $n$ is odd $\frac{3(n-1)}{2} + 1$ coefficients are needed to determine a NTF.

$^5$dc refers to zero frequency
CHAPTER 3. THE SEARCH

Note that the search is now constrained by the coding. We don’t look for NTF with poles outside the unit circle, and zeros not on the pass band unit-circle. This gives us a higher percentage of good and stable solutions in the search space at the expense of lost independence of empirical wisdom.

If the hit-percentage is still not sufficiently high for the initial or global search a further coefficients constraining can be accomplished by constraining the poles closer to dc, to lower the OBG as the linear model analysis tells us. This was done for very high orders (> 6).

Our search space is now a constrained multidimensional (filled) box. But further constraining inside the box is possible.
3.3.2 Local Constraining Inside the Fitness Function

As earlier mentioned the big areas of unstable behavior, in the search space of higher order DSM, are problematic for the search algorithms. Our external constraining give a better chance of finding stable candidates, but we can still have big unstable areas inside the search space that can be a problem for the given search algorithm. This is because unstable fitness makes little sense for the algorithms, especially local search algorithms. A total fit SNR measurement of a candidate NTF can take up to several seconds dependent on complexity of the measurement. We can end up spending a lot of time computing the fitness of unstable solutions. Much work can therefore be saved if we only compute the fitness measurement when we know or are pretty sure that the NTF is stable. This is difficult to constrain by outer limits since stability is determined by the poles and zeros total workings. For instance the root locus (see chapter 2.3) indicates that high OBG is the cause of instability. The total OBG cannot be determined by a single pole location but is a sum of all pole-zero placements. The coding is so called epistasis.

Our goal is not to try to guarantee stability since this would imply conservative bounds not allowing optimal solutions and giving us the same NTF restrictions as mentioned in chapter 2.5. We only want to increase the hit percentage and to avoid using too much computational effort on NTF that has very high probability of being unstable. We will show two methods to constrain inside in the fitness function, analysis by the Linear Model (LM), and by a fast stability test with the use of a computational fast simulation model, the difference equation.

1. LM analysis

If linear model analysis predict a very high probability of instability we can drop these measurements and set the returned fitness to the minimum fitness value, for instance 0 or $-\infty$. In practice, this is done by a frequency response analysis of the NTF which determines the maximal OBG (or by other norms). If this is higher than a given bound we don’t care to compute the SNR. But if this bound is too strict we can again risk to restrict the NTF too much so care must be taken. The bounds were typically set to a much higher value than typical OBG gains (OBG < 4 instead of < 1.5).

In the same manner we can, through an LM analysis, measure the predicted NTF noise attenuation. If this is too low we don’t care to measure the SNR since we already know this is not a good NTF solution.

2. Simulation

Another way to detect instability is through the difference equation. If instability or overloading occur, a typical characterization is that we get a long string of 1’s or 0’s in the digital bit-stream [7, Chapter 4.6], see figure 2.13. The difference equation is computationally fast and can be specialized to detect such long strings. A threshold of 32 must be used to tolerate low OBG designs. This method is much faster than an fit analysis and can save us much work regarding the search algorithms computational cost and effectiveness. The long-string checker needs a long duration to detect instability [11]. But if a long string is detected in the beginning of the simulation we don’t need to go on, so this can be an effective way to
Figure 3.6: We can constrain away NTF solutions with too high OBG and too low noise attenuation

detect instability.

Constraining in this way does not ruin for independency of empirical wisdom since it just tries a candidate NTF without caring about earlier experience and LM analysis.

The proposed methods can either be used together or a subset can be chosen. For instance if we want to try without the use of any empirical wisdom we can drop number 1. But LM analysis is faster and according to our experience loose OBG bounds and lower SNR limits do not eliminate NTF solutions of interest from the search space. In fact our search did lead to solutions well inside these bounds. If search algorithms begin to explore areas close to the bounds this surely indicates that the bounds should be loosened.

3.3.3 Summarize

Now we have a fully constrained fitness function. The initial search will only search inside the given outer limits. The fitness measurements takes a short time if the NTF is not accepted by the internal constraining. If the NTF is accepted the function carries out a quality measurement.

\[ f(\text{NTF}_{\text{ext}}) = \begin{cases} \text{quality measurement} & \text{if constraining accepts NTF} \\ 0 & \text{or} \ -\infty & \text{if NTF is not accepted} \end{cases} \]

Note that constraining the search space is mostly a concern for the initial and global search. The local search part of an algorithm may explore outside the external constraining area and is also fitter to explore spaces with low hit percentage and large unstable areas since its movements are restricted inside stable areas.
3.4 Determination of the Fitness Measurement

We have not yet discussed how we should measure the fitness for a candidate NTF. As noted earlier the fitness measurements should represent the quality we are searching for. This, as we have noted in chapter 2, is not a trivial task. But we will use some of the concepts from our proposed characterization method in chapter 2.4.1. One of the considerations we must take is the dilemma of stationary points.

3.4.1 Some Consideration of the Fitness Measurement

Since we are dependent on an input signal to simulate the behavior of the DSM, the quality measurement will be a stationary point given by this signal. The input signal will therefore directly influence the search landscape. For instance if we let the fitness function measure the capability to modulate a sinusoidal with frequency \( f \) and amplitude \( a \), the fitness function will guide the search toward a good solution for this sinusoidal. This NTF is not necessarily good for other frequencies or amplitudes in the pass-band and the given amplitude range. Or even worse, it is not necessarily stable. If we want a global quality throughout the pass-band the fitness function must be implemented to accommodate the desired specifications.

The length is also of importance. A short input signal will not guarantee stable behavior [11]. Therefore a large number of input steps and input values are necessary to ensure the stability of interest. Long signals may also make smoother search spaces, because of the increased number of samples to the fit measurements.

3.4.2 Using an SNR Measurement as the Fitness

A natural thought is to use an SNR estimate as the fitness for a candidate NTF. But as we have seen in chapter 2.4.1 this is not a trivial task. A high amplitude close to the amplitude threshold is needed to give a peak SNR performance of the modulators NTF solutions. How should we choose this amplitude? Two different strategies were considered.

1. **Fixed amplitude**
   This strategy is the most straight forward and consists of feeding the modulator simulator with one or several sinusoids with a fixed amplitude. This is an ideal method if a specific amplitude threshold design is wanted. The search will then in its nature automatically run the search toward an NTF that has this amplitude as its peak SNR amplitude.

2. **Dynamic amplitude**
   If just maximum SNR performance is wanted without any specific amplitude threshold needs, a variant of the method of chapter 2.4.1 can be used to guide the search towards modulators with maximum SNR performance. The method finds the candidates maximum threshold level \( \max T \), hence the NTFs peak SNR amplitude, and uses this level to measure its simulated SNR performance. The computational cost becomes higher since it first must determine the threshold level. You will see that this approach
has a noisy nature and is therefore not that suited for a local search. But it worked to some degree.

Another thought was to use the estimated pass-band attenuation as the fitness, and in this way not be dependent on an input amplitude to measure the fitness or how good modulators behave. This turns our search towards NTF with maximum pass-band attenuation. The concept is therefore not dependent on an input amplitude. But in practice it still is, since the noise floor is heavily influenced by the input signal to the modulator. It is still therefore a stationary point. But the most serious drawback is that it does not care about the input signal destiny, so this strategy failed.

If we want to use arbitrary signals as inputs we can not use fft to measure the SNR, since this is dependent on sinusidals to determine the power-ratio between the input signal and the noise floor. Instead we can use a least square comparison between the down-filtered output bit-stream and the original signal to measure the noise introduced by the modulator. This could be used for instance in an adaptive DSM implementation which was presented in chapter 2.6.5.
3.4.3 Smooth Search Spaces

An important specification for local search is smooth search space surfaces. How can we gain smooth surfaces? Is the length of the simulation duration of importance? As mentioned before long input signals could give a more accurate SNR estimate and hence give smoother surfaces. But the DSM landscape is sure to be noisy because of the chaotic and stationary properties. So maybe the average of several sinusoids should be used to make a smoother search space. This will also ensure a more global performance if the sinusoids are spread in the pass-band. Let’s try out some different concepts and compare their effect by some search space plots.

It’s not possible to plot a higher order DSM surface in three dimensions since it demands several parameters. But to reveal some of the properties we can choose to adjust two parameters and let the rest be fixed. This is the aim for this sub-chapter.

To determine a third-order NTF we need \( \frac{3(3-1)}{2} + 1 = 4 \) parameters as explained in chapter 3.3.1. We can constrain this down to two parameters if we fix all the zeros to 1 and the odd pole on the real axis. This poles placement was determined to be at 0.5. Now remains one pole-pair placement to determine which requires two parameters, the poles radian length and the angle.

Since we have constrained the coding down to two parameters a three dimensional plot can be carried out. We can now iterate through the search space for different placements of these two symmetric poles. It will not give a robust picture of the general surfaces but should indicate and reveal some of the smoothness properties. This can again indicate how we should choose the computational effort to gain smooth search spaces. Let us first plot the surface of the linear models estimated noise attenuation, i.e. the theoretical SNR of the frequency response of an NTF. This surface has exactly the smooth characterization we want.

![Figure 3.7: Search space surfaces for the noise attenuation of the NTFs where we have constrained away NTF which is not stable for input amplitudes of 0.4](image)

The following concepts were plotted.
3.4.3.1 Length of Input Signal or Amount of fft-points

Longer signals should give more samples to the SNR estimate and hence give a more accurate SNR estimate. Longer signals can also give a more accurate response of the properties of a modulator. As we can see in figure 3.8 there are smoother properties for the one with longer input signals. But the difference is not extreme, which indicates that the extra computational cost does not pay off. The computational cost is given in table 3.1.

![Graphs showing search space surfaces for different signal lengths](image)

Figure 3.8: Search space surfaces for input signal length of $2^{14}$, $2^{16}$, $2^{18}$ and $2^{20}$ samples.

<table>
<thead>
<tr>
<th>Length</th>
<th>$2^{14}$</th>
<th>$2^{16}$</th>
<th>$2^{18}$</th>
<th>$2^{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time in seconds</td>
<td>0.05</td>
<td>0.2</td>
<td>0.8</td>
<td>3.2</td>
</tr>
</tbody>
</table>

Table 3.1: Estimated computational cost of the fitness estimate for different lengths of the input signal. The estimates were done on a normal desktop computer.
3.4.3.2 Mean of Several Different Input Signals

An idea was to choose several frequencies in the pass-band and set the estimated SNR as the average value of these separate measurements. Single frequencies have its noises which give them a stationary point characterization as can be seen in figure 3.8. The mean of several sinusoidal fit measurement should therefore give smoother surfaces and give the surface a less stationary impression. As seen in figure 3.9 this concept has a positive effect on the smoothness properties of the surfaces at the expense of the computational cost.

![Graphs](image)

Figure 3.9: Search space surfaces when the fitness function uses two, five, ten or 25 separate SNR estimates with length $2^{16}$. 
CHAPTER 3. THE SEARCH

3.4.3.3 Concept on Dynamic Amplitude

To see the effect of using the concept of dynamic amplitude from chapter 3.4.2 as the fitness function, we have plotted the surface of this concept. This has a more noisy surface and is therefore not that suited for a local search as we can see in figure 3.10. If we increase the resolution of the \( \text{max}_U \) determination we gain a smoother surface. But there are still noisy areas. Note also that this concept has a larger stable part. This is because the former uses a fixed amplitude of 0.4 while this concept tries to find the stable input level.

![Figure 3.10: Search space surfaces when concept of dynamical amplitude is used. Both are ten-points with length \( 2^{16} \) but the latter has a higher \( \text{max}_U \) resolution](image)

3.4.3.4 Smooth Search Space Conclusions

We therefore conclude that the input length should be between \( 2^{16} \) and \( 2^{18} \) and the average of several signals make the search space smoother. From the plots it seems like a better idea to spend \( 10 \cdot 0.2 = 2 \) seconds on a ten-point mean SNR estimate of length \( 2^{16} \), than using 3.2 seconds on a single signal of length \( 2^{20} \). Its effects on the search are difficult to foresee but they indicate that several signals should be used. A choice of around 10 different signals of length \( 2^{16} \) seems sensible.
3.5 The Heuristic Search Algorithms

We now have, through our fitness function, a fully determined search space that we can begin to explore. But how and what is an effective way to find good or near optimum solutions? The simplest and most intuitive method would be a Brute-force strategy that consists of systematically enumerating all possible candidates for the solution [22]. I.e. search through the whole search space. For a real valued function we must first quantize the parameters but it is the same concept. This strategy is of little practical interest because of the exponential increase of size with the dimensions. In the case of non-linear functions with no analytical means to find optimum solutions, and without going through the whole search space, we are dependent on some kind of heuristics.

3.5.1 Basic of Heuristic Search Algorithms

The basic idea of heuristic search is that, rather than trying all possible solutions, we try to focus on a systematic approach that can find good solutions in minimum needed calculated points, i.e. minimum needed computational cost. Of course, we generally can’t be sure that good or near optimal solutions will be found. But we might be able to have a good guess. Heuristics are used to help us make that guess [21].

These methods are not dependent on a good well defined description of the fitness function and is maybe the best search which can be carried out as long as this does not exists. We, except theorists, don’t care if it is optimal as long as it wins or does our job well enough.

The science is new and nearly all general results are of the empirical ones. This makes the science a bit fuzzy or indeterminable but as several applications have shown these techniques can do a sensible job and outperform other solutions that exist [27][28].

There are several methods used today. This thesis will not try to uncover the art of Heuristic Search but try to express this work experience with the task. It is difficult to say anything in general, because of the close relationship between the search and its search space.

Note that we always refer to search for the maximum value. If global minimum is the goal, this is the same problem. The only difference is a change of a sign.

$$\min f(x) = \max(-f(x))$$

The reader may want to repeat the definitions given in chapter 3.1 before reading further. The definitions are highly relevant terms when we start the discussion of different search strategies to find stable high-order DSM.

First we will present three different fundamental search algorithms. Then we will present four hybrids of these main methods that try to combine the different strengths of the strategies. A natural combination is to combine a global search strategy with one that has a more global perspective. The following methods were tried out. We will begin with the most simple strategy.
3.5.2 Random Search (RS)

Instead of iterating through the whole search space can we let a random generator repetitively pick candidate NTFs for us and measure their fitness. While we let the search algorithm run, it will uncover more and more of the search space if the random generator is sufficiently random.

If we know or guess some of the expected statistical properties of the search space, we can systematically keep track of the estimated fitness of the candidates and make an appropriate stop condition.

This method can find better solutions faster than the iteration since its view is far more effectively global than the iterative method. But if the good solutions are a very small percentage of the search space, it will be outperformed by more systematical approaches that concentrate their searches around the good parts, or in other words try to figure out the search space.

But the simplicity of random search makes it a good benchmark for heuristic searches. Its simplicity also makes it a preferable initial search method for other algorithms.

![Random Search Diagram](image)

Figure 3.11: Illustration of Random Search. A random generator delivers random parameters. If the generator is sufficiently random the near optimal solutions will eventually be found.

<table>
<thead>
<tr>
<th>Pros</th>
<th>Simplicity and its global view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cons</td>
<td>An unintelligent strategy since it uses no effort on trying to figure out the search space, not effective for low-hit percentages.</td>
</tr>
</tbody>
</table>
3.5.3 Hill-Climbing (HC)

Hill-climbing\(^\text{a}\) is a local search technique where the name describes the principal function [23]. The strategy is to see the search space as a multi-dimensional hill and try to climb to the highest peak. There is however an essential difference between the hill climber and the algorithm. While the human hill climbers can use their sight and other senses the algorithm is blind. The algorithm must compute points around it to find out which way to climb. It’s common to climb the steepest direction but there also exist other strategies to determine direction or route [24].

In general an optimal way is difficult to find since this deeply depends on the search space. The DSM space is quite random and noisy so an optimal way is maybe not possible to accomplish. A more quantitative approach, i.e. to make several different climbs, seems to do a more robust job since it will uncover or explore more of the search space. This will also transform it to a more global search.

One of the advantages with the hill climb algorithm, or local search in general, is that it is better fitted for search spaces with low hit-percentages since it concentrates its effort inside the stable or accepted areas. It is not allowed to go outside the legal areas.

The strategy does not describe the start point. This could either be chosen in a systematical way or in a more random fashion. For more theory about hill climbing see [23].

![Diagram of Hill Climb](image)

Figure 3.12: Illustration of Hill Climb. Climbs from the start point to its local maximum point.

**Pros**
- Simple and robust concept and good for low hit-percentages.

**Cons**
- Finds only local maximum and is dependent on smooth search spaces.

\(^\text{a}\)Known as gradient ascent or descent for continuous spaces
3.5.4 Genetic Algorithm (GA)

Genetic Algorithm is a search strategy that tries to mimic nature’s natural selection method of evolution - *survival of the fittest*. In loose terms the algorithmic strategy is to take a population of different candidates and makes a new population based on the fitness of the old population. Candidates with higher fitness have higher change to influence the next generation. The production of the next generation can be based on both mutation and mating or combinations of the old population. In this way the populations should converge to good or near optimal solutions. Its history and operation is covered in [25].

An ordinary implementation of GA is to code the candidates as a gene-like bit coding. This is to randomize the search space to avoid getting stuck in the local maximum. But also real-coded GA is tried out in different papers [26].

A real-coded GA was tried out but it did not seem to be effective for our task. In the authors’ opinion GA is a good global search method but it is dependent that the genes, that is the candidates parameters, are not epistasis, to let the mating be effective. The coding must somehow do a sensible job for the search. For our search only the mutations gave better out-spring so it worked more like a random local search method. Hill Climbing therefore seems to be a better solution for this since it is a specialized local search technique.

![Figure 3.13: Illustration of Genetic Algorithm. A population of different genes is the base for the next generation.](image)

**Pros**
- Global view and robust for many applications

**Cons**
- Bad local view and often outrun by specialized strategies, dependent that parameters must not be completely epistasis?
3.5.5 Hybrid GA

A hybrid approach of GA and HC was tried out, a so-called hybridGA [26]. This tries to utilize both the global view of GA and the local search power of hill climbing.

The GA is first set to explore the search space in its global point of view. After some generation, or when the GA is not longer gaining fitness, the Hill Climber takes over and climbs the solutions that the GA found.

As mentioned the GA was not that effective in the DSM search space, so this method was rejected. Instead of letting GA deliver start points to a Hill Climber we can use a simpler algorithm.

Pros Benefits from both GAs global view and HC local view
Cons Dependent on appropriate search spaces for both GA and HC

Figure 3.14: Illustration of hybridGA. The last generation from GA is fed to the HC algorithm.
3.5.6 Random Restart Hill Climb (rrHC)

As noted in the chapter 3.5.3 we can turn the Hill Climb algorithm into a more global search method by a quantitative approach. Random Restart Hill Climb is exactly this. A random generator repeatedly picks starting points and lets a Hill Climber algorithm climb these point's local maximum potential.

The algorithm can be seen as a hybrid of random search and Hill Climb. The strategy combines the global perspective of random search with the Hill Climbs local search ability. If the start points are sufficiently random eventually the search will have explored a large part of the search space. Its quantitative nature makes it robust to some extent.

![Graph showing random restart hill climb]

Figure 3.15: Illustration of Random Restart Hill Climb. Several random start points are climbed

Pros  
A simple hybrid between local and global search strategies

Cons  
The random search will suffer from low hit-percentages and HC are dependent on smooth search spaces
3.5.7 Best of Random Restart Hill Climb (brrHC)

If the climbs computational cost is much higher than finding stable candidates it can be advantageous not to blindly climb every start point but instead use some time finding several hits and only climb the best.

*Best of Random Restart Hill Climb* climbs only a subset of several hits that the initial search finds. The choosing of the hits or points to climb can be chosen from their fitness, difference or other relevant information. If the climbs have a high computational cost compared to the cost of finding hits, for instance high dimensions and high computational cost of the fitness measurement, this strategy can ensure that only sensible hits are climbed. But if the hit-percentage is small the strategy will use much time finding the large amount of hits.

The elitist idea has no guarantee, since a bad start can turn out to have a higher potential than a good start candidate. But in a statistical sense a good start should have better potential than those with a lower start value.

---

**Figure 3.16:** Illustration of *Best of Random Restart Hill Climb*. Only the best start points are climbed.

**Pros**
- Effective for smooth search spaces and high hit-percentages

**Cons**
- Suffer from low hit-percentage
3.5.8 Dynamical Constrained Random Restart HC (dcr-rHC)

While the previous algorithm was a remedy for expensive climbs this strategy is meant to be a remedy for search spaces with low hit-percentages. Its function is to combine the hits to dynamically change the outer limits or the external constraining. The hit-percentage is then likely be higher in this area and the *initially search* will as an effect use less time finding good or stable candidates. Further independent hits, to further expand this constraining, should be used to ensure that we do not get stuck in a local maximum area.

In this way it aims to figure out where the good or stable solutions are and try to intensify the search in this area. The coding should not be epistasis for this strategy to be fully effective. This is not the case with the DSM search space but it still worked to some extent. The hit-percentage did increase.

![Graph of Parameter value vs Parameter number](image)

Figure 3.17: Illustration of *Dynamical Constrained Random Restart Hill Climb*. Two hits can constrain a parameter area. Several independent hits can be used to dynamically increase this area.

Pros  
- Increases the hit-percentages for the *initial search*

Cons  
- No guarantee that the dynamic constraining includes optimum and interesting areas
CHAPTER 3. THE SEARCH

3.5.9 Search Algorithm Conclusion

The idea of heuristic search is to systematically use a strategy so that the fitness function can lead us to good solutions, hopefully nearly optimal or at least better than a random search.

But to guarantee a successful heuristic search we need to now our ground - the search space. This is because the strategy must somehow do sensible work in the search space. For instance for GA to outperform RS it is dependent on that its bit coding must have some beneficial effect on the search space which again makes it benefit from its mating. If the search space is smooth enough to be climbable a local search technique should be chosen instead.

It is also important to identify the computational cost when analyzing the effectiveness. In this we can adjust the algorithm to balance the computational effort to gain an effective strategy.

The strategies presented in chapter 3.5.7 and 3.5.8 are modification of random restart hill climb, and were produced to accommodate to two different types of search space characterizations, respectively high and low hit-percentages. These are adjustments that deal with polluted search spaces like that of the stability problems of the DSM landscape.

This is probably typical for heuristic search. One has to adjust its algorithms to suit the search space characterizations to gain an effective strategy.

If the search space is totally chaotic, random search is maybe as good as the rest since it is difficult to have any sensible strategy for the search space. If the coding can somehow isolate the parameters to make a better search space (map the genes) a suitable search strategy should handle these types of problems better than a random search. But to map the search space to a less chaotic space would probably be just as difficult as finding optimal solutions in the original chaotic search space.

The only way to compare different search methods is by a statistical comparison of the number of look-ups needed to find a certain fitness. This was not done since it was not the scoop for this thesis.
Chapter 4

The Results of Searches

This chapter will present the achieved result from a heuristic search for higher-order delta-sigma modulators. As earlier noted different search strategies could have been compared but this was decided to be omitted. This thesis scoop was not to find the best online way but to try to optimize the NTF for optimal SNR performance or by other optimality properties. The brrHC algorithm from chapter 3.5.7 was mostly used with 10 input signals of length $2^{16}$. This algorithm was chosen since its quantitative approach seemed to handle our search space the best.

4.1 Comparison Between Searched Designs and Peak Classical Designs

To make comparisons between arbitrary NTFs a characterization method was needed. A proposed method was presented in chapter 2.4.1. This method not only makes it possible to make a fairly thorough comparison between different designs but it can also be used to determine the maximum SNR design for filter design techniques. Let us start with a natural goal of reaching maximum SNR for a given OSR and modulator order.

4.1.1 Maximum SNR for an OSR of 64 and Modulator Order 5

To find the peak SNR performance for a Chebychev design we can use our proposed method to find the best trade-off between OBG and noise-attenuation. This was done for a fifth-order design with an OSR of 64 in figure 4.1. From the figure we conclude that a peak SNR performance for a Chebychev design is reached with an OBG value of approximately 1.68. The characterization and specification for this design is given in figure 4.2.
Then a borHC search was run for the same order and OSR as above. The resulting design is characterized and specified by figure 4.3. We can see that the Chebychev design has an estimated SNR performance of approximately 115.5 dB and a global $max_U$ of approximately 0.3. Whereas the searched design has an estimated SNR performance of approximately 121 dB and a global $max_U$ of 0.35. I.e. the search algorithm found a design that has an SNR performance which is approximately 5.5 dB higher than the peak Chebychev design.

When we compare the two specifications we see that the searched design has a less flat OBG characterization. The pole locations remind us a little of a Elliptic filter design.
Figure 4.3: Peak SNR performance and specification for a fifth-order searched design with OSR of 64

The graph in Figure 4.4 presents the search for maximum SNR in a similar way as was done for Figure 4.2 and 4.3 for OSR of 32, 64 and 128 and orders from three to seven. As we can see there is more to gain for higher orders. But an improvement of approximately 3, 5, 6, 8 and 10 dB for orders from three to seven is still a good improvement.

Figure 4.4: Performance comparison between peak Chebychev designs and searched designs
4.1.2 A Search for an NTF With High $\max_U$ Tolerance

We see from figure 4.1 that the Chebychev design approach has a problem reaching a $\max_U$ tolerance of 0.9 for a fifth-order modulator. Let us now compare a Chebychev design and a searched design which both have the aim to reach a $\max_U$ of 0.9.

From figure 4.1 we pick an OBG value of 1.1 to try to reach the goal of $\max_U$ of 0.9. The characterization and specification is given in figure 4.5. We can see that it nearly reaches the 0.9 specification and has an overall performance SNR of approximately 72 dB.

![Figure 4.5: Characterization and specification for a fifth-order Chebychev design with high $\max_U$ tolerance, the OSR is 64](image)

Now let’s try to search for a design capable of a $\max_U$ tolerance of 0.9. The result is given in figure 4.6. As we can see this design reaches its $\max_U$ goal of 0.9 and has an SNR performance of approximately 85 dB. Hence it has an SNR performance which exceeds the Chebychev design by 13 dB and also the $\max_U$ range is a little better.

![Figure 4.6: Characterization and specification for a fifth-order searched design with high $\max_U$ tolerance, the OSR is 64](image)
CHAPTER 4. THE RESULTS OF SEARCHS

4.2 Discussions of the Search Results

The search found stable design and outperformed classical design methods as can be seen figure 4.4. There is more to gain for higher orders as can be seen from this figure.

To find a tolerable design is a fast and easy task. But to optimize the solutions takes more time, approximately one hour for third-order and up to a whole day for high orders with low hit-percentages. This varies because of uncertainties of the initial search and that different start points have different climbing costs.

As can be seen from the searched designs frequency response, the OBG is not as flat as possible. They typically have a higher maximum OBG and also a higher mean OBG than those Chebychev produce. The zeros placements are almost the same but vary a little between different searched designs. The searched NTF have poles that remind us of elliptical [9] filter design poles.

As already concluded in chapter 2.4.3 on page 33 Chebychev is not well suited for making NTFs with high max\(V\) tolerance (> 0.85). It seems that the search outperforms Chebychev design even more for these kind of requirements, as shown in chapter 4.1.2.

The aim for this search was to reach the highest possible estimated SNR. To measure this and to compare different NTF solutions was a proposed method presented because this was not found in literature. Special implementation consideration was not taken. But the proposed design method can also be used to take such considerations as mentioned in chapter 2.6.3.
Chapter 5

Conclusion

The aim for this thesis was to perform a heuristic search to find stable high-order single-bit delta-sigma modulators (DSM). As we noted in the introduction in chapter 1 the search was motivated by what we felt were restrictions of the existing noise transfer function (NTF) design methods of today.

We have presented the following motivations for using heuristic search as a design approach for higher order delta-sigma modulators.

- No iron clad design methodology is available. We may therefore find surprising results not likely to accidentally result from a skilled designer. This can again increase insight about the science regarding delta-sigma modulators.

- The design approach gives us a direct access to modulator specifications. This gives us a flexibility to determine the parameters and the ability to search for NTFs for special purposes. In this way it can be a more automatic design process and maybe also a more optimal way to customize NTFs for special purposes.

- The ability to automatically find good NTFs for a given signal can fit for a DSM with an adaptive NTF implementation. This adaptive modulator can then benefit from both a high SNR performance and a high maximum allowed input range. This is not possible with today's static NTF designs.

A characterization method was needed to enable us to make a thorough comparison between arbitrary NTFs. No methods were found in literature and therefore a proposed method was produced. The idea is formulated in a paper which is submitted to Norchip 2006.

The proposed NTF design approach found stable high-order DSM designs with all sorts of specifications. The approach outperforms the classical design method, especially when high maxU tolerance is needed. The result given in this thesis can be repeated as follows.

- An SNR improvement from the classical Chebychev designs of approximately 3, 5, 6, 8 and 10 dB for modulator orders from three to seven were found, see figure 4.4 on page 70.
• A high $\max_U (\max_U = 0.9)$ tolerant fifth-order NTF design with an even higher SNR performance improvement of 13dB was found.

• The design had higher OBG values than the classical designs. So the searched solutions will not likely be found by other methods.

We therefore conclude that the search was successful. It was found to be an aggressive way to find high SNR performance versus given parameters of interest.
Chapter 6

Further Work

- Standardizing the proposed characterization method of chapter 2.4.1 further and producing a complete DSM characterization and verification toolkit as an extension to [29].

- Implement a searched design, preferable one that is weighted towards the given implementation as noted in chapter 2.6.3, in hardware and compare it with other design methods.

- Try to develop an adaptive NTF implementation as mentioned in chapter 2.6.5.
Bibliography


76
BIBLIOGRAPHY


Appendix A

Source Code

A.1 Source Code for DSM engines (mex code for matlab)

The code genMex.c and efoMex.c are the difference equation implementations for respectively the delta-sigma loop and the error feedback only implemented in the data language C to gain computational speed. They return the bitstream to the simulated modulator behavior. efoCheck.c and genCheck.c is special implementation of these which only returns a boolean, if the behavior was stable or not.

A.1.1 mexGen.c

```c
#include "mex.h"
#include <stdio.h>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray * phrs[])
{
    int n, i, j, a;
    double pq = 0, w = 0;
    double *x, *y, *a, *b, *w;
    x = mxGetPr(phrs[0]); /* Input signal */
    a = mxGetPr(phrs[1]); /* Loop filter.. */
    b = mxGetPr(phrs[2]); /* .. coefficients */
    n = mxGetNumberOfElements(phrs[0]); /* Length of input signal */
    o = mxGetNumberOfElements(phrs[1]); /* Modulator order */
    w = mxMalloc(o, sizeof(double));
    plhs[0] = mxCreateDoubleMatrix(1, n, mxREAL); /* Output array */
    y = mxGetPr(plhs[0]);

    /* Modulator loop */
    for (i=0; i<n; i++) {
        pq = 0;
        for (j=0; j<o; j++)
            pq += a[j]*w[j];
        y[i] = (pq >= 0) ? 1 : -1; /* The 1-bit truncator */
        w = x[i] - y[i];
        for (j=0; j<o; j++)
            w = b[j]*w[j];
    }
}
```

79
APPENDIX A. SOURCE CODE

```c
#include "mex.h"
#include <stdio.h>

void mexFunction(int nlhs, mxArray *plhs[1], int nrhs, const mxArray * prhs[1])
{
    int n, i, j, o;
    double pq = 0, w = 0;
    double *x, *y, *a, *b, *w_;
    x = mxGetPr(prhs[0]); /* Input signal */
    a = mxGetPr(prhs[1]); /* Loop filter */
    b = mxGetPr(prhs[2]); /* Coefficients */
    n = mxGetNumberOfElements(prhs[0]); /* Length of input signal */
    o = mxGetNumberOfElements(prhs[1]); /* Modulator order */
    w = mxMalloc(o, sizeof(double)); /* Output array */
    y = mxGetPr(plhs[0]);

    /* Modulator loop */
    for (i=0; i<o; i++) {
        pq = x[i];
        for (j=0; j<o; j++)
            pq += a[j] * w[j];
        y[i] = ((pq >= 0) ? 1 : -1); /* The 1-bit truncator */
        w = pq - y[i];
        for (j=0; j<o; j++)
            w -= b[j] * w[j];
        for (j=-1; j>0; j--)
            w_[j] = w_[j-1];
        w_[0] = w;
    }
    mxFree(w_);
}
```

A.1.2 mexEco.c

```c
for (j=-0; j>0; j--)
    w[j] = w[j-1];
```

```c
w[0] = w;
```
A.1.3 genCheck.c

```c
#include "mex.h"
#include <stdio.h>
#include <math.h>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray * prhs[],)
{
    int i, j, o, cc = 0, osr, n;
double pq = 0, w = 0, y, sb;
double amp, t;
double a, b, *w = m_malloc(sizeof(double) * n);
	sig = mxGetPr(prhs[0]); /**< Input signal spec */
a = mxGetPr(prhs[1]); /**< Loop filter .. */
b = mxGetPr(prhs[2]); /**< .. coefficients */

amp = sig[0]; /**< Amplitude of input signal */
f = sig[1]; /**< Normalized frequencies */

osr = (int) sig[2]; /**< OSR */

n = (int) sig[3]; /**< Length of signal */

w = mxGetNumberOfElements(prhs[1]); /**< Modulator order */

for (i = 0; i < n; i++) {
    pq = 0;
    for (j = 0; j < n; j++)
        pq += a[j] * w[j];
    if (pq > 0) {
        y = 1;
        cc++;
        if (cc > 32) { /**< String counter */
            boo[0] = 0;
            break;
        }
    } else {
        y = -1;
        cc = 0;
    }
    w = amp * cos(sb * 2 * M_PI * (i + 1) / n) - y;
    for (j = 0; j < n; j++)
        w[j] = a[j] * w[j];
    for (j = 0; j < n; j++)
        w[j] = w[j] - w[j - 1];

    w[0] = w;
}
m_free(w);
}
```
A.1.4 efoCheck.c

```c
#include "mex.h"
#include <stdio.h>
#include <math.h>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *
    prhs[]) {
    int i, j, o, cc = 0, osr, n;
    double pq = 0, w = 0, y, sb;
    double amp, t;
    double *a, *b, *w, *boo, *sig;

    sig = mxGetPr(prhs[0]); /* Input signal spec */
    a = mxGetPr(prhs[1]); /* Loop-filter */
    b = mxGetPr(prhs[2]); /* Filter coefficients */

    amp = sig[0]; /* Amplitude of input signal */
    t = sig[1]; /* Normalized frequencies */
    osr = (int) sig[2]; /* OSR */
    n = (int) sig[3]; /* Length of signal */

    o = mxGetNumberOfElements(prhs[1]); /* Modulator order */
    w = mxMalloc(o, sizeof(double));

    plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL); /* Output value */
    boo = mxGetPr(plhs[0]); /* Boolean wave stable */
    boo[0] = 1; /* Initially stable */
    sb = 1 + n/(2 * osr); /* Signal bandwidth */

    /* Modulator loop */
    for (i = 0; i < n; i++) {
        pq = amp * cos(sb * 2 * M_PI * (i + 1)/n);
        for (j = 0; j < o; j++)
            pq += a[j] * w[j];

        if (pq >= 0) {
            y = 1;
            cc++;
            if (cc > 32) { /* String counter */
                boo[0] = 0;
                break;
            }
        } else {
            y = -1;
            cc = 0;
        }

        w = pq - y;
        for (j = 0; j < o; j++)
            w[j] = b[j] * w[j];

        for (j = -1; j > 0; j--)
            w[j] = w[j - 1];
        w[0] = w;
    }

    mxFree(w);
}
```
A.2 Source Code for the Proposed Characterization Method

A.2.1 CVntf.m

```matlab
function [snr, maxu] = CVntf(ntf, maxUres, snrMres, fres, type, pl);

% A CV for for a NTF for delta sigma modulator.
% Usage: [snr, maxu] = CVntf(ntf, maxUres, snrMres, fres, 'type', pl);
% maxUres - resolution of the maxU determination
% snrMres - resolution of the snrM determination
% fres - pass-band frequency resolution
% type - Simulation model type, efo or gen
% pl - 0, no plots
% pl = 1, only maxU determination and plot
% pl = 2, both maxu and snr are plotted

fu = 0.1; fes = 1; % Make passband frequencies array
maxu = maxU(ntf, maxUres, fek, type); % Perform maxU estimate

fes = fu{3:end-1}; % Can not measure SNR near dc and passband edge
if pl == 1 & (maxu > 0.01)
    amp = min(maxu);
    snr = snrM{ntf, amp, snrMres, fres, type};
else
    snr = 0;
end

% Plot formatting
if pl == 1
    figure
    hold on
    plot(feu, maxu, 'k');
    box off;
    grid on;
    xlabel('Normalized pass-band frequency', 'FontSize', 12);
    ylabel('max U', 'FontSize', 12);
end

if pl == 2
    figure
    [AX, HI, H2] = plotyy(feu, maxu, fres, snr);
    ylabel('Normalized pass-band frequency', 'FontSize', 12);
    ylabel('SNAD dB', 'FontSize', 12, 'Color', 'k');
    set(HI, 'Color', 'k');
    set([H2, 'Color', 'k']);
    set([AX(1), 'YColor', 'k']);
    set([AX(2), 'YColor', 'k']);
    set(minv = floor(min(maxu+10)/10);
    maxv = ceil(maxu+10)/10;
    axis([AX(1), [0 1 minv maxv]]);
    axis([AX(2), [0 1 minv maxv]]);
    box off;
    grid([AX(1), 'on']);
    grid([AX(2), 'on']);
end
```
A.2.2 maxU.m

```matlab
function maxu = maxU(ntf,duration,fe,type);
% Determination of maxU properties of a efo NTF
% Usage: maxu = maxU(ntf,duration,fe,type);
% duration - Duration of stability test signals
% fe - Normalized pass-band frequencies array
% type - Simulation model type, efo or gen

res = 1e-3; % Descimal stop criteria for amplitude
N = 2^duration;
pfit = 0; % Polynomial fit? (variance estimate, see below)

% Set check function and respective loop-filter coeffs after type
if type == 'efo'
    chec = @efoCheck;
    [a,b] = ntf2efo(ntf);
elseif type == 'gen'
    chec = @genCheck;
    [a,b] = ntf2gen(ntf);
end

% Perform max_U estimation, a *** half search in amplitude
maxamp = zeros(1,length(fe));
i = 0;
for i = fe
    stp = 1;
    amp = 1;
    while res < stp; % Stop criteria: while step < than given resolution
        stp = stp/2;
        if feval(chec,[amp f ntf,osr N],a,b)
            maxamp(i) = amp;
            amp = amp + stp;
        else
            amp = abs(amp - stp); % Negative values can occur
        end
    end
end

if pfit % Instead of min can we choose a variance estimate
    P = polyfit(fe,maxamp,1);
    Y = polyval(P,fe);
    [ma,index] = max(Y-maxamp);
    maxu = Y - ma;
else
    maxu = maxamp;
end
```
A.2.3 snrM.m

function snrm = snrM(ntf,amp,duration,fe,type)
% Broad snr measurement, measure SNR estimate of several frequencies
% Usage: snrm = snrM(ntf,amp,duration,fe,type)
% amp — Amplitude for the input signals
% duration — Duration for the input signals — amount of fft-points
% fe — Normalized pass-band frequencies for input signals
% type — Simulation model type, efo or gen

if type == 'efo'
    func = @mexEfo;
    [a,b] = ntf2efo(ntf);
end
if type == 'gen'
    func = @mexGen;
    [a,b] = ntf2gen(ntf);
end

maxSnr = zeros(1,length(fe));
N = 2^duration;
pb = N/(2*ntf.osr); % passband
i = 0;
for nf = fe
    i = i+1;
    x = amp*DSain(nf,N,ntf.osr);
    yn = feval(func,x,a,b); % A mex engine takes care of the simulation
    [f,s] = sy(yn,N,3,pb);
    [snr,linex,linery] = snl(f,s,pb);
    maxSnr(i) = snr;
end
snrm = maxSnr;
A.3 Source Code for the Search Algorithms

A.3.1 newAliens.m (Random Search)

```matlab
function [aliens, hp] = newAliens(n, nCoef, func, crit, A, x)
    % A Random Search which tries to find n new good/stable individuals
    % which are accepted by the criteria crit
    % Usage: [aliens, hp] = newAliens(n, nCoef, func, crit, A, x)
    % n - Number of wanted individuals
    % nCoef - Number of parameters to the fitness function
    % crit - Fitness criteria to be good/stable or accepted
    % func - The fitness function
    % A - Parameter area matrix, A = [a(1, :), ... ; a(n, :)]
    % Returns the aliens and the hit-percentage
    aliens = zeros(n, nCoef+1);
    cnt = 0;
    [w, a] = size(A);
    % Loop that tries to find n new stable candidates
    for i = 1:n
        a = zeros(1, nCoef+1);
        % As long as we do not have a new stable candidate
        while 1
            cnt = cnt + 1;
            % Counter
            a(2:end) = rand(w, 1) - (A(:, 2) - A(:, 1)) / A(:, 1);
            if a(1) < crit;
                % If criteria is met
                fprintf('Found : %g, num2str(a(1));
                % Breaks the while-loop
                break;
            end
        end
        aliens(i, :) = a;
    end
    hp = n + 100 / cnt;
    disp(sprintf('Hit-percentage %g', hp));
```

A.3.2 HillClimb.m

```matlab
function [fittest, steps] = HillClimb(flag, fitnessFunc, maxG, step, x);
    % A Hill-Climb search strategy implementation with gradient search.
    % Climbs the steepest in +/- stepsize for every parameter and the
    % climbed gradient.
    % Usage: [fittest, steps] = HillClimb (flag, fitnessFunc, maxG, step, x);
    % flag - Start point
    % fitnessFunc - Fitness function
    % maxG - Maximum steps allowed
    % step - Size of step step
    % Returns the climbed point (flag) and number of climbed steps
    start = now;
    nCoef = length(flag) - 1;
    % Time stamp
    cd = 7;
    % Number of parameters to func
    % Some needed memory information about the climbs
    new = zeros(nCoef + 2, 1, nCoef + 1);
    mem = zeros(10, nCoef);
    zPcent = 0;
    cPcent = 0;
    % Zero progression counter
    % Continuous progression counter
    gcent = 0;
    % Gradient progression counter
    gradient = 0;
```
memi = 0;
gen = 0;

% Hill-Climb loop
while gen < maxG & zPcnt < 5 % Stop conditions
  % First search in every parameters +/- direction
  for n = 1:nCoef
    % Coef Index, Pos and Negative Direction
    ci = n+1; pd = n; nd = n-nCoef;
    new(pd,:) = [inf flag(2:end)];
    new(nd,:) = [inf flag(2:end)];
    new(pd,ci) = new(pd,ci) + step;
    new(nd,ci) = new(nd,ci) - step;
    new(pd,1) = feval(fitnessFunc, new(pd,2:end), 0, x);
    new(nd,1) = feval(fitnessFunc, new(nd,2:end), 0, x);
  end
  new(nCoef*2+1,:) = [inf (flag(2:end)+gradient)]; % Also try gradient
  new(nCoef*2+1,:) = feval(fitnessFunc, new(nCoef*2+1,2:end), 0, x);
  [y, i] = min(new(:,1)); % Point and index of the best new
  new = sortrows(new); % sort the news
  if new(1,1) < flag(1); % If we have found a positive climb
    zPcnt = 0; % Update zero progression
    cPcnt = cPcnt + 1;
    flag = new(1,:); % New flag point
    memi = memi + 1; if memi > 10, memi = 1; end
    if i == nCoef+2+1 % If climbed in gradient direction
      gcnt = gcnt + 1;
      memi(memi,:) = gradient;
    else % Or climbed in a parameter direction
      gcnt = 0;
      if i > nCoef
        i = nCoef-i; % Which way we climbed
      end
      % Update memory
      memi(memi,:) = zeros(1,nCoef);
      memi(memi,abs(i)) = sign(i)*step;
    end
    gradient = sum(memi)/10; % Update gradient
    if gcnt > 2 % Increase gradient value if gcnt > 2
      gradient = gradient + 2;
    end
  end
  gen = gen + 1;
  if cPcnt > 10 % Increase step size if cPcnt > 10
    step = step*2;
    cPcnt = 0;
  end
  if plot
    feval(fitnessFunc, flag(2:end), 1);
    pause(1);
  end
  else % If none of the points were better
    zPcnt = zPcnt + 1;
    cPcnt = 0;
    step = step/10;
  end
end
fittest = flag;
steps = gen;

% A DSM specific information
 global osr;
 ntf = rad2num(flag(2:end), osr);
 [snr, maxCBG, meanCBG] = DSfreqNtf(ntf,nf,inf,0);
 disp(sprintf('Climbed to %1.5g by %ld steps, took %f sec -> HeI=%3.4g, He2=%3.4g, snr=%3.4g', ..., flag(1), gen, (now-start) + 60*60*24, maxCBG, meanCBG, snr));
A.3.3 borHC.m

```matlab
function fittest = borHC(popSize, deep, amp, n, OSR);
% The 'Best of random restart Hill Climb' search algorithm
% Hybrid of random search and hill climb. Becomes rhHC if sr = 1.
% Usage: fittest = borHC(popSize, deep, amp, order, OSR);
% popSize = amount of generated candidates
% z = Modulator input signal specification
% n = modulator order
% OSR = modulator oversampling rate
% fittest = Fitness function

cle;
global osr;

n = now;

% Choose standard parameters areas, 0 to 1.
SA = zeros(nCoef,2);
SA(:,1) = 0;
SA(:,2) = 1;

fittestFunc = @efoRALZM;

fittestFunc = @efoEllip;

sr = 0.1;

crit = -10;

x = [amp 0.2 0.3 0.5 0.7 0.9];

disp(fittestFunc);

disp(sprintf('borHC with modulator order %d ', nCoef));

disp(sprintf('and specification: deep=%d osr=%d amp=%f = started', ...

    deep, osr, amp));

% First a random initial search for stable/good solutions
[pop, hp] = newAliens(popSize, nCoef, fittestFunc, crit, SA, x);

pop = sortrows(pop);

for i = 1:round(popSize*sr);

disp(sprintf('Climbs from %s', num2str(pop(i,:))));

end

pop = sortrows(pop);

fittest = pop(1,:);

disp(sprintf('borHC with modulator order %d ', nCoef));

disp(sprintf('and specification: deep=%d osr=%d amp=%f = finished', ...

    deep, osr, amp));
```

A.3.4 dcrrHC.m

```matlab
function fittest = dcrrHC(runs,deep,amp,n,CSR);
% The 'Dynamical Constrained rrHC' search algorithm
% Usage: fittest = dcrrHC(runs,deep,amp,n,o);
% runs - Amount of search rounds
% deep - How deep the climb should be
% amp - Wanted max u range
% n - Number of fitness function parameters
% CSR - Oversampling rate

nCoef = n; % Number of coefs;
global osr; % Set the global CSR
osr = CSR;

% Choose standard parameters areas, 0 to 1.
SA = zeros(nCoef,2); % Start area
DA = zeros(nCoef,2); % Dynamical area
SA(:,1) = 0; SA(:,2) = 1; % Start area

fitnessFunc = @refRAZM; % Set fitness function
x = [amp 0.2 0.3 0.5 0.7 0.9]; % Fitness specification
krit = -10; % Criteria for a good/stable solution

clear;

% Start the algorithm with some information
disp(fitnessFunc);
disp(sprintf('dcrrHC with modulator order %d ', nCoef));
disp(sprintf(' and specification: deep-%d osr-%d amp-%f - started ', ...
        deep, osr, amp));

% Initial SA round, before we can start the while loop
sah = newAllians(1,nCoef,fitnessFunc,krit,SA,x);
[sah, steps] = hillClimX(sah,fitnessFunc,deep,0.1,x);
DA(:,1) = sah(2:end)';
fittest = sah;

cnt = 0;
while cnt < runs;
    cnt = cnt + 1;
    disp(sprintf('-------- Generation %d started -------- ', cnt));
    % First a new independent hit
    dah = [inf zeros(1,nCoef)];
    sah = newAllians(1,nCoef,fitnessFunc,krit,SA,x);
    [sah, steps] = hillClimX(sah,fitnessFunc,deep,0.1,x);

    % Update DA
    for i = 1:nCoef
        if sah(1,i+1) < DA(i,1)
            DA(i,1) = sah(i+1);
        elseif sah(1,i+1) > DA(i,2)
            DA(i,2) = sah(i+1);
        end
    end

    % Then several hit from dynamical constrained area (DA)
    pop = newAllians(5,nCoef,fitnessFunc,krit,DA,x);
    pop = sortrows(pop);
    dah = pop(1,:);
    dah = hillClimX(dah,fitnessFunc,deep,0.1,x);
    for i = 1:nCoef
        if dah(i+1) < DA(i,1)
            DA(i,1) = dah(i+1);
        end
        if dah(i+1) > DA(i,2)
            DA(i,2) = dah(i+1);
        end
    end
end
```

APPENDIX A. SOURCE CODE
end

% Update if we have found a new better solution
if sah(1) < dah(1)
    bh = sah;
else
    bh = dah;
end

if bh(1) < fittest(1)
    fittest = bh;
    disp(sprintf('*** New best %s ***', num2str(fittest))); end

disp(fitnessFunc);
disp(sprintf('dcrnHC with modulator order %d ', nCoef));
disp(sprintf(' and specification: deep-%d osr-%d am-%f - finished ', ...
    deep, osr, amp));
A.4 Source Code for a Fitness Function Example

A.4.1 efoRADZM.m

```matlab
function fitness = efoRADZM(A,p1,x);

% A fitness function implementation, fitness is the estimated SNR
% of the NTF solutions for the given input signal x. The parameters A
% is in rad2 format.
%
% Usage: fitness = efoRADZM(A,p1,x);
%
% A = Fitness function parameters
% p1 = plot?
% x = Input signal specifications

n = now;
global osr;
ntf = rad2ntf(A,osr);
krit = 1.0;
[snr, maxOEG, meanOEG] = DSfreqNtf(ntf,krit,0);

% If accepted by internal constraining
if maxOEG < krit ...
    & maxOEG < 1.5 ...
    & snr > 50 ...
    & stabCheck(ntf,x(1),[0:0.01:0.1 x(2:end)],20,'efo')

% Fitness measurement
fitness = - mean(snrM(ntf,x(1),16,x(2:end),'efo'));

% Can print the hits to screen if wanted
if p1 == 1 & fitness < -10;
    disp(sprintf('Found : maxOEG %f, meanOEG %f and snr: %f, ga %g', ...
        maxOEG,meanOEG,snr,fitness));
    disp(sprintf('fitness = %s', num2str(A)));
end
else
    fitness = inf;
end
```
A.5 Source Code for Some Needed Functions

A.5.1 zp2ntf.m and ntf2zp.m

```matlab
function NTF = zp2ntf(z, p, osr);
% zplane to simple NTF format
% Usage: [z, p] = synthMe(N, obj, osr, p1);
NTF.z = z;
NTF.p = p;
NTF.osr = osr;
```

```matlab
function [z, p, osr] = ntf2zp(NTF);
% Transform from ntf format to zplane + osr specification
% Usage: [z, p, osr] = ntf2zp(NTF);
z = NTF.z;
p = NTF.p;
osr = NTF.osr;
```

A.5.2 ntf2gen.m and ntf2efo.m

```matlab
function [a, b] = ntf2gen(NTF);
% Transform from NTF to loop-filter coefficients for the
% delta-sigma loop, direct form loop-filter coefficients
% Usage: [a, b] = ntf2gen(NTF);
[nume, dene] = zp2tf(z, p, 1);
b = nume(2:end);
num = dene - nume;
a = nume(2:end);
```

```matlab
function [a, b] = ntf2efo(NTF);
% Transform from NTF to loop-filter coefficients for the
% Error Feedback Only direct form loop-filter coefficients
% Usage: [a, b] = ntf2efo(NTF);
[nume, dene] = zp2tf(z, p, 1);
b = dene(2:end);
a = dene - nume;
a = a(2:end);
```

A.5.3 DSsin.m

```matlab
function y = DSsin(nfreq, length, osr);
% Makes a sinus signal from given normalized freq (0/dc to 1)
% to passband.
% Usage: y = DSsin(nfreq, length, osr);
% nfq = Normalized frequency, f = [0, 1]
% length = Wanted length
% osr = Oversampling rate
N = length;
if nfreq == 0 % If DC
    y = ones(1, N);
else % If AC
```
A.5.4 radz2ntf.m, rad2zp.m and DSzero.m

```matlab
function NTF = radz2ntf(A, osr);
% Transform from radz format to NTF format
% Usage: NTF = radz2ntf(A, osr);
1 l = length(A);
2 % How many zeros
3 nzero = floor(l/3);
4 o = l - nzero;
5 % Transform the first coeffs to poles
6 p = zeros(o,1);
7 if mod(o,2) % hvis odder har vi en pol på x-aksn (Re aksen)
8 p(o) = rad2zp(0, A(o),0);
9 o = o - 1;
10 end
11 % Here we can constrain the poles inside 1. and 4. quadrant
12 for k = 1:2:o
13 p(k) = rad2zp(A(k)+ad, A(k+1),0); % 0.5+0.5*
14 p(k+1) = rad2zp(-A(k)+ad, A(k+1),0);
15 end
16 % Transform the last nzero coeffs to zeros
17 z = DSzero(A(o+1:end),0,osr,0);
18 NTF = zp2ntf(z,p,osr);
end
```

```matlab
function c = rad2zp(rad, r, pl);
% Transform from radian format to coordinates in the z-plane
% Usage: p = rad2zp(rad, r, pl);
1 c = r*cos(rad) + j*sin(rad);
2 if pl
3 zplane(p,[1]);
4 end
end
```

```matlab
function z = DSzero(c, o, osr, pl);
% from zero in the pushed up circle to z-plane
% Usage: z = DSzeros(c, o, osr, pl);
1 wb = pi/osr;
2 z = ones(o,1);
3 l = o-length(c)+2;
4 i = l+1;
5 for ci = 1:length(c)
6 z(i) = cos(wb*real(ci)) + j*sin(wb*imag(ci));
7 z(i+1) = cos(wb*real(ci)) - j*sin(wb*imag(ci));
8 i = i+2;
9 end
end
```

A.5.5 snd.m, sy.m and bhh.m

These programs is used to estimate SNR. They are written by Mats Hovin.

```matlab
function [snd_out, lineex, liney] = snd(f, s, bw, plott);
```
% Calc S/N or in dB of f,s in dB
% s must be in dB
% Syntax: fsnd_out, liney = snd(f,s,bw,plot);
% liney = dc cut, sig_low cut, sig
% sig_high cut
% Removes 8 start bin from s
% OPTIONAL: if exist plot - plots f,s with cut frequencies in fig 11
% bin no's start at 1, not 0
% ok 14/08/01

N-length(s);
if N<30 disp('VELDIG KORT S VÆKSTE !!!'); hopp=1; snd_out=0; linex=[0 0.01]; liney=linex; end;
if ~hopp
    bin_fre=f(2);
    bwBin=floor(bw/bin_fre)+1;
    if bw_bin>N disp('TM FCR STOR FOR DATA !!!'); hopp=1; snd_out=0; linex=[0 0.01]; liney=linex; end;
    if bw_bin<20 disp('VELDIG LITE BW DATA !!!'); hopp=1; snd_out=0; linex=[0 0.01]; liney=linex; end;
else
    if ~hopp
        sig, sig_bin = max(s(9:bw_bin));
        sig_bin = sig_bin+8;
        sig_fre = (sig_bin-1)+bin_fre;
        cut_dc_bin = 8;
        cut_dc_fre = (cut_dc_bin-1)+bin_fre;
        cut_sigm_bin = (sig_bin-7);
        cut_sigm_fre = (cut_sigm_bin-1)+bin_fre;
        cut_sigm_max_bin = min(sig_bin+7,N-1);
        cut_sigm_max_fre = (cut_sigm_max_bin-1)+bin_freq;
        if (cut_sigm_max_bin-1)+bw_bin disp('SIGNAL FCR NÆR BW !!!'); hopp=1;
            snd_out=0; linex=[0 0.01]; liney=linex; end;
        if (cut_sigm_min_bin-1)+(cut_dc_bin-1) disp('SIGNAL FCR NÆR DC !!!');
            hopp=1; snd_out=0; linex=[0 0.01]; liney=linex; end;
else
        if ~hopp
            Sn = [s(cut_dc_bin+1:cut_sigm_min_bin-1) cut_sigm_min_bin+1:
                  bw_bin];
            Psn_tot = 10*log10( sum(10.^(Sn/10)));
            snd_out = sig - Psn_tot;
            liney=[cut_dc_freq cut_sigm_freq sig_freq cut_sigm_max_freq];
            liney=[sig sig sig]
        end;
    end;
if exist('plott') & ~hopp
    figure(11);
    plot(s(1:bw_bin),s(1:bw_bin),'k'); hold on
    plot(s(1:bw_bin+1:N),s(bw_bin+1:N),'r');
    line([linex(1) linex(1),], linex(1) min(s)], 'color', 'b', 'linestyle', ' ');
    line([linex(2) linex(2)], linex(1) min(s)], 'color', 'b', 'linestyle', ' ');
    line([linex(4) linex(4)], linex(1) min(s)], 'color', 'b', 'linestyle', ' ');
    plot([sig_fre],[sig],'.r');
    ylabel('Power Spectral Density (dB)');
    xlabel('Frequency (Hz)');
title(['S/N = ' num2str(round(snd_out)) ' dB (' ...
                          num2str(round(log2(10.^(snd_out/20)+10)) / 10) ... '
                          bit'));
end;
APPENDIX A. SOURCE CODE

68  \texttt{grid on;}
69  \texttt{hold off;}
70  \texttt{zoom on;}
71
72  \textbf{function} [f,s]=sy(data,fdata,win,bw)
73  \% Modified periodogram PSD estimator -- provides frequency axes
74  \% Returns freq axes from 0 -- (fdata/2). Removes DC power.
75  \% win=0: Rect, win=1: Hamming, win=2: Blackman, win=3: BHH.
76  \% syntax: [f,s]=sy(inndata, freq_of_inndata, window, bw); \texttt{plot}(f,s)
77  \% OPTIONAL: bw -- if given, \texttt{p,f} is clipped to bandwidth
78  \% Window loss: BHH--4.3 dB, Blackman--3.4 dB, Hamming--1.6 dB, Rect--0 dB
79  \% ok 13/08/01 Mats
80
81  Z=size(data);
82  if \(Z(2)==1\), data=data';\texttt{end;}
83  N = \texttt{length(data)};
84  NofBinOut = \texttt{floor}(N/2)+1;
85  BinFreq = \texttt{fdata}/N;
86  MaxFreqOut = (NofBinOut-1)*BinFreq;
87  \% nargin=3
88  f=0:BinFreq:MaxFreqOut;
89  \texttt{end;}
90
91  if nargin=4
92  f=0:BinFreq:bw;
93  \texttt{end;}
94
95  data = data - \texttt{mean(data)};
96
97  if win=3
98  \texttt{U} = \texttt{mean(bhh(N),2);}
99  \texttt{s} = ((\texttt{abs(fft(data.*bhh(N))))}^2)/\texttt{(N*U)};
100  \texttt{end;}
101
102  if win=2
103  \texttt{U} = \texttt{mean(blackman(N),2);}
104  \texttt{s} = ((\texttt{abs(fft(data.*blackman(N))))}^2)/\texttt{(N*U)};
105  \texttt{end;}
106
107  if win=1
108  \texttt{U} = \texttt{mean(hamming(N),2);}
109  \texttt{s} = ((\texttt{abs(fft(data.*hamming(N))))}^2)/\texttt{(N*U)};
110  \texttt{end;}
111
112  if win=0
113  \texttt{s} = \texttt{(abs(fft(data))}^2)/\texttt{N;}
114  \texttt{end;}
115
116  s=10.*\texttt{log10(s)};
117  s=s(1:\texttt{length(f)});

1  \textbf{function} [w]=bhh(bigN)
2  \%
3  \%
4  \% bhh -- The Blackman--Harris--Hoodie window
5  \%
6  \%
7  \% syntax: bhh(length)
8  \%
9  \% ok 13/08/01 Mats
A.5.6 DSfreqzNtf.m

```
function [snr, maxOBB, meanOBB] = DSfreqzNtf(nft, krit, pl);

% Frequency response of both the NTF and the STF of an
% given ntf.
% Usage: [snr, maxOBB, meanOBB] = DSfreqzNtf(nft, krit, pl);

% ntf - Specification of the ntf
% krit - only OBB < krit is wanted can this be used, see below
% pl - no plot
% 1 plots frequency response
% 2 plots above and the z-plane

N = 128; % Resolution of frequency response
CSR = ntf.osr;

[num, den] = zp2tf(nft.z, nft.p, 1); % Transform ntf to transfer func.
num = den = num; % NTF to STF transformation...

w = 1e3 .* pi ./ (N-1); pi = 1e3;
[Hx, w] = freqz(num, den, w); % Freq. response of STF
[He, w] = freqz(num, den, w); % Freq. response of NTF

He = 20 * log10(abs(He)); % To dB format
% .. ditto

% Begin to plot the freq. response
if pl
    figure(11)
    hold off;
    f = 0.1 ./ (length(w) - 1) : 1;
    plot(f, Hx, 'k');
    hold on;
    plot(f, He, 'k');
end

obj = 10 .* (He / 20); % From dB to normal gain
maxOBB = max(obj); % Determination of max OBB
meanOBB = mean(obj(round(N/CSR:end))); % Mean OBB

% Only compute the snr if maxOBB accepted
if maxOBB < krit
    w = 1e3 .* pi ./ (N-1) .* CSR : (pi / CSR) : 1e3;
    [He, w] = freqz(num, den, w); % Almost always 1
    snr = -20 * log10(abs(He));
    He = 20 * log10(abs(He));
else
    snr = 0;
end

if pl % Plots the in-band freqz response
    f = 0.1 ./ (length(w) - 1) : 1;
    plot(f, He, 'k');
    plot(f, Hx, 'k');
end
```
A.5.7 stabCheck.m

```matlab
function stable = stabCheck(ntf, amp, fes, dur, type)

% Takes several stability checks for several frequencies in the pass band
% Usage: stable = stabCheck(ntf, amp, fes, dur)
% %
% % amp = Amplitude of stability test input signals
% % fes = pass band frequency resolution
% % dur = duration of the test signals, format 2^dur
% % type = Simulation model type, efo or gen

% Set check function and respective loop - filter coefs after rtype
if type == 'efo'
    chec = @efoCheck;
    [a,b] = ntf2efo(ntf);
end
if type == 'gen'
    chec = @genCheck;
    [a,b] = ntf2gen(ntf);
end

stable = 1;
for i = fes
    if abs(eval(@efoCheck, [amp f i*nf osr 2^dur], a, b))
        stable = 0;
        break
    end
end
```
Appendix B

Paper Submitted to Norchip 2006

Follows on the next pages.
Appendix C

Paper To Be Submitted Shortly

Follows on the next pages.