NTNU
Innovation and Creativity

Low Power Continuous-Time Delta-Sigma ADC

The robustness of finite amplifier GBW compensation

Jon Helge Nistad

Master of Science in Electronics
Submission date: June 2006
Supervisor: Trond Ytterdal, IET
Co-supervisor: Are Hellandsvik, Atmel Norway AS

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

The object of this thesis is modeling and simulation of a 2-order single bit continuous-time delta sigma ADC with 10 ENOB accuracy and 50 kHz signal bandwidth using VHDL-AMS. The work will focus on implementation and testing of a compensation technique for finite amplifier GBW in the integrators, where the main goal is to see how an ADC employing the compensation technique is affected by nonidealities compared to a similar ADC without compensation.

Assignment given: 16. January 2006
Supervisor: Trond Ytterdal, IET

Abstract

This paper reports on the modeling and simulation of a continuous-time $\Delta\Sigma$ *analog to digital converter* (ADC) in *VHDL AMS*. The ADC is intended for use in a microcontroller and is therefore underlying restrictions on power consumption. Continuous-time $\Delta\Sigma$ architectures are well known for their good low-power capabilities compared to discrete-time realizations. This is due to their reduced demands to the *gain bandwidth product* (GBW) of the internal amplifiers in the ADCs. Continuous-time ADCs often operate with GBWs in the range of the *sampling frequency* (f_s). The ADC presented in this work is also employing a previously reported compensation technique which ideally allows the GBW to be reduced further > 20 times of this. Considering that the current drain in the amplifiers usually is proportional with GBW, this could be a promising power saving technique.

The work focuses on the development of two similar models of a 2-order continuous-time $\Delta\Sigma$ ADC in *VHDL-AMS*, where one of the ADCs is using the compensation technique. The main purpose is to see how the compensated ADC is affected by nonidealities such as GBW-variation, finite amplifier gain, RC-product variation, excess loop delay and finite DAC slew rate compared to the performance of the noncompensated ADC. The required accuracy for the modeled ADCs is $62dB$ *Signal to Noise and Distortion Ratio* (SNDR), thus an appropriate *oversampling ratio* (OSR) also must be found.

The simulations show that the compensated ADC has similar performance as the noncompensated ADC operating with $GBW = 10f_s$ when subject to the different nonidealities. With an $OSR = 64$ it stays within the accuracy specification for GBWs $\geq 0.05f_s$. This is however *only* valid if actual GBW stays within $\pm 40\%$ of the GBW compensated for. For larger deviations, especially lower GBW values, the SNDR drops rapidly. It is also shown that the internal signal swing in the ADC is reduced for low GBW values. This may limit the practical achievable SNDR when subject to circuit noise. If these potential drawbacks are circumvented, the compensation technique could lead to a further decrease of the power consumption in continuous-time $\Delta\Sigma$ ADCs.

Acknowledgements

This diploma thesis was submitted to the Department of Electronics and Telecommunications at the Norwegian University of Science and Technology (NTNU). The work presented in this diploma thesis was carried out during the spring 2006, under the supervision of Professor Trond Ytterdal, NTNU and Are Hellandsvik, Atmel Norway. I would like to acknowledge my two supervisors for their great encouragement and support.Their helpfulness and competence has been very inspiring during my work with this thesis.

I would also like to thank Øystein, Andreas, Torgeir and Anita for cheerful friendship and constructive suggestions.

Trondheim, 12th June 2006

Jon Helge Nistad

Contents

List of Figures

Chapter 1

Introduction

An important part of modern microcontrollers is the *analog to digital converter* (ADC). If the microcontroller is supposed to do calculations based on analog signals from e.g. analog sensors, the need for an internal ADC is apparent. In a microcontroller environment there are tight restrictions on the power consumption. The growing market of battery powered hand-held applications is especially demanding low power devices.
Oversampling data converters such as $\Delta\Sigma$ ADCs are often used when high resolution and resonably low signal bandwidths like $50-100\ kHz$ is needed. This is typical single loop, first or second order discrete time $\Delta\Sigma$s with high *oversampling ratios*(OSR) and 1-bit quantizers. In these converters the power consumtion is by far limited by the high sampling rates. This is because the amplifier bandwidth needed in discrete time modulators is in the range of 5-10 times the *sampling frequency* (f_s). An approach to reduce the OSR is to use higher order modulators and multibit quantizers, but this tends to heavily increase the complexity of the design.
In stead of discrete-time filters, modulators with continuous-time filters has gained popularity the past few years. These modulators only need amplifiers with bandwidths in the range of f_s and also has implicit anti-alias filtering which significally relaxes the demands on preceding filter stages. Thus the overall ADC power consumption is significantly reduced compared to discrete-time realizations.
The work presented here is using continuous-time circuitry, and is focusing on a compensation technique presented in [Ortmanns, 2004] which could decrease the required amplifier bandwidth additional 10-20 times if it is feasible to use in a circuit realization. A continuous-time $\Delta\Sigma$ ADC employing the compensation technique will be modelled in [VHDL-AMS], and compared to a similar ADC without the compensation during simulations. The main purpose of this work is to see how circuit imperfections and nonidealities affect the performance of the compensated ADC compared to the noncompensated one.

Chapter 2

Theory

This chapter covers the fundamentals of A/D conversion, an introduction to $\Delta\Sigma$-modulation and theory regarding development of continuous-time $\Delta\Sigma$-modulators.

2.1 Analog to Digital Conversion

2.1.1 Basics of A/D conversion

The conversion of a continuous-time analog signal into a digital one is done in two operations. First there is a sampling of the of the analog signal (usually with a constant *sample period* T_s), then a quantization of the signal amplitude is done. If the signal band of a sampled signal is less than half the sampling frequency, the sampling in time is a completely invertible process. Looking at the frequency spectrum of a sampled signal in figure 2.2 this could be understood. When a signal is sampled at uniform time intervals, this results in a periodicity of the signal spectrum at multiples of the sampling frequency f_s in the frequency domain as seen in the figure. With simple low-pass filtering it is clear that the original baseband spectrum can be reconstructed as long as the spectrums doesn't overlap. This is achieved when

$$f_s \geq 2f_0 = f_N \tag{2.1}$$

where f_0 is the bandwidth of the input signal. This equation is known as the *Nyquist theorem*, and f_N is called the *Nyquist frequency*. An analog filter preceeding the sampling operation is required to assure that the input signal bandwidth is limited to f_0. This filter is known as the *antialiasing filter* (AAF). A basic ADC structure is shown in figure 2.1. An ADC working at a sampling frequency that equals to f_N is called a *Nyquist Rate* converter, these converters are hard to design in practice because of the zero transition band required for the AAF (p.8 [Ortmanns, 2006]). To overcome this problem, this type of converters often use a slight amount of oversampling. The

oversampling ratio (OSR) is defined as

$$OSR = \frac{f_s}{2f_0} \tag{2.2}$$

Nyquist rate converters operates in most cases with an $OSR = 1.5 \rightarrow 10$ [Johns, 1997]. Increasing the OSR greatly relaxes the demands to the AAF, thus simplifies the design and reduces the power and chip area of the filter (p.8 [Ortmanns, 2006]).

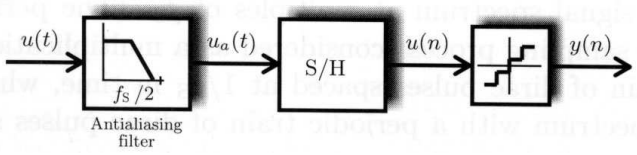

Figure 2.1: Block diagram general ADC(fig. 2.2 [Ortmanns, 2006])

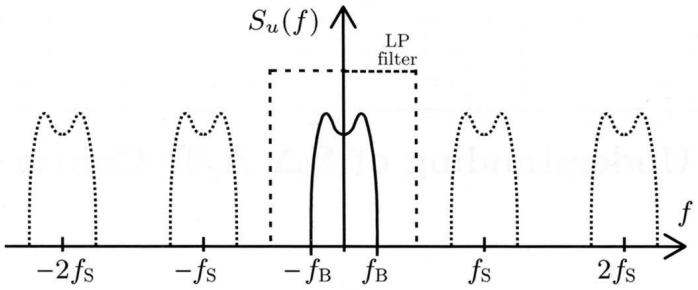

Figure 2.2: Spectral sampling operation (fig. 2.1 [Ortmanns, 2006])

The quantizer encodes a continuous range of analog values into a set of predefined discrete levels. Quantization is usually uniform and the space between two adjacent output levels of the quantizer is defined as the *quantizer step width*:

$$\Delta = \frac{FS}{2^{B_{int}} - 1} \tag{2.3}$$

where FS is the *full-scale* input range and $2^{B_{int}}$ is the number of different output levels. Since an infinte number of input values of the sampled input signal is mapped to an finite number of values in the quantizer, the quantization is an noninvertible process.

2.1.2 White noise assumption

If the input signal $x(n)$ has a *rapidly and random* variying behavior, the quantization noise $e(n)$ can be approximated as a random number uniformly

distributed between $\pm\frac{\Delta}{2}$ and uncorrelated with its previous values, where Δ is the step size of the quantizer. It is also assumed that $e(n)$ have statistical properties independent of $x(n)$. By these properties, $e(n)$ is classified as white noise with a mean square value of $e_{rms}^2 = \frac{\Delta^2}{12}$ (p.8 [Schreier, 2005]).

2.1.3 The Oversampling benefit

When using a one-sided representation of the frequency domain, the *power spectral density* (PSD) of the quanization noise is:

$$S_e(f) = e_{rms}^2 \left(\frac{2}{f_s}\right) \tag{2.4}$$

Equation 2.4 implies that the quantization noise is uniformly distributed in the frequency range $0 < f < \frac{f_s}{2}$. The signal band however, might have a range from $0 < f < f_0$. The total in-band noise power is then calculated by using equation 2.2 and 2.4:

$$q_{rms}^2 = \int_0^{f_0} S_e(f)df = \frac{2f_0 e_{rms}^2}{f_s} = \frac{e_{rms}^2}{OSR} \tag{2.5}$$

Equation 2.5 shows for each doubling of OSR, the in-band noise power decreases by $3dB$ or 0.5 bits. Data converters employing oversampling to benefit from this property is called *oversampled* converters. By increasing the OSR they can achieve higher accuracy than a Nyquist converter using the same quantizer.

2.2 ΔΣ-modulators

Oversampled converters are also usually using ΔΣ-modulation to decrease the in-band noise power even further. The basics of the ΔΣ-modulation technique is presented in the following sections.

2.2.1 Noise Shaping

A general noise-shaped ΔΣ modulator and its linear model is shown in figure 2.3. It consists of a loop filter $H(z)$ and a quantizer. 1-bit quantizers are often used due to its inherent linearity (p.7 [Schreier, 2005]). Using the linear model, we can derive a *signal transfer function* (STF), $S_{TF}(z)$, and a *noise transfer function* (NTF), $N_{TF}(z)$. It is assumed that $u(n)$ and $e(n)$ are two independent inputs.

$$S_{TF}(z) = \frac{Y(z)}{U(z)} = \frac{H(z)}{1 + H(z)} \tag{2.6}$$

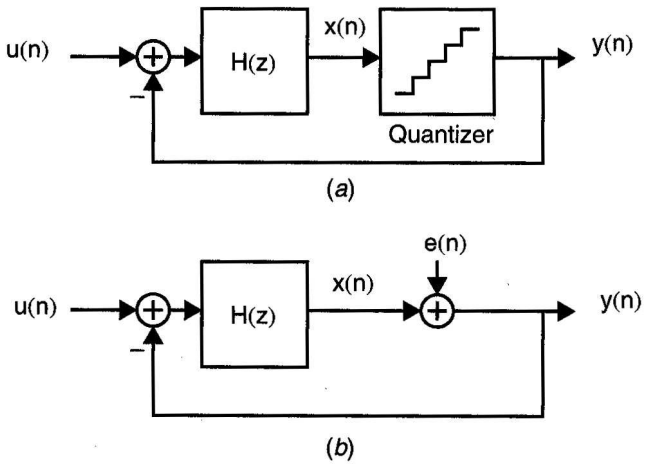

Figure 2.3: A general $\Delta\Sigma$-modulator and its linear model (fig. 14.6 [Johns, 1997])

$$N_{TF}(z) = \frac{Y(z)}{E(z)} = \frac{1}{1 + H(z)} \tag{2.7}$$

Combining these two equations gives a output function:

$$Y(z) = S_{TF}(z)U(z) + N_{TF}(z)E(z) \tag{2.8}$$

The zeros of $N_{TF}(z)$ will be equal to the poles of $H(z)$. This means that if $|H(z)| \gg 1$ then $|N_{TF}(z)| \ll 1$. To benefit from these noise shaping qualities, $H(z)$ is chosen such that its magnitude is large in the signal band (from 0 to f_0). This will give $|S_{TF}(z)| \approx 1$, and a $|N_{TF}(z)| \approx 0$ in the signal band. Thus, the quantization noise is reduced in the signal band while the signal itself is mostly unaffected.

First-order noise shaping

When realizing a first order noise shaping, $N_{TF}(z)$ should have its zero at DC, resulting in a high-pass filtering of the quantization noise. Since the zeros of $N_{TF}(z)$ are equal to the poles of $H(z)$, we choose $H(z)$ to be a discrete-time integrator which have this transfer function:

$$H(z) = \frac{1}{z - 1} \tag{2.9}$$

Combing equation 2.6, 2.7 and 2.9 gives a STF given by

$$S_{TF}(z) = \frac{Y(z)}{U(z)} = \frac{H(z)}{1 + H(z)} = \frac{\frac{1}{z-1}}{1 + \frac{1}{z-1}} = z^{-1} \tag{2.10}$$

and a NTF given by

$$N_{TF}(z) = \frac{Y(z)}{E(z)} = \frac{1}{1+H(z)} = \frac{1}{1+\frac{1}{z-1}} = 1 - z^{-1} \tag{2.11}$$

The equations show that the STF is a simple delay, while the NTF is a first order high-pass filter. In (p.542 [Johns, 1997]) it is shown that with such a first order modulator, each doubling of the OSR gives $9dB$ or 1.5 bits increase in the SNR. Compared to the $3dB$ increase with only oversampling, this shows the effect of noise shaping.

Higher-order noise shaping

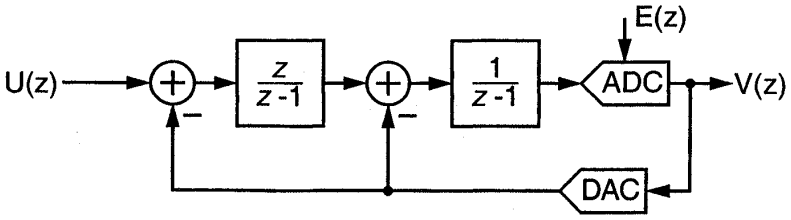

Figure 2.4: 2-order ΔΣ-modulator (fig. 1.6 [Schreier, 2005])

One way to further increase the resolution of the modulator, would be to use a higher-order loop filter. If another integrator and feedback path is added such as in figure 2.4, linearized analysis gives (eq. 1.8 [Schreier, 2005]):

$$V(z) = z^{-1}U(z) + (1 - z^{-1})^2 E(z) \tag{2.12}$$

The NTF is now a second-order high-pass filter. It is shown in (p.9 [Schreier, 2005]) that this gives $15db$ or 2.5 bits increase in resolution when doubling the OSR.

By adding more integrators and feedback branches to the loop, higher order NTFs can be obtained. For L^{th}-order loops resulting in $N_{TF}(z) = (1 - z^{-1})^L$, a function for approximation of the teoretical in-band noise power is (eq. 1.10 [Schreier, 2005]):

$$q_{rms}^2 = \frac{\pi^{2L} e_{rms}^2}{(2L+1)(OSR)^{2L+1}} \tag{2.13}$$

This function is plotted in figure 2.5

Stability considerations will reduce the practical achievable resolution of higher-order modulators. For higher-order single-bit modulators the difference is substantial. More than $60dB$ for a 5^{th}-order modulator (p.10 [Schreier, 2005]). These stability issues arise when the modulator order is higher than second-order.

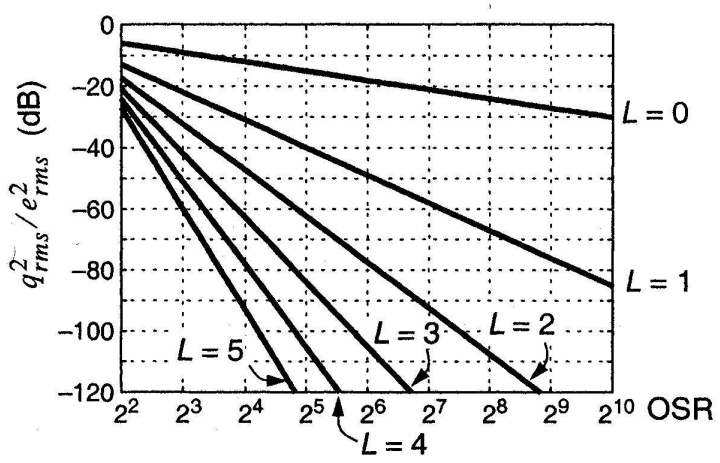

Figure 2.5: Theoretical in-band noise power for L^{th}-order $\Delta\Sigma$-modulators. (fig. 1.7 [Schreier, 2005])

2.3 Continuous-Time $\Delta\Sigma$-Modulators

The $\Delta\Sigma$-theory presented so far is based on purely *discrete-time* (DT) circuitry. The analog input $u(n)$ to the modulator is a sampled signal, and the loop filter $H(z)$ is made of DT, Z-domain filters. The majority of published $\Delta\Sigma$-modulators over the last decades have also been DT-realizations using *switched capacitor* (SC) circuitry. This is mostly because it is easy to map the mathematics of the modulators onto the implementation (p. *xxv* [Cherry, 2000]), and the well known implementations and high linearity of SC-filters (p. 39 [Ortmanns, 2006]). Due to the popularity of this approach, there is a great amount of literature and tools dealing with synthesis and implementation of DT $\Delta\Sigma$-modulators, e.g. [Schreier, 2005] and [Norsworthy, 1997].

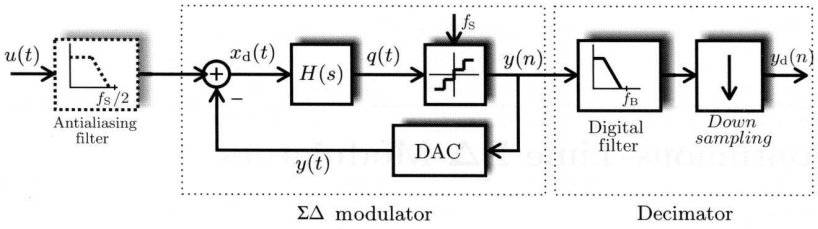

Figure 2.6: Block diagram of general CT ADC(fig. 3.1 [Ortmanns, 2006])

The first implementations of $\Delta\Sigma$-ADCs used *continuous-time* (CT) loop filters, but when the SC technique became popular in the 1980s, it was chosen in favour of its CT counterpart because of the properties already men-

tioned. A typical block diagram of a CT ΔΣ ADC is shown in figure 2.6. The input signal is fed through an optional AAF and into the CT-modulator. The loop filter is now made of CT-filters such as *active RC*-filters and *gmC*-filters. The sampling operation is finally done in the internal quantizer which is clocked at the sampling frequency f_s. In the following sections the key advantages an disadvantages between CT contra DT realizations is presented.

2.3.1 Sampling Operation

A key advantage of the CT modulator is that the sampling operation is done inside the ΔΣ loop (figure 2.6) which makes the nonidealities in the sampling process subject to noise-shaping. In the a DT modulator the sample operation is done with a *sample and hold* (S/H) circuit at the input of the converter, thus every error in this operation is directly added to the input signal (p.40 [Ortmanns, 2006]). In addition to the noise shaped sampling errors, moving in the sampling operation inside the loop also results in some degree of *implicit antialiasing filtering*. This heavily reduces the specification of the front-end AAF and sometimes makes it unnecessary. In high-speed circuits or applications with very low OSR, the *implicit antialiasing filtering* can be a key argument in choosing a CT implementation (p.40 [Ortmanns, 2006]). For an in-depth understanding of this property of CT modulators, chapter 3.5 in [Ortmanns, 2006] is recommended reading.

2.3.2 Loop Filter Realization

The loop filters in both DT and CT architectures are integrators or resonators depending on the prefered transfer function. In DT modulators the signals are quickly changing pulses, and the maximum clock rate of DT modulators with SC circuits is limited by the *operational transconductance amplifier* (OTA) bandwidth and the settling time for the charge transfers. In CT modulators all signals ar represented by analog, continuous-time waveforms, resulting in a theoretical clock rate that is in the order of a magnitude higher than its DT counterpart in the same technology. In real-life applications, this usually factor lies between three and five (p.41 [Ortmanns, 2006]). Another difference is the absoulute accuracy of the filter transfer functions. In SC integrators, the integrator gain is determined by a capacitor ratio, which can have an accuracy higher than 0.1%. In CT integrators, the integrator gain is set by a RC or gmC product which can vary with an amount of $\pm 50\%$. This results in performance degradation and in worst case unstable operation in CT modulators.

2.3.3 Quantizer realization

In both CT and DT implementations of $\Delta\Sigma$-modulators, all the nonidealities of the quantization are subject to noise shaping. This is due to the placement of the quantizer inside the modulator loop. The decision time of the quantizer is on the other hand more critical in CT modulators. In a DT modulator, the quantizer has half a clock cycle to determine its value. In a CT modulator the quantizer should ideally be infinitely fast since the result is needed instantly to generate the feedback signal. This could be a source to severe performance limitations if not taken care of.

2.3.4 Feedback realization

The feedback signal of a DT modulator using SC circuitry is made by charging a capacitor to a reference voltage and then discharging it onto the the integrating capacitance of the integrator. The feedback signal of a CT modulator is an continuous waveform integrated over time, which makes it sensitive to every deviation from an ideal one. This results in reduced performance and even instability in some situations. Most CT modulators use rectangular feedback pulses in a *non return to zero* (NRZ) or *return to zero* (RZ) configuration as seen in figure 2.7.

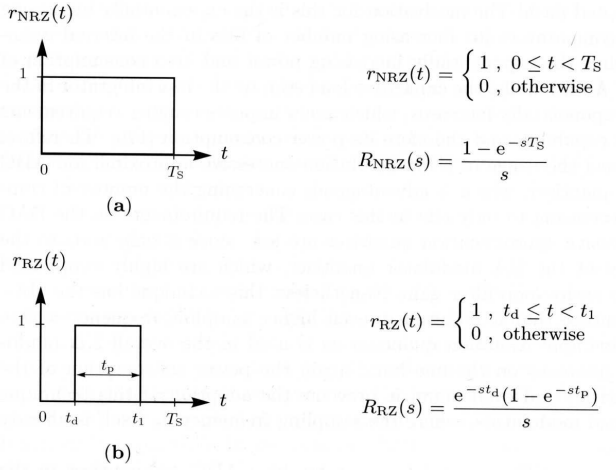

Figure 2.7: (a) NRZ DAC feedback pulse (b)RZ DAC feedback pulse (fig. 3.2 (a)-(b) [Ortmanns, 2006])

2.4 DT to CT Conversion

Most published work on $\Delta\Sigma$ modulators has been focused on DT implementations. A common way to design a CT loop filter is to start witch a DT

loop filter $H(z)$ with the desired specifications. Simulating the ideal modulator behaviour in the discrete-time domain is simple and very fast using tools like the *Delta Sigma Toolbox for Matlab* [$\Delta\Sigma$ Toolbox].

2.4.1 The impulse-invariant transform

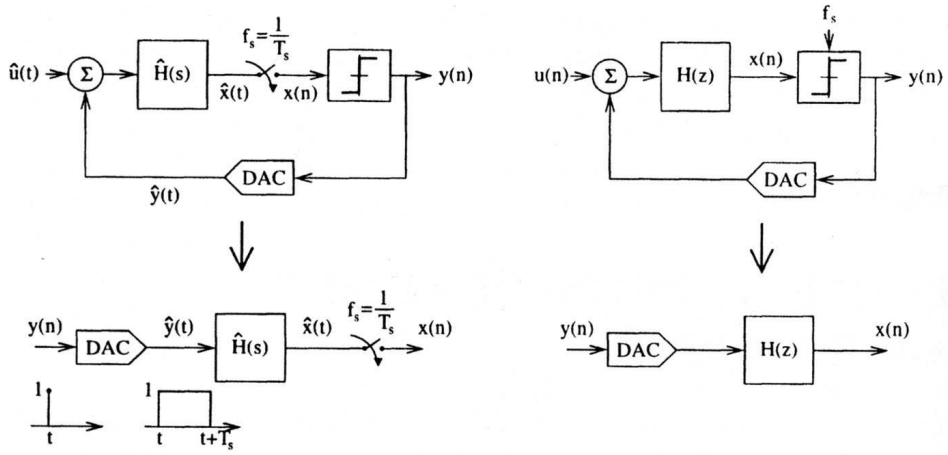

Figure 2.8: CT-DT equivalence(fig. 2.1 [Cherry, 2000])

For a succesful DT to CT modulator transformation, the equivalence between these two structures must be investigated. The clocked internal quantizer of the CT modulator makes the modulator to a kind of DT system at this point (p.48 [Ortmanns, 2006]. From figure 2.8 a DT to CT equvalence is achieved if the input to the quantizers $\hat{x}(t)$ and $x(n)$ are equal at the sampling instants. This means

$$x(n) = \hat{x}(t)|_{t=nT_s} \tag{2.14}$$

If this is fulfilled, the bitstreams out of both modulators is the same, and thus also the noise performance (p.48 [Ortmanns, 2006]). The condition in 2.14 is satisfied if the impulse responses of the open-loop diagrams in 2.8 is are equal at sampling times. This leads to the condition in the z-domain (p.31 [Cherry, 2000])

$$\mathcal{Z}^{-1}\{H(z)\} = \mathcal{L}^{-1}\{\hat{R}_D(s)\hat{H}(s)\}|_{t=nT_s} \tag{2.15}$$

or in the time domain

$$h(n) = [\hat{r}_D(t) * \hat{h}(t)]|_{t=nT_s} = \int_{-\infty}^{\infty} \hat{r}_D(\tau)\hat{h}(t-\tau)d\tau|_{t=nT_s} \tag{2.16}$$

where $\hat{r}_D(t)$ is the the impulse response of the feedback DAC. This transformation is called *the impulse-invariant transform* because we require the

open-loop impulse responses to be the same at the sampling instants. This transformation makes it possible to design a CT loop filter $H(s)$ that together with feedback DAC transfer function $\hat{r}_D(t)$ exactly matches the noise shaping behaviour of a DT loop filter $H(z)$ (p.31 [Cherry, 2000]).

To simplify the use of the transform, conversion tables are made for basic loop filter poles such as $1/(z-z_k)^i, i = 1, 2, 3...$ Table 2.1 covers first and second order filter poles.

Table 2.1: CT equivalents for rectangular feedback DAC pulses for first- and second-order DT low-pass loop filter poles. α and β are the timings for the rising and falling edge of the DAC pulse relative to the sampling period. (Tab. 3.2 [Ortmanns, 2006])

$\mathcal{Z}$-domain	$\mathcal{S}$-domain equivalents with f_s (Hz)$= 1/T_s$
$\frac{1}{z-1}$	$\frac{\omega_0}{s},\ \omega_0 = \frac{f_s}{\beta-\alpha}$
$\frac{1}{(z-1)^2}$	$\frac{\omega_1 s+\omega_0}{s},\ \omega_0 = \frac{f_s^2}{\beta-\alpha},\ \omega_1 = \frac{1}{2}\frac{f_s(\alpha+\beta-2)}{\beta-\alpha}$

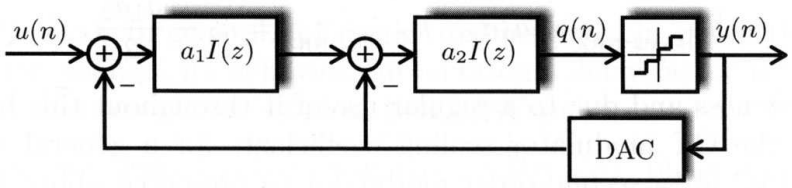

Figure 2.9: Block Diagram 2-order DT modulator(fig. 3.6 a) [Ortmanns, 2006])

Consider the second order DT modulator in figure 2.9 with the integrators $I(z) = z^{-1}/(1-z^{-1})$. The loop filter would then have the transfer function

$$LF(z) = -a_2 I(z) - a_1 a_2 I^2(z) = -\frac{a_2}{z-1} - \frac{a_1 a_2}{(z-1)^2} \tag{2.17}$$

By using rows 1 and 2 from table 2.1 we get

$$LF(s)|_{DT-CT} = -a_2 \frac{\frac{f_s}{\beta-\alpha}}{s} - a_1 a_2 \frac{\frac{1}{2}\frac{f_s(\alpha+\beta-2)}{\beta-\alpha} s + \frac{f_s^2}{\beta-\alpha}}{s^2} \tag{2.18}$$

The second order CT modulator in figure 2.10 with the integrators $Int_i =$

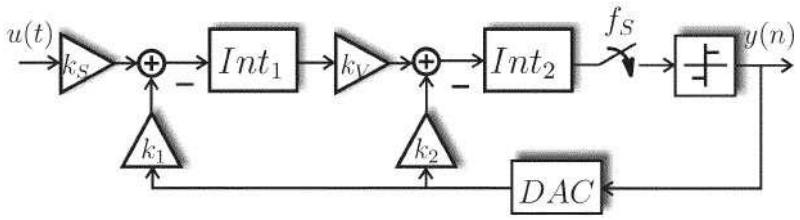

Figure 2.10: CT $\Delta\Sigma$ modulator (fig. 2(a) [Ortmanns, 2004])

$I(s) = \frac{f_s}{s}$ has the transfer function

$$LF(s) = -k_2 I(s) - k_1 k_v I^2(s) = -k_2 \frac{f_s}{s} - k_1 k_v \frac{f_s^2}{s^2} \tag{2.19}$$

Combining equation 2.19 and 2.18 yields these expressions for the scaling coefficients k_i:

$$\begin{aligned} k_{sig} &= a_1 a_2 \\ k_v &= 1 \\ k_1 &= \frac{a_1 a_2}{\beta - \alpha} \\ k_2 &= \frac{a_2(2 + a_1\alpha + a_1\beta - 2a_1)}{2(\beta - \alpha)} \end{aligned} \tag{2.20}$$

As an example one can use a NRZ DAC feedback pulse ($\alpha = 0, \beta = 1$) which gives the coefficients:

$$k_{sig} = k_1|_{NRZ} = a_1 a_2, \quad k_2|_{NRZ} = a_2 - \tfrac{a_1 a_2}{2} \tag{2.21}$$

By using the *optimal* values $a_1 = a_2 = 0.5$ for a second order DT modulator found in [Marques, 1998], the following CT feedback coefficents are obtained using 2.21

$$k_1|_{NRZ} = 0.5 \cdot 0.5 = 0.25, \quad k_2|_{NRZ} = 0.5 - \tfrac{0.5 \cdot 0.5}{2} = 0.375 \tag{2.22}$$

2.5 Gain Bandwidth Product

The *gain bandwidth product* (GBW) of an amplifier is defined as:

$$GBW = A \cdot \omega_{3dB}\ [rad/s], \tag{2.23}$$

where A is the dc-gain of the amplifier and ω_{3dB} is the $-3dB$ frequency. GBW is used to determine the maximum gain that can be extracted from an amplifer for a given frequency and vice versa. In common CMOS amplifiers

the correspondance between GBW and the transconductance parameter g_m is:

$$GBW = \frac{g_m}{C}, \tag{2.24}$$

where C is the load capacitance. Depending on the operating point and scaling of the transistors, the variation of g_m is usually proportional with the drain current I_d in practice. Hence by equation 2.24 it can be said that the amplifer GBW also is proportional to the current drain of the amplifier. In many cases lowering the GBW actually could give a larger current reduction than this due to the possibility to use more optimal transistor geometries since the transistor sizes are scaled down with the current.

Chapter 3

Circuit Design Considerations

The CT $\Delta\Sigma$ modulator is built of three major building blocks as in figure 2.6: a CT loop filter $H(s)$, a clocked internal quantizer and a CT feedback DAC. Due to variations in the manufacturing process and circuit imperfections, which comes from design (e.g. OTA gain) or are intrinsic (e.g. component mismatch), each of these blocks would deviate from ideal behavior.

3.1 Feedback DAC nonidealities

A major error contributor to CT $\Delta\Sigma$ modulators is the feedback DAC. Errors in the outermoust feedback loop adds directly to the input signal, which means that these errors are not subject to noise shaping (p.84 [Ortmanns, 2006]). If a typical rectangular feedback form is assumed (e.g NRZ and RZ), most common feedback DAC errors can be derived. The feedback pulse can be affected by timing errors which varies the position and length of the pulse. This can be a constant delay τ_d of the pulse often refered to as *excess loop delay*, or a statistical variation of the position or length of the DAC pulse caused by *clock jitter*. The pulse edges would also inevitably have nonideally rising and falling times due to finite slew rate in the DAC.

3.1.1 Excess Loop Delay

Excess loop delay is an unwanted delay t_d between the ideal and the implemented feedback DAC pulse:

$$t_d = \tau_d T_s, \tag{3.1}$$

where τ_d is the excess loop delay relative to the sampling period T_s. This delay can arise from finite respond time in the DAC and delays in the path between the quantizer and the DAC. The excess loop delay causes two different nonideal effects in CT $\Delta\Sigma$ modulators. When the loop delay shifts the DAC pulse, but the pulse retains within its sampling period as seen

in figure 3.1 or when the loop delay shifts parts of the DAC pulse in to the next sampling period as in figure 3.2. The latter happens when using a NRZ-DAC or even sometimes with RZ-DACs in high speed modulators (p.86 [Ortmanns, 2006]).

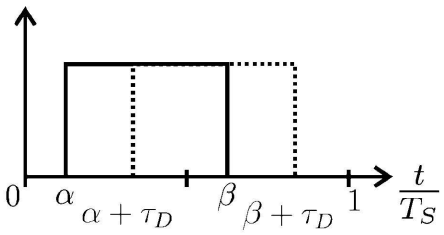

Figure 3.1: Excess loop delay in RZ DAC

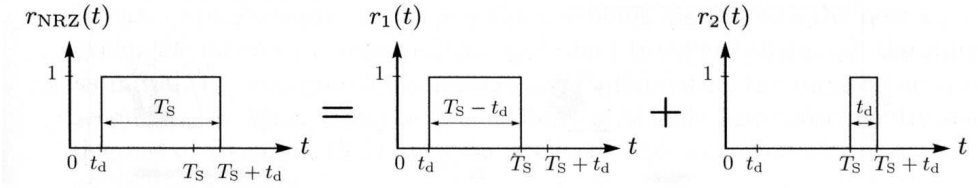

Figure 3.2: Excess loop delay in NRZ DAC (fig. 4.2 [Ortmanns, 2006])

Consider a second-order modulator as in figure 2.10 with an ideal NRZ DAC with pulse position $\alpha = 0, \beta = 1$, and an excess loop delay τ_d. The pulse position will then be $\alpha = \tau_d, \beta = 1 + \tau_d$. For mathematical purposes the it is useful to divide the pulse into two separate pulses. One placed in the original sampling period with position $\{\alpha_1 = \tau_d,\ \beta_1 = 1\ \}$ and another placed in the following sample period with position $\{\alpha_2 = 0, \beta_2 = \tau_d\}$ as shown in figure 3.2. Now it is possible to calculate the effects of this delay using table 3.1 to find the equivalent DT loop filter. ($f_s = 1$)

$$\frac{-k_1}{s^2} \rightarrow LF_{CT\rightarrow DT}|_1 = \frac{-k_1(\beta_1(2-\beta_1) - \alpha_1(2-\alpha_1))z - k_1(\beta_1^2 - \alpha_1^2)}{2(z-1)^2} + \frac{-k_1(\beta_2(2-\beta_2) - \alpha_2(2-\alpha_2))z - k_1(\beta_2^2 - \alpha_2^2)}{2(z-1)^2} \cdot z^{-1} \tag{3.2}$$

$$\frac{-k_2}{s} \rightarrow LF_{CT\rightarrow DT}|_2 = \frac{-k_2(\beta_1 - \alpha_1)}{z-1} + \frac{-k_2(\beta_2 - \alpha_2)}{z-1} \cdot z^{-1} \tag{3.3}$$

By combining these two equations and the values for α_i and β_i, the DT representation of the loop filter including the excess loop delay τ_d is given

as

$$\begin{aligned} LF(z,\tau_d) =& \frac{(-k_1\tau_d^2 + (2k_2+2k_1)\tau_d - 2k_2 - k_1)z^2}{2z(z-1)^2} \\ &+ \frac{(2k_1\tau_d^2 + (-k_1-4k_2)\tau_d + 2k_2 - k_1)z}{2z(z-1)^2} \\ &+ \frac{2k_2\tau_d - k_1\tau_d^2}{2z(z-1)^2} \end{aligned} \tag{3.4}$$

This transfer function contains terms to the power z^{-3} indictating that the second-order CT modulator has turned into a third-order one due to the excess loop delay. For $\tau_d = 0$ the expression in 3.4 is equal to a ideal second-order system. In figure 3.3 the effect of excess loop delay in a second-order modulator is illustrated. The poles is moving towards the unit circle with a growing τ_d, and finally exceeds the stability boundary. Thus excess loop delay may cause an unstable modulator.

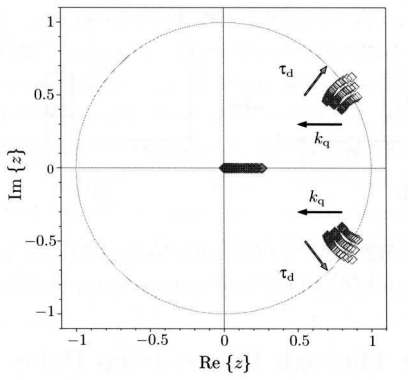

Figure 3.3: Pole plot of 2-order CT modulator (fig. 4.3 a) [Ortmanns, 2006])

3.1.2 Compensation for Excess Loop Delay

To compensate for the increased modulator order due to excess loop delay, a technique described in [Ortmanns, 2006] and [Cherry, 2000] can be used. As seen in figure 3.4, an auxiliary feedback DAC (DAC_{HRZ}) is added to the system. A half delay RZ DAC with $\{\alpha_h = 0.5, \beta_h = 1\}$ is a feasible implementation (p.89 [Ortmanns, 2006]). The concept for this compensation is to find the equivalent DT loop filter for the nonideal modulator in figure 3.4 and tune the coefficents to match the original ideal second-order loop filter. By using table 3.1 the loop filter of the first branch in figure 3.4 is found

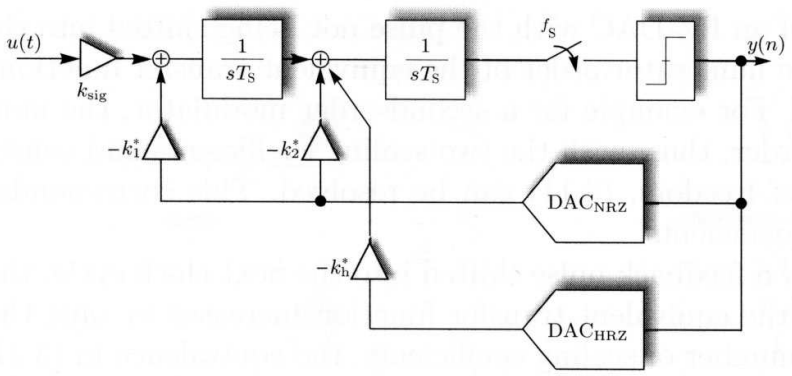

Figure 3.4: Compensation for excess loop delay in second-order modulator (fig. 4.4 [Ortmanns, 2006])

Table 3.1: DT equivalents for rectangular feedback DAC pulses for first- and second-order CT low-pass loop filter poles. (Tab. 3.6 [Ortmanns, 2006])

$\mathcal{S}$-domain	$\mathcal{Z}$-domain equivalents with f_s (Hz)$= 1/T_s$
$\frac{f_s}{s}$	$\frac{\omega_0}{z-1},\ \omega_0 = \beta - \alpha$
$\frac{f_s^2}{s^2}$	$\frac{\omega_1 z+\omega_0}{(z-1)^2},\ \omega_0 = \frac{\beta^2-\alpha^2}{2},\ \omega_1 = \frac{[\beta(1-\beta)-\alpha(1-\alpha)]}{2}$

(with NRZ DAC and $f_s = 1$):

$$\begin{aligned} LF_{CT}|_1 =& \frac{-k_1}{s^2} \rightarrow LF_{CT\rightarrow DT}|_1 \\ =& \frac{-k_1(\beta_{11}(2-\beta_{11}) - \alpha_{11}(2-\alpha_{11})z - k_1(\beta_{11}^2 - \alpha_{11}^2)}{2(z-1)^2} \\ &+ \frac{-k_1(\beta_{21}(2-\beta_{21}) - \alpha_{21}(2-\alpha_{21}))z - k_1(\beta_{21}^2 - \alpha_{21}^2)}{2(z-1)^2} \cdot z^{-1}, \end{aligned} \tag{3.5}$$

where $\{\alpha_{11} = \tau_d, \beta_{11} = 1\}$ and $\{\alpha_{21} = 0, \beta_{21} = \tau_d\}$. For the second branch a similar loop filter is found:

$$LF_{CT}|_2 = \frac{-k_2}{s} \rightarrow LF_{CT\rightarrow DT}|_2 = \frac{-k_2(\beta_{12} - \alpha_{12})}{z-1} + \frac{-k_2(\beta_{22} - \alpha_{22})}{z-1} \cdot z^{-1}, \tag{3.6}$$

where $\{\alpha_{12} = \tau_d, \beta_{12} = 1\}$ and $\{\alpha_{22} = 0, \beta_{22} = \tau_d\}$. For the new branch with the HRZ feedback DAC one obtains:

$$LF_{CT}|_h = \frac{k_h}{s} \rightarrow LF_{CT\rightarrow DT}|_h = \frac{k_h(\beta_{1h} - \alpha_{1h})}{z-1} + \frac{-k_h(\beta_{2h} - \alpha_{2h})}{z-1} \cdot z^{-1}, \tag{3.7}$$

where $\{\alpha_{1h} = 0.5 + \tau_d, \beta_{1h} = 1\}$ and $\{\alpha_{2h} = 0, \beta_{2h} = \tau_d\}$. These three loop filters form together the DT representation of the CT modulator in 3.4 with excess loop delay τ_d in the feedback path.

By setting $\tau_d = 0$ in the expression in 3.4, the DT equivalent transfer function for a ideal second order CT modulator is found:

$$LF_{2nd}|_{ideal}(z) = -\frac{1}{2}\frac{(k_1 + 2k_2)z}{(z-1)^2} - \frac{1}{2}\frac{k_1 - 2k_2}{2(z-1)^2} \tag{3.8}$$

The new feedback coefficients is found by solving the combined equation from 3.8 and 3.5-3.7:

$$\{LF_{CT}|_1 + LF_{CT}|_2 + LF_{CT}|_h\} = LF_{2nd}|_{ideal}(z) \tag{3.9}$$

which gives the following coefficients

$$k_1^* = k_1, \quad k_2^* = \tfrac{3k_1\tau_d}{2} + 2k_2, \quad k_h^* = k_1\tau_d + k_2 \tag{3.10}$$

where k_i^* are the coefficients of the compensated modulator in 3.4 and k_i are the coefficents of the ideal modulator. Thus with only on extra feedback DAC, the errors due to excess loop delay is ideally cancelled.

3.1.3 Clock Jitter

Clock jitter is statistical variations of the sampling frequency. When moving from the analog to the discrete-time domain in the quantizer, the continuous-time signal is ideally sampled with a constant time interval $T_s = \frac{1}{f_s}$. Circuit

non-idealites and the quality of the clock source will result in a deviation of the sampling interval. This will introduce errors in the system because the samples are taken at the wrong time. A CT modulator has two error sources due to clock jitter. First the sampled internal quantizer is susceptible to jitter affected sampling errors. The placement of the quantizer inside the modulator loop make these errors heavily supressed by noise shaping, and hence they may be neglected in practice (p.95 [Ortmanns, 2006]). The errors generated in the feedback DACs due to clock jitter is much more of concern in CT modulators. The feedback waveform is integrated over time, and a statistical variation of of this waveform results in a statistical integration error, which again gives a raised noise floor (p.95 [Ortmanns, 2006]).

For a NRZ feedback DAC pulse, equation 5.14 in [Cherry, 2000] calculates the SNR in the baseband for a modulator where the jitter noise is the dominating noise source:

$$SNR_{NRZ} = 10log_{10}\frac{OSR \cdot \frac{V_{in}^2}{2}}{\sigma_{\delta_y}^2(\frac{\sigma_\beta}{T_s})}\ dB, \tag{3.11}$$

where V_{in} is the input signal amplitude, σ_β/T_s is the jitter variance relative to the clock period and $\sigma_{\delta_y}^2$ is the variance of $\delta_y = y(n) - y(n-1)$ where $y(n)$ is the modulator output stream.

In a NRZ modulator, the jitter only matter when the output changes. I.e. the DAC pulse doesn't change if the output is stable. In a RZ modulator, both the rising and falling edge of the DAC pulse occur every clock cycle. This results in an increased number of transitions affected by the jitter, thus increased noise. Therefore a general rule of thumb is that a modulator with RZ DAC pulses has $6dB$ higher inband jitter noise than a modulator with NRZ DAC pulses (p. 116 [Cherry, 2000]).

3.1.4 Unequal DAC Pulse Rise/Fall Time

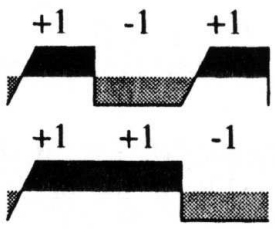

Figure 3.5: Effect of unequal DAC pulse rise and fall times (fig. 2.10 [Cherry, 2000])

CT $\Delta\Sigma$-modulators are in general sensitive to the exact shape of the DAC pulse. Any nonuniformities of the pulse tends to degrade the modulator performance. Consider a DAC output as in figure 3.5 where the rise

time is nonzero an the fall time is zero. As mentioned in the previous section the DAC pulses are integrated over time, and it is clearly seen in the figure that the area of the two pulse trains differ. In (eq. 2.3 [Cherry, 2000]) there is presented an expression regarding maximum allowable asymmetry contra SNR and OSR for a given sampling period T_s:

$$\tau \leq \frac{4 \cdot T_s \cdot \sqrt{OSR}}{SNR_{desired}}, \tag{3.12}$$

where the assymetry τ is measured as in figure 3.6.

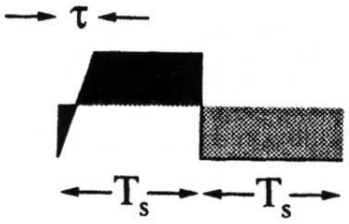

Figure 3.6: Asymmetric DAC pulse (fig. 2.11 [Cherry, 2000])

3.2 Filter Nonidealities

The transfer function of the loop filter defines the noise-transfer funtion, and thus the noise-shaping behaviour of the $\Delta\Sigma$-modulator. The loop filter usually consists of several first order filters, commonly arranged in a feedback or feedforward architecture (p. 117 [Ortmanns, 2006]). In CT modulators these single filters are realized using RC-integrators, gmC-integrators or LC-resonators. In the following active RC-integrators will be discussed.

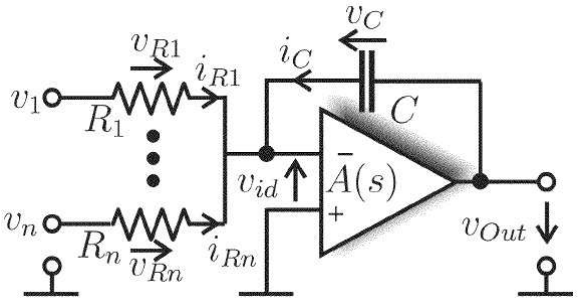

Figure 3.7: Active RC integrator (fig. 1 [Ortmanns, 2004])

Figure 3.7 shows a typical RC-integrator with n inputs and an amplifier with transfer function $A(s)$. The scaled integrator corner frequency ω_I

is controlled by the RC-product, R_iC. The following ideal relation can be found for the integrator corner frequency in CT modulators (eq. 5.1 [Ortmanns, 2006]):

$$\omega_{I_s}|_i = \frac{1}{R_iC} = |k_i f_s| \tag{3.13}$$

Where k_i is the CT scaling coefficients and f_s is the sampling frequency. The scaling coefficients can be found e.g. using a DT to CT conversion as described in section 2.4, and then mapped to different resistor values R_i.

The *integrator transfer function* (ITF) of the i^{th} input path to the integrator output with N different inputs is (eq. 5.2 [Ortmanns, 2006]):

$$ITF_i(s) = \frac{k_i f_s}{s(1+\frac{1}{A(s)}) + \frac{1}{A(s)}\sum_{l=1}^{N} k_l f_s}|_{A(s)\to\infty} \approx \frac{k_i f_s}{s}, \tag{3.14}$$

which equals an ideal integrator transfer function as long as the amplifier gain $A(s)$ is infinite.

3.2.1 Finite OTA gain

A well studied nonideal effect in $\Delta\Sigma$ modulators is the behaviour when the integrator amplifiers has finite dc-gain. Applying finite dc-gain $A(s) = A_{dc}$ to equation 3.14 yields

$$ITF_{i_{A_{dc}}}(s) = \frac{k_i f_s}{s(1+\frac{1}{A_{dc}}) + \frac{1}{A_{dc}}\sum_{l=1}^{N} k_l f_s} \approx \frac{k_i f_s}{s + \frac{1}{A_{dc}}\sum_{l=1}^{N} k_l f_s} \tag{3.15}$$

The ITF is now a first-order pole transfer function with a dc gain of $A_{dc}k_i/\sum_l k_l$ and a pole at $f_s\sum_l k_l/A_{dc}$. The pole is no longer placed at dc, which again effects the placement of the zeros in the NTF of the modulator (see equation 2.11). When all the zeros in the NTF moves away from dc, this reduces the amount of attenuation of the quantization noise in the baseband of the modulator. This nonideality is usually called *leaky integration* and affects both CT and DT modulators in the same way (p. 120 [Ortmanns, 2006]).

Equation 5.8 in [Ortmanns, 2006] gives an expression for the *integrated in-band noise* (IBN) of a general second order modulator with finite amplifier dc gain A_{dc}:

$$IBN_2(A_{dc}) \approx \frac{\Delta^2}{12}\frac{1}{k_{1|NRZ}^2 k_q^2}\left(\frac{\pi^4}{5}\frac{1}{OSR^5} + \frac{2\pi^2}{3}\frac{1}{OSR^3 A_{dc}^2} + \frac{1}{OSR\cdot A_{dc}^4}\right) \tag{3.16}$$

If the dc gain is in in the range of the OSR, e.g. $A_{dc} \approx OSR$, every part of equation 3.16 is proportional to $1/OSR^5$, the ideal noise shaping supression for a second order modulator (p. 120 [Ortmanns, 2006]).

3.2.2 Integrator Gain errors

Variations in the integrator gain due to nonideal scaling coefficients is also a nonideal effect that has to be considered when designing CT modulators. Integrator gain is mapped to resistor-capacitor products, e.g. $1/RC$, which can vary largely over process and temperature. The absolute component values can typical variate with 20% for capacitors and 30% for resistors, which gives a worst case RC variation of 56% [Project Meeting]. This surely makes this an error worth to investigate. Combining equation 3.13 and 3.14 and applying a tolerance δ_{RC} to the integrator scaling coefficient k_i gives an ITF:

$$ITF_{RC}(s)|_i = \frac{1}{sR_iC(1+\delta_{RC})} = \frac{f_s}{s}\frac{k_i}{(1+\delta_{RC})} = GE_{RC}\frac{k_i f_s}{s}, \qquad (3.17)$$

where GE_{RC} is the resulting equivalent gain error of the integrator. Equation 5.13 in [Ortmanns, 2006] gives an expression for IBN of single-loop N^{th}-order modulator with gain eror GE:

$$IBN_N(GE) \approx \frac{\pi^{2N}}{2N+1}\frac{\Delta^2}{12}\frac{1}{k_{1|NRZ}^2 k_q^2 OSR^{2N+1}}\prod_{i=1}^{N}\frac{1}{GE_i^2} \qquad (3.18)$$

Equation 3.18 show that the IBN is still proportional to $1/OSR^{2N+1}$, thus the modulator still shows N^{th}-order noise shaping.

3.2.3 Finite Amplifier Gain-Bandwidth Product

Finite amplifier *gain-bandwidth product* (GBW) introduces non-dominant poles in integrator transfer functions. Consider the amplifier in figure 3.7 modelled as a single pole amplifier, with a transfer function given by (eq. 1 [Ortmanns, 2004]:

$$A(s) = \frac{A_{dc}}{\frac{s}{\omega_A}+1}, \quad GBW = A_{dc}\omega_A \quad [rad/s], \qquad (3.19)$$

where ω_A is the dominant pole of the amplifier. Combining equation 3.14 and 3.19 gives an ITF for the i^{th} input path due to finite GBW in the amplifier considering a large dc gain in the amplifier (eq. 3 [Ortmanns, 2004]:

$$ITF_{GBW}(s)|_i \approx \frac{k_i f_s}{s}\frac{\frac{GBW}{GBW+\sum_l |k_l f_s|}}{\frac{s}{GBW+\sum_l |k_l f_s|}+1} \qquad (3.20)$$

This ITF is equivalent to an ideal scaled integrator as in equation 3.14 in series with a gain error and a additional second integrator pole. The gain error and second pole can be described as:

$$GE_{GBW} = \frac{GBW}{GBW+\sum_l \omega_I|_l}, \quad \omega_p = GBW + \sum_l \omega_I|_l, \qquad (3.21)$$

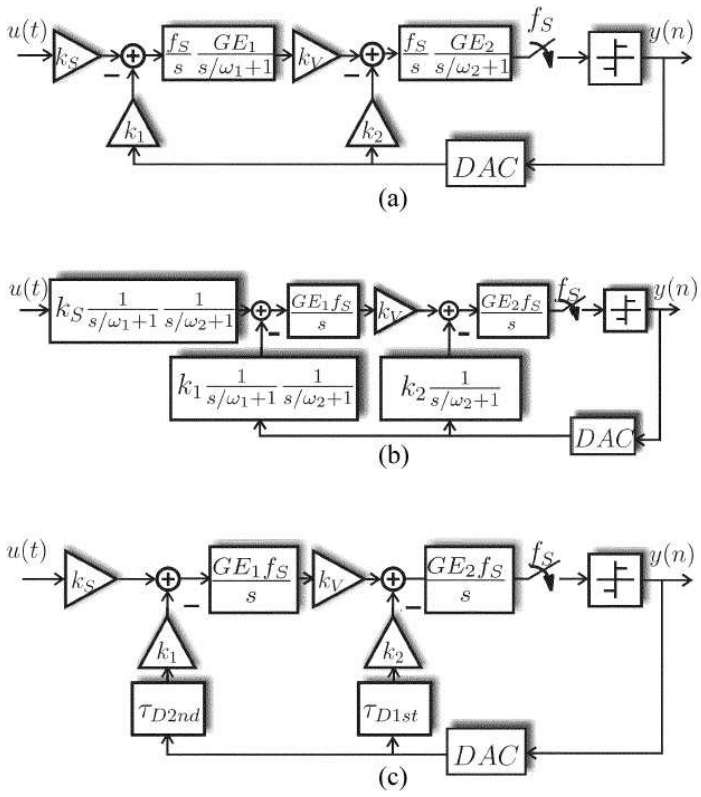

Figure 3.8: (a) Integrators with gain error and amplifier pole. (b) Modified block diagram. (c) Complete model with delays and gain errors. (fig. 2 [Ortmanns, 2004])

where $\omega_I|_l$ is the corner frequency of the l^{th} integrator. This results in a new expression for the ITF:

$$ITF_{GBW}(s)|_i \approx \frac{k_i f_s}{s} \frac{GE_{GBW}}{\frac{s}{\omega_p}+1} \tag{3.22}$$

When neglecting the influence from the second integrator pole, the errors are similar to the gain-errors from RC-variations (sect. 3.2.2). For very low GBWs this simplification fails, thus the influence from the second integrator pole has to be investigated. To simplify the following, a variable c is defined as the ratio between f_s and GBW:

$$c = \frac{GBW}{2\pi f_s} \tag{3.23}$$

Figure 3.8 a) show a second order CT modulator with ITFs as in equation 3.22. The nonideal poles in the ITFs can be moved behind the summing nodes to achieve the modified system in figure 3.8 b). Now the feedback transfer functions are scaled single or double pole systems. Because

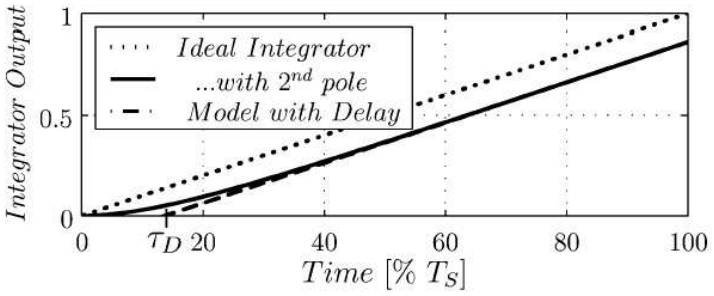

Figure 3.9: Non-ideal and ideal integrator step response (fig. 3 [Ortmanns, 2004])

of the rectangular feedback pulse, the combination of the feedback transfer functions with the integrators in the forward path can be modeled as non-ideal integrators with delayed output slopes (p. 1091 [Ortmanns, 2004]). An example of this is seen in figure 3.9, where the ideal and nonideal step response of an integrator is plotted. The final model for a modulator with finite GBW is shown in figure 3.8 c). The extra integrator poles are now reduced to the feedback delays τ_{D2nd} and τ_{D1st}. It should also be mentioned that the input transfer function is not affected by the poles ω_{p_i} as long as $\omega_{p_i} \gg f_{sig}$, where f_{sig} is the input signal frequency (p. 1089 [Ortmanns, 2004].

In (p. 1090 [Ortmanns, 2004]) expressions for the feedback delays τ_{D_i} have been calculated:

$$\tau_{D1st} = \frac{1 - e^{-\omega_{p2}/f_s}}{\omega_{p2}}, \tag{3.24}$$

$$\tau_{D2nd} = \frac{w_{p1}^2(1 - e^{-\omega_{p2}/f_s}) - w_{p2}^2(1 - e^{-\omega_1/f_s})}{\omega_{p1}\omega_{p2}(\omega_{p1} - \omega_{p2})}, \tag{3.25}$$

where ω_{p_i} are the poles calculated from eq. 3.21 for the different integrators.

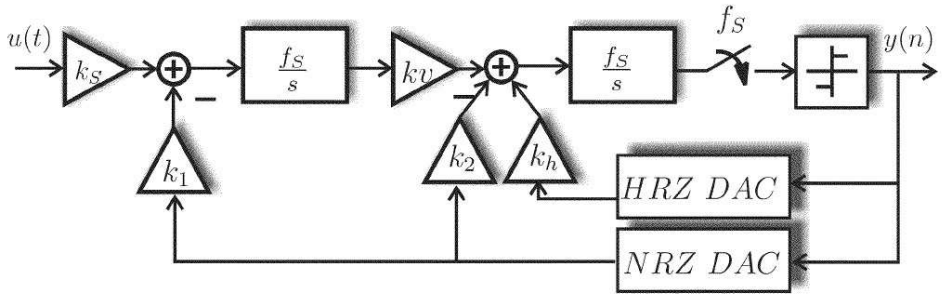

Figure 3.10: Compensation for GBW induced feedback delay in second-order modulator (fig. 5 [Ortmanns, 2004])

3.3 Motivation for using amplifers with low GBW

By using amplifiers with $c \geq 1$, the errors from finite GBW in CT modulators usually is negligible. As indicated in section 2.5, the current draw from an amplifier is approximately proportional with its GBW. In $\Delta\Sigma$ modulators, the design of the first integrator is crucial to the overall performance due to its placement in the noise shaping loop. In a reported 3-order CT modulator (sec. 7.3 [Ortmanns, 2006]), 55% of the overall power budget is consumed in the first integrator, 27% in the second and third modulator, 10% for the bias and 8% for the digital part. This means that 82% of the total power is consumed in the integrators, where the amplifiers usually is the most power hungry part. Thus reducing GBW with a factor of 10-20 would have a significant inpact on the overall power consumption.

3.4 Compensation for finite amplifier GBW errors

It has been shown that finite amplifier GBW in the integrators can be modeled as a gain-error and an extra loop delay in the feedback branches. In section 3.1.2 a compensation technique for excess loop delay was presented, and this can also be used to compensate for the extra loop delay induced by finite amplifier GBW. Figure 3.10 shows a second-order modulator with GBW induced feedback delays and an extra HRZ-feedback DAC for compensation.

As for the excess feedback delay compensation in section 3.1.2, the non-ideal CT modulator can be transformed into a DT equivalent and matched with the ideal DT modulator to find the new feedback coefficients. The expressions for the DT equivalents for the CT loopfilter branches are equal to the ones found in equations 3.5-3.7, but the α and β coefficients are changed due to having different delays in the branches. Table 3.2 shows the new α and β values.

Table 3.2: α and β coefficient mapping with finite amplifer GBW compensation

Coefficient	Excess loop delay	GBW induced delay
α_{11}	τ_d	τ_{D2nd}
α_{21}	0	0
β_{11}	1	1
β_{21}	τ_d	τ_{D2nd}
α_{12}	τ_d	τ_{D1st}
α_{22}	0	0
β_{12}	1	1
β_{22}	τ_d	τ_{D1st}
α_{1h}	$0.5 + \tau_d$	$0.5 + \tau_{D1st}$
α_{2h}	0	0
β_{1h}	1	1
β_{2h}	τ_d	τ_{D1st}

In (eq. 19 [Ortmanns, 2004] the following expressions for the scaling coefficients k_i where found considering DT scaling $a_1 = a_2 = 0.5$ (the *optimal* scaling values from [Marques, 1998]):

$$\begin{aligned} k_{sig} &= k_1 = \frac{1}{4} \\ k_v &= 1 \\ k_2 &= \frac{6\tau_{D1st} + 4\tau_{D1st}\tau_{D2nd} - \tau_{D2nd}^2}{8\tau_{D1st}} \\ k_h &= \frac{3\tau_{D1st} + 2\tau_{D1st}\tau_{D2nd} - \tau_{D2nd}^2}{4\tau_{D1st}} \end{aligned} \tag{3.26}$$

Since the scaling coefficients in equation 3.26 are dependent of the calculated feedback-delays, and the delays again are dependent on the scaling coefficients, an iterative calculation procedure must be done. In (p. 137 [Ortmanns, 2006]) such a calculation procedure also involving compensation for the GBW induced gain-error is presented:

1. Start the calculation process with the scaling coefficients in eq. 2.20 and $k_h = 0$. Calculate the gain-error GE_1 with eq. 3.21 and correct for it through $k_v = 1/GE_1$

2. Calculate the resulting GBW-induced delays with eq. 3.21 and 3.24.

3. Use eq. 3.26 to assign a set of compensation coefficients.

4. Repeat steps 2 and 3 up to a certain accuracy is achieved.

5. The gain error GE_2 is (ideally) compensated for by the use of single-bit quantizer.

In [Ortmanns, 2004] it is shown with simulations that this compensation ideally works for very low GBW-values. Even with $c < 0.02$ the modulator works as intended. It is important to remember that this compensation technique require amplifiers with *1-pole* transfer characteristics similar to what presented in equation 3.19.

3.5 Quantizer Nonidealities

Since the quantizer is located in the modulator loop at the place with the highest noise supression, most nonidealities in quantizer has little influence on the modulator performance. E.g. offset, hysteresis and nonlinearities (p.155 [Ortmanns, 2006]). Timing induced errors like propagation and signal dependent delay however, has a more severe inpact on the overall performance. Propagation delay will have the same influence as excess loop delays in the DACs, which was discussed in section 3.1.1. Even worse is signal dependent delay due to quantizer metastability. When the delay varies with the input signal amlitude, also the length of the feedback DAC pulses will be signal dependent. This is known as signal dependent timing jitter (p. 156 [Ortmanns, 2006]). However, this signal dependency can be circumvented by inserting a latch between the quantizer and the feedback DACs, giving the quantizer time to settle before its output is latched (p. 1348 [Gerfers, 2003]. If these timing issues are accounted for, the quantizer performance is of minor importance in CT $\Delta\Sigma$ modulators, compared to the nonidealities of the feedback DAC or the loop filter (p. 156 [Ortmanns, 2006]).

Chapter 4

System Requirements

4.1 System Specification

The ADC spesification is as follows.

- 50 *kHz* signal bandwidth
- 10 effective number of bits accuracy (SNDR[1] $\geq 62dB$)
- Using continuous-time $\Delta\Sigma$ architecture

The ADC is also meant to operate in a microcontroller with a very tight power budget Thus there are also a few other requirements that must be accounted for:

- Low voltage operation, preferably $\leq 1.5V$
- Low power

4.2 Modulator Architecture

When looking for different CT modulator architectures suitable for very low power consumption and the required signal bandwidth, an article [Ortmanns, 2004] describing the finite GBW compensation technique in section 3.2.3 and 3.4 was found. Simulations performed in this article showed that for GBWs as low as $c = 0.15$ (see eq. 3.23 for definition of c), the modulator would ideally work as intended. The article included simulations applying nonidealites such as GBW variantion and RC-product variation which also showed the robustness of this compensation approach. The simulations included both second and third order modulators, but the latter had stability problems when subject to the nonidealities. A second order modulator architecture as in figure 2.10 was therefore chosen as the basis for this work, because of

[1] signal to noise plus distortion ratio

its simplicity and the promising results applying GBW compensation with such modulators in [Ortmanns, 2004].

4.3 Simulation Objectives

4.3.1 Feedback coefficients and OSR

The first objective is to find a set of feedback coefficients and OSR to meet the accuracy specification. This should be done in the DT domain using the *Delta sigma toolbox for Matlab* [$\Delta\Sigma$ Toolbox].

4.3.2 Robustness of the compensated CT $\Delta\Sigma$ Modulator

The second order modulator should be simulated with and without the GBW compensaton. The influence from nonidealities like finite OTA gain, finite OTA GBW, RC-product variation, excess loop delay and finite DAC slew rate should be investigated for both models to see how the compensation affects the stability of the modulator. An important aspect is to see how the compensated modulator handles these nonidealites when the amplifier GBW deviates from the value compensated for. This is not covered in [Ortmanns, 2004].

Chapter 5

Model Descriptions

The model used to simulate the system is presented in this chapter. It is written in *VHDL-AMS* [VHDL-AMS], an extension of the discrete event based *VHDL* language which also supports the description and the simulation of analog, and mixed-signal circuits and systems. The simulations are done with the *Advance MS* suite from *Mentor Graphics* [Advance MS].

5.1 Building Blocks

5.1.1 Resistor Model

The resistors are modelled as a two-terminal device with the following equation:

$$U = I \cdot R, \tag{5.1}$$

where U is the voltage between the terminals, I is the current flowing through the resistor and R is the resistance. The source code for this model is found in appendix B.1.

5.1.2 Capacitor Model

The capacitors are modelled as a two-terminal device with the following equation:

$$I = C \cdot \frac{dV}{dt}, \quad V(0) = V_0, \tag{5.2}$$

where I is the current through the capacitor, C is the capacitance, V is the voltage between the terminals and V_0 is the initial value of V. The source code for this model is found in appendix B.2.

5.1.3 OTA model

The OTAs are modeled as fully differential single pole amplifiers based on the transfer function from equation 3.19

$$A(s) = (eq.3.19) = \frac{A_{dc}}{\frac{s}{\omega_A}+1}, \qquad GBW = A_{dc}\omega_A \ [rad/s], \tag{5.3}$$

$$\Delta_{V_{in}} = V_{in+} - V_{in-}, \tag{5.4}$$

$$V_{out+} = \frac{\Delta_{V_{in}}}{2} \cdot A(s), \tag{5.5}$$

$$V_{out-} = -\frac{\Delta_{V_{in}}}{2} \cdot A(s), \tag{5.6}$$

where GBW is the *gain bandwidth product* of the OTA and A_{dc} is the dc-gain. The source code for this model is found in appendix B.3.

5.1.4 Integrator Model

The integrators are modeled as fully differential *active RC* integrators similar to the single ended version discussed in section 3.2. It is a structural model put together from the resistor, capacitor and OTA models described in the previous subsections. Figure 5.1 shows the structure of the integrators. The integrator structures are embedded into the complete modulator

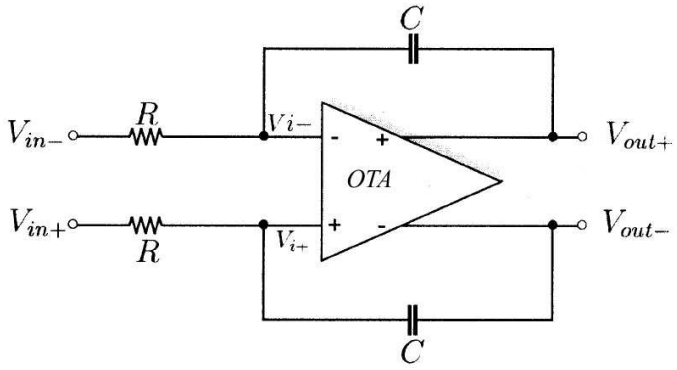

Figure 5.1: Fully differential active RC integrator

models in section 5.2 and the source code for these can be found in appendix B.7 and B.8.

5.1.5 NRZ DAC Model

The NRZ DAC is modeled as a clocked voltage switch which will switch between its two reference voltages if the DAC input has changed before a rising clock edge. It also incorporates delayed switching and different slew rates for rising and falling edges. The source code for this model is found in appendix B.4.

5.1.6 HRZ DAC Model

The HRZ DAC is modeled similar to the NRZ DAC model, but differs somewhat since it is zeroed at every rising clock edge and switched on at every falling edge. The *OFF* functionality is done by forcing the current through the switch to zero. Also this DAC model incorporates delayed switching and different slew rates for rising and falling edges. The source code for this model is found in appendix B.5.

5.1.7 Quantizer Model

The quantizer model is made as a clocked comparator with digital output. The output changes if the differential input has crossed a voltage treshold level before a rising clock edge. The source code for this model is found in appendix B.6.

5.2 Modulator Models

5.2.1 2-order CT $\Delta\Sigma$ modulator with active-RC integrators

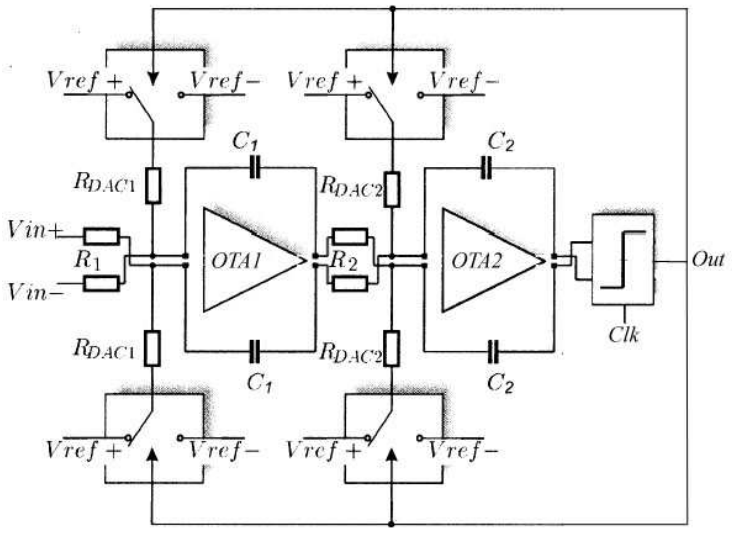

Figure 5.2: 2-order CT $\Delta\Sigma$ modulator with active-RC integrators

The model for the 2-order CT-modulator is a structural model made of the components described in the previous section. Figure 5.2 is a schematic of how the modulator is put together. It consists of two fully differential

integrators, a quantizer and four NRZ DAC elements with corresponding resistors R_{DAC_i}.

To avoid timing issues, the DAC clock (not shown in the figure) is a delayed version of the quantizer clock. Thus the model also includes a simple delayed clock generator. The source code for this model can be found in appendix B.7.

5.2.2 2-order GBW compensated CT $\Delta\Sigma$ modulator with active-RC integrators

The model for the 2-order GBW compansated CT-modulator is almost identical to the modulator without compensation. In addition to the components in figure 5.2, two HRZ DAC elements with corresponding resistors R_{HRZ} has been added to the system. This is done according to the block diagram in figure 3.10, i.e. they are connected to the two inputs of OTA2 in figure 5.2. The source code for this model can be found in appendix B.8.

5.3 SNDR calculation

The output from the CT modulators is a serial stream of bits. To plot the output frequency spectrum and calculate the SNDR, this bitstream needs postprocessing. The [Octave] script in appendix D.1 reads the output bitstream from the VHDL testbench in appendix B.9, calculates the SNDR according to the method presented in (Appendix A.4 [Schreier, 2005]), and finally writes the calculated output noise spectrum and integrated noise spectrum to a *.csv*-file which can be read by a waveform viewer.

Chapter 6

Simulation Results and Discussion

6.1 Scaling coefficients and OSR

In [Marques, 1998] they found the *optimal*[1] scaling values for a second-order DT modulator to be $a_1 = a_2 = 0.5$. A second-order DT loop filter with these scaling coefficients is used as the basis for the CT modulators in this text. By using the impulse invariant transform from section 2.4.1 the following CT scaling coefficients were found (NRZ feedback pulse):

$$k_{sig} = k_1 = 0.25, \quad k_v = 1 \quad k_2 = 0.375$$

By using the [Matlab] script in appendix C.2 simulations were done to find an appropriate OSR and to verify the DT-CT transform (do a CT-DT transform). Figure 6.1 shows the simulated SNR vs. input amplitude for an OSR of 32, 48 and 64 with the [Matlab] model. The SNR peaks at $55db$ for OSR=32, $65db$ for OSR=48 and $71.5db$ for OSR=64. The DT CT-DT transformed loop filter turned out to be identical with the original DT loop filter, they had identical transfer functions. The specification (sec. 4.1) claims $10\ ENOB$ or $62dB$ SNDR, which means only OSR=64 will give enough margins considering other noise sources and nonidealities than quantization noise. In the following simulations an OSR=64 will be used.

[1] Noise shaping ability versus stability

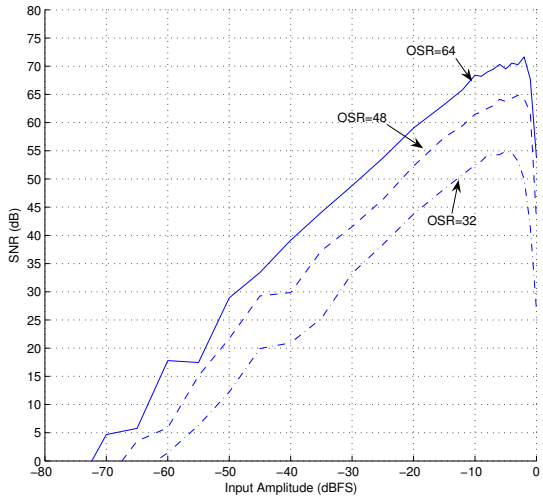

Figure 6.1: SNR vs Input amplitude for different OSR

6.2 Ideal 2-order CT modulator perfomance

When using OSR=64 and the specification claims $50kHz$ signal bandwidth, the needed sampling rate is given by equation 2.2, $f_s = 64 \cdot 50kHz \cdot 2 = 6.4MHz$. This is the sampling rate used in the further simulations. The resistor and capacitor values[2] used are:

$$C = 15.625pF$$
$$R = \frac{10k\Omega}{k_i}$$
$$\frac{1}{RC}|_{k_i=1} = 6.4 \cdot 10^6 = f_s,$$

where k_i is the scaling coefficient. E.g. when the scaling coefficient k_1 is 0.25, the corresponding resistor R_1 will be $40k\Omega$.

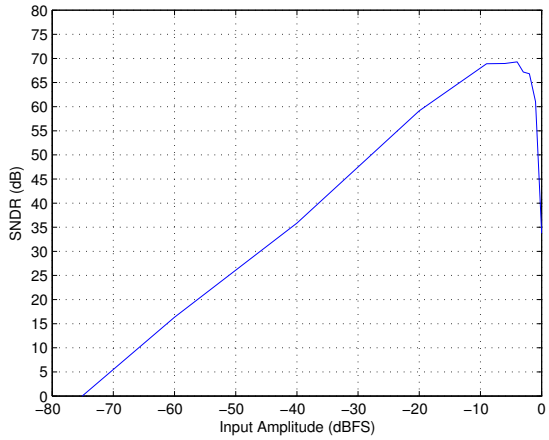

Figure 6.2: SNDR vs Input amplitude for 2-order CT modulator. $f_s = 6.4MHz$, $f_{in} = 9kHz$

Figure 6.2 shows the SNDR vs. input amplitude for the 2-order CT modulator [VHDL-AMS] model. The SNDR peaks at $69.3dB$ for an input amplitude of $-4dBFS$. All following simulations are done in [Advance MS] with the [VHDL-AMS] models.

6.3 Finite amplifer GBW performance

Figure 6.3 shows how finite amplifier GBW in the integrators affect the performance of the CT modulator. Without compensation, the SNDR suffer already when $c \leq 2$. The compensated modulator works all the way

[2]The capacitor and resistor values are selected to match f_s, while still having resonably geometries

down to $c = 0.02$ with no visible performance loss. Figure 6.4 shows the frequency spectrums for the compensated modulator with different compensation values. There is no sign of degradation of the signal in the frequency domain due to the compensation.

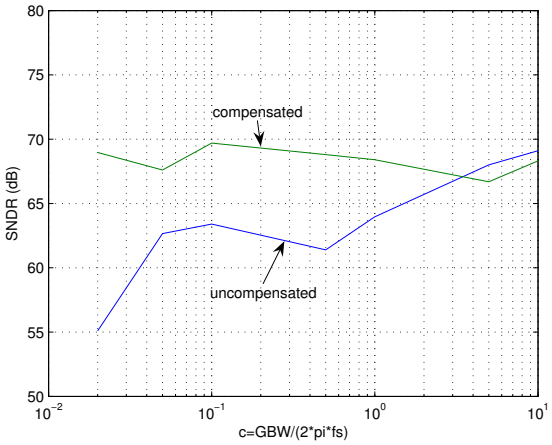

Figure 6.3: SNDR vs GBW for 2-order CT modulator and ideally compensated modulator. $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

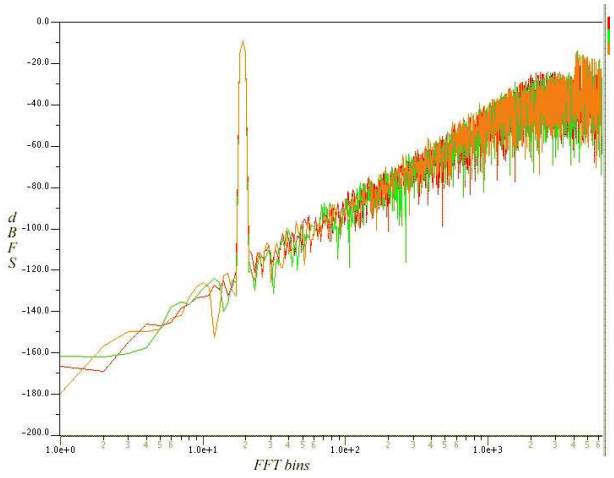

Figure 6.4: Frequency spectrum for compensated CT modulators with $c = 10$ (red), $c = 0.1$ (green), $c = 0.05$ (orange). $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

6.4 Robustness of GBW compensation

In the following sections when refered to simulations with the noncompensated model, these simulations are done with a GBW ten times the sampling frequency. (c=10)

6.4.1 Variations in amplifier GBW

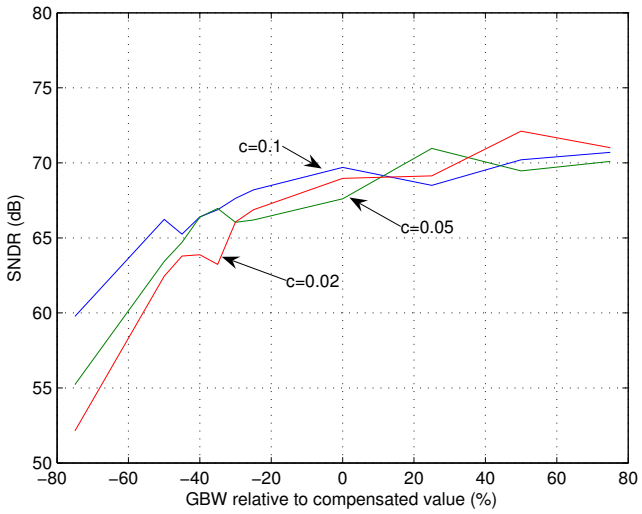

Figure 6.5: SNDR when GBW deviates from compensated value. $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

The compensated modulator performed very well considering ideal compensation, i.e. the GBW compensated for matches the actual GBW in the amplifiers. Figure 6.5 shows how the compensated modulators are affected by variation in the actual amlifier GBW. For positive variations the performance is mostly unaffected, but for negative variations the SNDR suffers. For lower c-values the SNDR rolls of earlier. For $c = 0.1$ and $c = 0.05$, the degradation is acceptable down to approximately -40% deviaton from compensated amplifier GBW. In the further simulations a GBW variation of $\pm 40\%$ is used.

6.4.2 Finite OTA gain

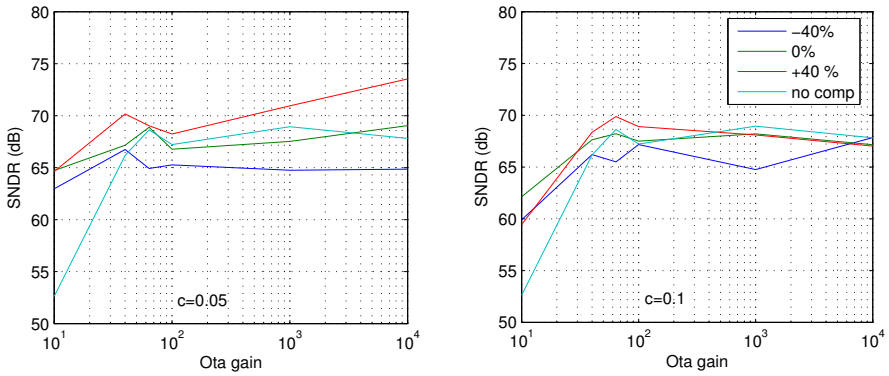

Figure 6.6: SNDR for different values of OTA gain and $\pm 40\%$ variation of GBW. $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

In figure 6.6 the influence from finite OTA gain is presented. The compensated modulator has been simulated with $\pm 40\%$ variation of the amplifier GBW. In section 3.2.1 it was stated that the OTA dc-gain should be at least $\geq$OSR to avoid performance degradation. This can also be seen in the figure where the SNDR falls when the gain is to low. The compensated modulator does not seem to be more affected by finite OTA gain than the noncompensated modulator. In fact, in figure 6.6 it keeps the SNDR better for low gain values than the noncompensated modulator.

6.4.3 RC product variation

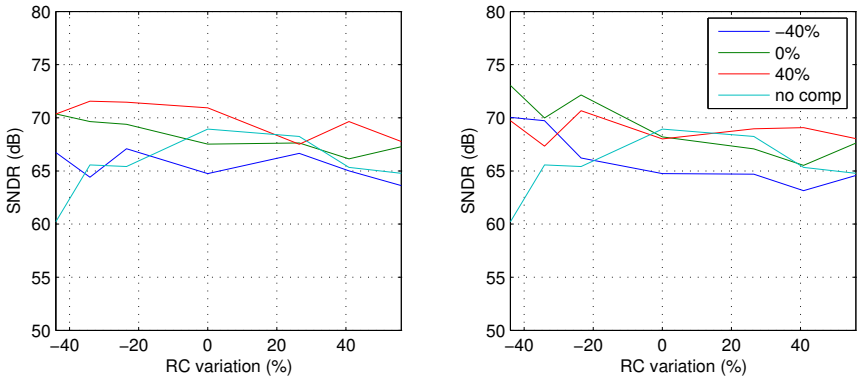

Figure 6.7: SNDR vs RC product deviation with $\pm 40\%$ variation of GBW. $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

In figure 6.7 the influence from RC product variation in the integrators is presented. In section 3.2.2 it was stated that the resistor value can vary with $\pm 30\%$ and the capacitors with $\pm 20\%$. This gives a RC-product variation from -44% to $+56\%$ which also is used in the simulations. The resistor and capacitor values are adjusted with an equal amount relative to their maximum deviation. I.e. if the capactior changes $+10\%$, the resistor changes $+15\%$. The figure shows that the susceptibility to RC-variations for the compensated modulator is comparable to the modulator without compensation. For the most extreme negative RC-product variation, it even seems to perform better.

6.4.4 Excess Loop Delay

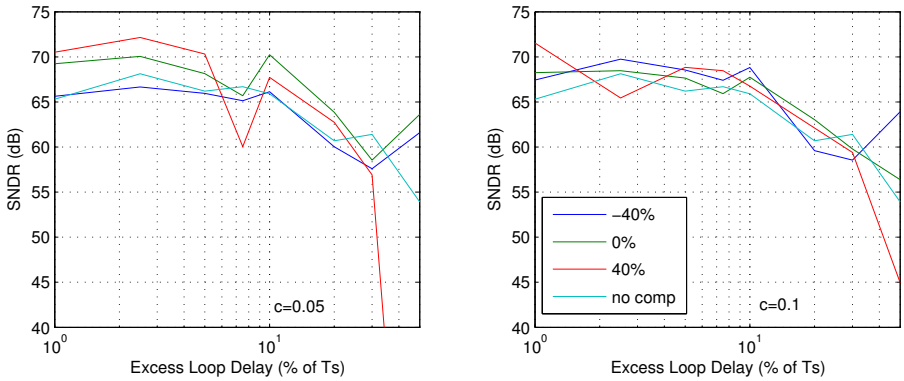

Figure 6.8: SNDR vs Excess Loop Delay with $\pm 40\%$ variation of GBW. $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

In figure 6.8 the influence from excess loop delay in the feedback loops is presented. The modulator with compensation for $c = 0.05$ has a significant perfomance drop for a loop delay $\tau_d = 7.5\%$ of T_s, and becomes unstable for the highest delay values. With $c = 0.1$ there is no significant difference in performance compared to the modulator without compensation. A τ_d of 7.5% gives a maximum tolerable delay $t_d = 0.075 \cdot 156.25\ ns = 11.7\ ns$ when $c = 0.05$.

6.4.5 DAC slew rate

Finite Slew Rate

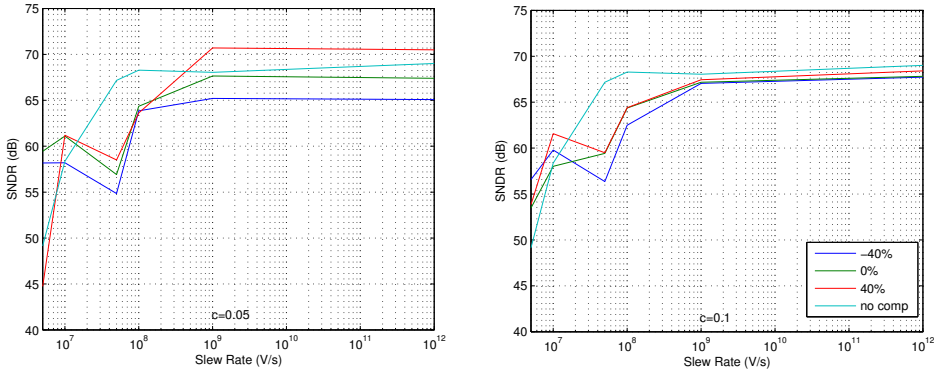

Figure 6.9: SNDR vs DAC Slew Rate with $\pm 40\%$ variation of GBW. $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

In figure the 6.9 influence from finite DAC slew rate is presented. The SNDR starts to decrease for the compensated modulators when the slew rate is lower than $10^8\ V/s$, but the noncompensated modulator is mostly unaffected down to $10^7\ V/s$.

Asymmetric Slew Rate

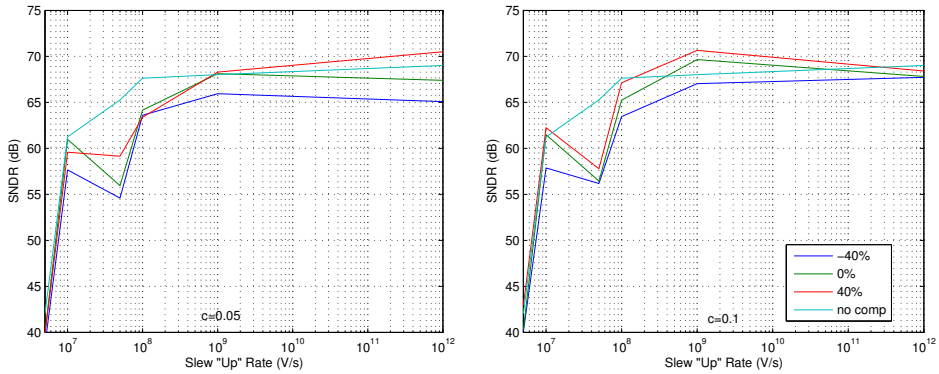

Figure 6.10: SNDR vs DAC Slew "Up" Rate with $\pm 40\%$ variation of GBW and Slew "Down" Rate $= 10^{12}\ V/s$. $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

In figure the 6.10 the influence from asymmetric DAC pulses due to different rise and fall times in the DACs are presented. The DAC slew "down" rate was set to $10^{12}\ V/s$ and the slew "up" rate varies.. The SNDR in the

compensated modulator falls of at $10^8\ V/s$ a decade before the noncompensated one. This might be a result of the low slew rate and not the asymmetry tough, since the results are almost equivalent to what seen for the finite slew rate simulations in figure 6.9.

6.5 Internal Signal Swing

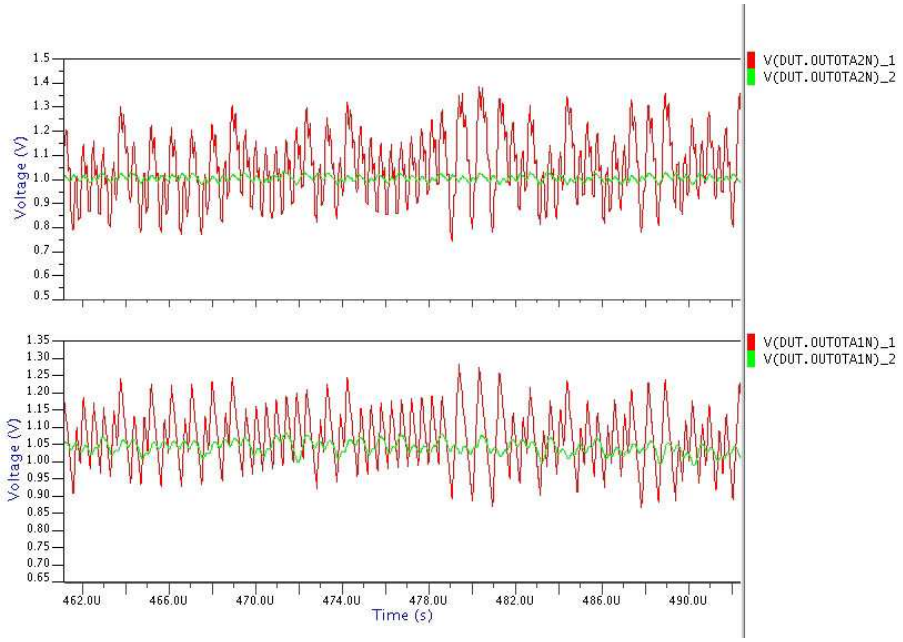

Figure 6.11: Signal swing at integrator ouputs for the second and first integrator in a compensated modulator when $GBW_{red} = 10c$ and $GBW_{green} = 0.05c$. $f_s = 6.4MHz$, $f_{in} = 9kHz$, $|V_{in}| = -9dBFS$

Figure 6.11 shows the output of the second and first integrator in an ideally compensated modulator. The red plot is with $c = 10$ and the green with $c = 0.05$. It is clear from the figure that the modulation of the signal has a reduced signal swing when $c = 0.05$. In a real system with noise this could lead to a limitation of the achievable SNDR. A transient noise analysis of the modulators would probably accomodate this, but the simulation tool [Advance MS] turned out to have a very limited functionality for this matter with mixed signal systems.

Chapter 7

Comments

The initial DT simulations showed that an OSR=64 is needed to meet the given accuracy specifications. The calculated CT scaling coefficients were also verified by doing a CT→DT conversion in [Matlab] and then check the correspondance with the original DT transfer function.

When using ideal compensation, the compensated modulator works as intended down to $c = 0.02$ (The lowest c-value tested in simulations). The noncompensated modulator performance is already degraded when $c \leq 2$. The simulations indicates that the modulator incorporating the compensation technique has similar robustness to nonidealities even for GBW values as low as $c \geq 0.05$ compared to the noncompensated modulator wich was simulated with $c = 10$. It is important to understand that this is only valid when the deviation of the amplifier GBW is within $\pm 40\%$ of the GBW compensated for. Especially when the actual amplifer GBW is less than this, the SNDR will suffer.

It was also discovered that the internal signal swings in the modulator is lowered when c is reduced. This might be a limitation for the pratical achievable SNDR in a real circuit.

The simulation of the CT modulators in [Advance MS] is a rather time consuming operation. Because of the long simulation time it was not feasible to do all the simulations on different frequencies and with different input amplitudes. The SNDR plots are an indication for how the modulators perform, and the results must be treated thereafter.

Chapter 8

Conclusions

The main goal of the work presented in this text was to investigate how circuit imperfections affects the performance of a continuous-time $\Delta\Sigma$ ADC employing finite amplifier GBW compensation compared to a similar ADC without compensation, and to find a setup which gives a resolution of more than 10 ENOB or 62 dB SNDR. Initial discrete-time simulations were done in [Matlab] and the ADC models are made in [VHDL-AMS] and simulated with [Advance MS]. Here follows the conclusions drawn from this work, and finally some thoughts about future work on this subject.

8.1 Conclusions

The discrete-time simulations showed that an $OSR = 64$ was needed to meet the accuracy specification. Two structural models of a 2-order continuous-time modulator were made, one of them incorporating the compensation technique for finite amplifier GBW. It was verified that the GBW compensation technique works very well under ideal conditions. I.e. if the actual GBW is equal to the value compensated for it performs like an ideal non-compensated CT modulator with $c > 2$ down to $c = 0.02$. It is also shown that the performance is quite dependent on deviation of the actual amplifier GBW. The dependency also increases when lowering the c-value compensated for. As long as the actual GBW stays within $\pm 40\%$ of the compensation value, simulations showed that the compensated modulators with $c \geq 0.05$ has similar performance as a noncompensated modulator with $c = 10$ when subject to circuit imperfections like finite OTA gain, integrator gain variations, excess loop delay and slew rate limitations. In previous work with this compensation technique [Ortmanns, 2004], circuit imperfections were only simulated on modulators with ideal compensation. Thus the discovered relatively high sensitivity to nonidealities under GBW variations is new information.
The simulations also revealed a lower internal signal swing in the modula-

tors when the amplifier GBW was reduced. This would probably make the SNDR in the compensated modulators more susceptible to circuit noise in a circuit implementation.

Even though some potential drawbacks with the compensation technique has been found in this work, it might still be a promising track to follow. If amplifiers with stable GBW and the right 1-pole transfer characteristics is found, and the signal swing issue can be neglected or somehow counteracted, the current consumption in CT modulators could be significantly reduced.

8.2 Future Work

A natural step further would be to extend the simulations with noise analysis to see how the reduced internal signal swing affects the performance when noise is included. It could also be useful to investigate the possibility of changing the scaling coefficients to compensate for the reduced signal swing.
The sensitivity to GBW variations will also require amplifier topologies which can provide the required stability of the GBW. Finding such an amplifier architecture with a 1-pole transfer function similar to the one used in the model would also be neccesary before doing a circuit implementation of the system.

Bibliography

[Cherry, 2000] James A. Cherry and W. Martin Snelgrove, 2000, *Continuous-Time Delta-Sigma Modulators for High Speed A/D Aonversion*

[Gerfers, 2003] Friedel Gerfers, Maurits Ortmanns and Yiannos Manoli, *A 1.5-V 12-bit Power-Efficient Continuous-Time Third-Order $\Sigma\Delta$ Modulator*, IEEE Journal of Solid-State Circuits, vol. 38, no. 8, pp. 1343-1352, August 2003.

[Johns, 1997] David A. Johns and Ken Martin, 1997, *Analog integrated circuit design*

[Marques, 1998] A. Marques, V. Peluso, M. S. Steyaert and W. M. Sansen, *Optimal Parameters for $\Delta\Sigma$ Modulator Topologies*, IEEE Transactions on Circuits and Systems -II, vol. 45, no. 9, pp. 1232-1241, Sept. 1998.

[Norsworthy, 1997] Steven R. Norsworthy, Richard Schreier and Gabor C. Temes, 1997, *Delta-Sigma Data Converters*

[Ortmanns, 2004] Maurits Ortmanns, Friedel Gerfers and Yiannos Manoli, *Compensation of Finite Gain-Bandwidth Induced Errors in Continuous-Time Sigma–Delta Modulators*, IEEE Transactions on Circuits and Systems -I, vol. 51, no. 6, pp. 1088-1099, June. 2004.

[Ortmanns, 2006] Maurits Ortmanns and Friedel Gerfers, 2006, *Continuous-Time Sigma-Delta A/D Conversion*

[Schreier, 2005] Richard Schreier and Gabor C. Temes, 2005, *Understanding Delta-Sigma Data Converters*

[VHDL-AMS] *VHDL-AMS, IEEE Std 1076.1-1999*, http://standards.ieee.org/reading/ieee/std/dasc/1076.1-1999.pdf

[Advance MS] *Mentor Graphics Advance ms*, http://www.mentor.com/products/fv/ams/advance_ms/

[Matlab] *Matlab*, http://www.mathworks.com/products/matlab/

[Octave] *GNU Octave*, http://www.octave.org/

[ΔΣ Toolbox] Richard Schreier, ΔΣ *Toolbox* http://www.mathworks.com/matlabcentral/fileexchange

[Project Meeting] Project meeting with Trond Ytterdal and Are Hellandsvik, 3.March 2006

Appendix A

Scaling coefficients compensated modulator

Table A.1: Scaling coefficients for GBW compensated 2-order modulator

c-value	k_2	k_h	GE_1
0.5	0.8302	0.6964	0.8627
0.3	0.8461	0.6637	0.7904
0.2	0.8539	0.6364	0.7154
0.1	0.8500	0.5787	0.5569
0.09	0.8468	0.5672	0.5307
0.08	0.8423	0.5530	0.5013
0.07	0.8358	0.5349	0.4680
0.06	0.8263	0.5110	0.4299
0.05	0.8119	0.4777	0.3859
0.04	0.7890	0.4276	0.3345
0.03	0.7487	0.3428	0.2738
0.02	0.6654	0.1752	0.2008

Table A.1 shows the scaling coefficents and the first integrator gain error in a compensated modulator for different values of c. The values have been calculated using the Matlab script in appendix C.1

Appendix B

VHDL-AMS Models

B.1 Resistor Model

```
1  ------------------------------------------------------------------------
2  -- Tres-ent.vhd
3  -- Project     : ctdsm
4  ------------------------------------------------------------------------
5  -- File        : res-ent.vhd
6  -- Author      : Jon Helge Nistad
7  -- Company     :
8  -- Created     : 2006-06-10
9  -- Last update: 2006-06-10
10 -- Platform    :
11 -- Standard    : VHDL'93, VHDL-AMS, Math Packages
12 ------------------------------------------------------------------------
13 -- Description: VHDL AMS entity for a resistor
14 ------------------------------------------------------------------------
15 -- Copyright (c) 2006
16 ------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author   Description
19 -- 2006-06-10  1.0      jonhelge         Created
20 ------------------------------------------------------------------------
21
22
23
24 library ieee_proposed;
25 use ieee_proposed.electrical_systems.all;
26
27 entity res is
28   generic (r            : resistance);
29   port (terminal n1, n2 : electrical);
30 end;
```

```
1  ------------------------------------------------------------------------
2  -- Tres-ideal.vhd
3  -- Project     : ctdsm
4  ------------------------------------------------------------------------
5  -- File        : res-ideal.vhd
6  -- Author      : Jon Helge Nistad
7  -- Company     :
8  -- Created     : 2006-06-10
9  -- Last update: 2006-06-10
10 -- Platform    :
11 -- Standard    : VHDL'93, VHDL-AMS, Math Packages
12 ------------------------------------------------------------------------
13 -- Description: Ideal Architecture of resistor
14 ------------------------------------------------------------------------
15 -- Copyright (c) 2006
16 ------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author   Description
19 -- 2006-06-10  1.0      jonhelge         Created
20 ------------------------------------------------------------------------
21
22
23
```

```
24  architecture ideal of res is
25    quantity v across i through n1 to n2;
26  begin
27    v == i*r;
28  end architecture;
```

B.2 Capacitor Model

```
1  ------------------------------------------------------------------------------
2  -- Tcap_vinit-ent.vhd
3  -- Project     :
4  ------------------------------------------------------------------------------
5  -- File        : cap_vinit-ent.vhd
6  -- Author      : Jon Helge Nistad
7  -- Company     :
8  -- Created     : 2006-06-10
9  -- Last update: 2006-06-10
10 -- Platform    :
11 -- Standard    : VHDL'93, VHDL-AMS, Math Packages
12 ------------------------------------------------------------------------------
13 -- Description: Entity for capacitor with initial voltage
14 ------------------------------------------------------------------------------
15 -- Copyright (c) 2006
16 ------------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2006-06-10  1.0      jonhelge         Created
20 ------------------------------------------------------------------------------
21
22
23
24 library IEEE_proposed;
25 use IEEE_proposed.electrical_systems.all;
26
27 entity cap is
28   generic(cap_value     : real;
29           init_voltage : voltage
30           );
31   port(terminal pos, neg : electrical
32        );
33 end entity cap;
34
35 ------------------------------------------------------------------------------
```

```
1  ------------------------------------------------------------------------------
2  -- Tcap_vinit-beh.vhd
3  -- Project     :
4  ------------------------------------------------------------------------------
5  -- File        : cap_vinit-beh.vhd
6  -- Author      : Jon Helge Nistad
7  -- Company     :
8  -- Created     : 2006-06-10
9  -- Last update: 2006-06-10
10 -- Platform    :
11 -- Standard    : VHDL'93, VHDL-AMS, Math Packages
12 ------------------------------------------------------------------------------
13 -- Description: Architecture for capacitor with inital voltage
14 ------------------------------------------------------------------------------
15 -- Copyright (c) 2006
16 ------------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2006-06-10  1.0      jonhelge         Created
20 ------------------------------------------------------------------------------
21
22
23
24 architecture ideal of cap is
25   quantity v across i through pos to neg;
26 begin
27   if domain = quiescent_domain use
28     v == init_voltage;
29 else
30   i == cap_value*v'dot;
31 end use;
32 end architecture ideal;
33
34 ------------------------------------------------------------------------------
```

B.3 OTA model

```
1  ------------------------------------------------------------------------
2  -- Filename:        OTA_ent.vhd
3  -- $Id: .emacs,v 1.417 2002/12/10 19:08:14 rs Exp $
4  -- Description:
5  -- Author:          Jon Helge Nistad <>
6  -- Created at:      Tue Feb  7 15:17:30 2006
7  -- Modified at:     Thu Feb 23 13:47:39 2006
8  -- Modified by:     Jon Helge Nistad <>
9  --
10 ------------------------------------------------------------------------
11 library IEEE;
12 use IEEE.MATH_REAL.all;
13 library ieee_proposed;
14 use ieee_proposed.electrical_systems.all;
15
16 entity diffOTA is
17
18   generic (
19     gain : real := 1.0;
20     GBW  : real := 1.0;
21     vcmn : real := 0.0);
22   port (
23     terminal inpp  : electrical;
24     terminal inpn  : electrical;
25     terminal outpp : electrical;
26     terminal outpn : electrical);
27
28 end entity diffOTA;
```

```
1  ------------------------------------------------------------------------
2  -- Filename:        diffOTA_gbw.vhd
3  -- $Id: .emacs,v 1.417 2002/12/10 19:08:14 rs Exp $
4  -- Description:
5  -- Author:          Jon Helge Nistad <>
6  -- Created at:      Tue Feb  7 15:35:55 2006
7  -- Modified at:     Mon Mar 13 12:55:03 2006
8  -- Modified by:     Jon Helge Nistad <>
9  --
10 ------------------------------------------------------------------------
11
12 architecture gbw of diffOTA is
13
14   quantity vinp across inpp to electrical_ref;
15   quantity vinn across inpn to electrical_ref;
16   quantity voutp across ioutp through outpp to electrical_ref;
17   quantity voutn across ioutn through outpn to electrical_ref;
18
19   -- The dominant pole
20   constant w_A : real := (math_2_pi*GBW/gain);
21
22   -- Parts of the transfer function
23   constant num : real_vector := (0 => gain);
24   constant den : real_vector := (1.0, 1.0/w_A);
25   quantity vdiff_half : voltage := 0.0;
26
27
28 begin  -- architecture gbw
29   vdiff_half  == (vinp-vinn)/2.0;
30   voutp == vcmn+vdiff_half'ltf(num,den);  -- multiply with transfer function
31   voutn == vcmn-vdiff_half'ltf(num,den);  -- and set outputs
32
33 end architecture gbw;
```

B.4 NRZ DAC Model

```
1  ------------------------------------------------------------------------
2  -- Tdac_ent.vhd
3  -- Project     :
4  ------------------------------------------------------------------------
5  -- File        : dac_ent.vhd
6  -- Author      : Jon Helge Nistad
7  -- Company     :
8  -- Created     : 2006-06-10
9  -- Last update: 2006-06-10
10 -- Platform    :
11 -- Standard    : VHDL'93, VHDL-AMS, Math Packages
12 ------------------------------------------------------------------------
```

```
13  -- Description: entity for the nrz dac
14  ------------------------------------------------------------------------------
15  -- Copyright (c) 2006
16  ------------------------------------------------------------------------------
17  -- Revisions  :
18  -- Date        Version  Author  Description
19  -- 2006-06-10  1.0      jonhelge         Created
20  ------------------------------------------------------------------------------
21
22
23
24  library IEEE;
25  use IEEE.MATH_REAL.all;
26  use IEEE.std_logic_1164.all;
27  library ieee_proposed;
28  use ieee_proposed.electrical_systems.all;
29
30  entity dac is
31
32    generic (
33      vrefP      : real := 5.0;           -- positive reference voltage
34      vrefN      : real := 0.0;           -- negative reference voltage
35      slew_up    : real := 1.0e12;        -- slew "up" rate
36      slew_down  : real := 1.0e12;        -- øsew "down" rate
37      delay      : time := 0.0 ns);       -- excess loop delay
38
39    port (
40      signal   input  : in std_logic;
41      signal   clk    : in std_logic;
42      terminal output :    electrical);
43
44  end entity dac;
```

```
1   ------------------------------------------------------------------------------
2   -- Tdac_ideal.vhd
3   -- Project    :
4   ------------------------------------------------------------------------------
5   -- File       : dac_ideal.vhd
6   -- Author     : Jon Helge Nistad
7   -- Company    :
8   -- Created    : 2006-06-10
9   -- Last update: 2006-06-10
10  -- Platform   :
11  -- Standard   : VHDL'93, VHDL-AMS, Math Packages
12  ------------------------------------------------------------------------------
13  -- Description: architecture for nrz dac
14  ------------------------------------------------------------------------------
15  -- Copyright (c) 2006
16  ------------------------------------------------------------------------------
17  -- Revisions  :
18  -- Date        Version  Author  Description
19  -- 2006-06-10  1.0      jonhelge         Created
20  ------------------------------------------------------------------------------
21
22
23
24  architecture ideal of dac is
25
26    quantity vout across iout through output to electrical_ref;
27    signal   vinternal : real := 0.0;
28  begin  -- architecture
29
30    dac : process(clk) is
31    begin  -- process clocked
32      if clk'event and clk = '1' then
33        if input = '1' then                 -- check the input
34          vinternal <= vrefN after delay; -- add the excess delay
35        else
36          vinternal <= vrefP after delay;
37        end if;
38      end if;
39    end process dac;
40
41    vout == vinternal'slew(slew_up, slew_down); -- set the output and
42                                                -- apply the slewing
43
44  end architecture ideal;
```

B.5 HRZ DAC Model

```
1   ------------------------------------------------------------------------------
```

```
2  -- Tdac_hrz_ent.vhd
3  -- Project     :
4  ------------------------------------------------------------------------------
5  -- File        : dac_hrz_ent.vhd
6  -- Author      : Jon Helge Nistad
7  -- Company     :
8  -- Created     : 2006-03-09
9  -- Last update: 2006-06-10
10 -- Platform    :
11 -- Standard    : VHDL'93, VHDL-AMS, Math Packages
12 ------------------------------------------------------------------------------
13 -- Description: entity of hrz-dac
14 ------------------------------------------------------------------------------
15 -- Copyright (c) 2006
16 ------------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2006-03-09  1.0      jhnistad         Created
20 ------------------------------------------------------------------------------
21
22 library IEEE;
23 use IEEE.MATH_REAL.all;
24 use IEEE.std_logic_1164.all;
25 library ieee_proposed;
26 use ieee_proposed.electrical_systems.all;
27
28 entity hrz_dac is
29
30   generic (
31     vrefP      : real := 1.0;           -- positive reference voltage
32     vrefN      : real := 0.0;           -- negative reference voltage
33     slew_up    : real := 1.0e15;        -- slew "up" rate
34     slew_down  : real := -1.0e15;       -- slew "down" rate
35     delay      : time := 0 ns);         -- excess loop delay
36
37   port (
38     signal   input  : in std_logic;
39     signal   clk    : in std_logic;
40     terminal output :    electrical);
41
42 end entity hrz_dac;
```

```
1  ------------------------------------------------------------------------------
2  -- Tdac_hrz_ideal.vhd
3  -- Project     :
4  ------------------------------------------------------------------------------
5  -- File        : dac_hrz_ideal.vhd
6  -- Author      : Jon Helge Nistad
7  -- Company     :
8  -- Created     : 2006-06-10
9  -- Last update: 2006-06-10
10 -- Platform    :
11 -- Standard    : VHDL'93, VHDL-AMS, Math Packages
12 ------------------------------------------------------------------------------
13 -- Description: Architecture for HRZ dac
14 ------------------------------------------------------------------------------
15 -- Copyright (c) 2006
16 ------------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2006-06-10  1.0      jonhelge         Created
20 ------------------------------------------------------------------------------
21
22
23
24 architecture ideal of hrz_dac is
25
26   quantity vout across iout through output to electrical_ref;
27   signal   vinternal : real := 0.0;
28 begin -- architecture ideal
29
30
31   dac : process(clk) is
32   begin -- process
33     if clk'event and clk = '1' then         -- at rising edge
34       vinternal <= 0.0 after delay;         -- zero the internal signal
35     elsif clk'event and clk = '0' then      -- at falling edge
36       if input = '1' then
37         vinternal <= vrefP after (delay); --check input
38       else
39         vinternal <= vrefN after (delay);
40       end if;
41     end if;
42   end process dac;
```

```
43
44    if vinternal = 0.0 use
45      iout ==0.0;                                    --cut current when zeroed
46    else
47      vout == vinternal'slew(slew_up, slew_down); -- set the ouput
48  end use;                                          -- and apply the slewing
49
50  end architecture ideal;
```

B.6 Quantizer Model

```
1  -------------------------------------------------------------------------
2  -- Filename:       quantizer_ent.vhd
3  -- $Id: .emacs,v 1.417 2002/12/10 19:08:14 rs Exp $
4  -- Description:
5  -- Author:         Jon Helge Nistad <>
6  -- Created at:     Fri Feb  3 08:55:44 2006
7  -- Modified at:    Fri Feb  3 09:16:32 2006
8  -- Modified by:    Jon Helge Nistad <>
9  --
10 -------------------------------------------------------------------------
11 library IEEE;
12 use IEEE.MATH_REAL.all;
13 use IEEE.std_logic_1164.all;
14 library ieee_proposed;
15 use ieee_proposed.electrical_systems.ALL;
16
17 entity diffquantizer is
18
19   generic (
20     treshold : real := 0.0);
21
22   port (
23     terminal inpp  :     electrical;
24     terminal inpn  :     electrical;
25     signal   output : out std_logic;
26     signal   clk    : in  std_logic);
27 end entity diffquantizer;
```

```
1  -------------------------------------------------------------------------
2  -- Filename:       quantizer_ideal.vhd
3  -- $Id: .emacs,v 1.417 2002/12/10 19:08:14 rs Exp $
4  -- Description:
5  -- Author:         Jon Helge Nistad <>
6  -- Created at:     Fri Feb  3 09:01:54 2006
7  -- Modified at:    Fri Feb  3 09:10:06 2006
8  -- Modified by:    Jon Helge Nistad <>
9  --
10 -------------------------------------------------------------------------
11
12 architecture ideal of diffquantizer is
13
14   quantity vinp across inpp to electrical_ref;
15   quantity vinn across inpn to electrical_ref;
16   quantity vdiff :voltage := 0.0;
17 begin  -- architecture ideal
18   vdiff == vinp-vinn;
19   clocked : process (clk) is
20   begin  -- process clocked
21     if clk'event and clk = '1' then      -- rising clock edge
22       if vdiff'above(treshold) then
23         output <= '1';
24       else
25         output <= '0';
26       end if;
27     end if;
28   end process clocked;
29
30 end architecture ideal;
```

B.7 2-order CT modulator Model

```
1  -------------------------------------------------------------------------
2  -- Fctsdm2_ent.vhd
3  -- $Id: .emacs,v 1.417 2002/12/10 19:08:14 rs Exp $
4  -- Description:
5  -- Author:         Jon Helge Nistad <>
6  -- Created at:     Fri Feb  3 10:01:41 2006
```

```
7  -- Modified at:   Sat Jun 10 16:00:31 2006
8  -- Modified by:   Jon Helge Nistad <>
9  ------------------------------------------------------------------------
10
11 library IEEE;
12 use IEEE.MATH_REAL.all;
13 use IEEE.std_logic_1164.all;
14 library ieee_proposed;
15 use ieee_proposed.electrical_systems.all;
16
17 entity ctsdm2 is
18
19
20   generic (
21     GBW                : real := 1.0;   -- OTA GBW in Hz
22     otagain            : real := 1.0;   -- OTA gain
23     vcmn               : real := 0.0;   -- common mode voltage
24     R1val              : real := 1.0;   -- Value of R1
25     R2val              : real := 1.0;   -- Value of R2
26     Rdac1val           : real := 1.0;   -- Value of RDAC1
27     Rdac2val           : real := 1.0;   -- Value of RDAC2
28     Hdacval            : real := 1.0;   -- Value of HDAC
29     C1val              : real := 1.0;   -- Value of C1
30     C2val              : real := 1.0;   -- Vaule of C2
31     vrefP              : real := 1.0;   -- Positive Reference Voltage
32     vrefN              : real := -1.0;  -- Negative Reference Voltage
33     dac1delay          : time := 1.0 ns; -- Excess loop delay DAC1
34     dac2delay          : time := 1.0 ns; -- Excess loop delay DAC2
35     delayedclock_delay : time := 1.0 ns; -- Delay in clock generator
36     hrzdacdelay        : time := 1.0 ns; -- Excess loop delay in HRZ DAC
37     dac_rise           : real := 1.0;    -- Slew "up" rate in the DACs
38     dac_fall           : real := -1.0); -- Slew "down" rate in the DACs
39
40   port (
41     terminal inpp   :     electrical;
42     terminal inpn   :     electrical;
43     signal   output : out std_logic;
44     signal   clk    : in  std_logic);
45
46 end entity ctsdm2;
```

```
1  ------------------------------------------------------------------------
2  -- Fctsdm2_nocomp.vhd
3  -- $Id: .emacs,v 1.417 2002/12/10 19:08:14 rs Exp $
4  -- Description:
5  -- Author:          Jon Helge Nistad <>
6  -- Created at:      Fri Feb  3 10:04:37 2006
7  -- Modified at:     Sat Jun 10 16:09:48 2006
8  -- Modified by:     Jon Helge Nistad <>
9  --
10 ------------------------------------------------------------------------
11 library MGC_AMS;
12 use MGC_AMS.ELDO.all;
13
14
15
16 architecture nocomp of ctsdm2 is
17
18
19   terminal inOTA1p  : electrical;
20   terminal inOTA1n  : electrical;
21   terminal inOTA2p  : electrical;
22   terminal inOTA2n  : electrical;
23   terminal outOTA1p : electrical;
24   terminal outOTA1n : electrical;
25   terminal outOTA2p : electrical;
26   terminal outOTA2n : electrical;
27   terminal inRdac1a : electrical;
28   terminal inRdac1b : electrical;
29   terminal inRdac2a : electrical;
30   terminal inRdac2b : electrical;
31   terminal T3       : electrical;
32
33   signal delayedclock : std_logic := '0';
34   signal s1 : std_logic;
35
36
37 begin  -- architecture with no compensation
38
39   -- Here are the delta sigma modulator put together of
40   -- differen sub components
41
42   OTA1 : entity work.diffOTA(gbw)
43     generic map (
```

```
44          gain => otagain,
45          GBW  => GBW,
46          vcmn => vcmn)
47
48        port map (
49          inpp  => inOTA1p,
50          inpn  => inOTA1n,
51          outpp => outOTA1p,
52          outpn => outOTA1n);
53
54      OTA2 : entity work.diffOTA(gbw)
55        generic map (
56          gain => otagain,
57          GBW  => GBW,
58          vcmn => vcmn)
59        port map (
60          inpp  => inOTA2p,
61          inpn  => inOTA2n,
62          outpp => outOTA2p,
63          outpn => outOTA2n);
64
65
66      R1a : entity work.res(ideal)
67        generic map (
68          r => R1val)
69        port map (
70          n1 => inpn,
71          n2 => inOTA1p);
72
73      R1b : entity work.res(ideal)
74        generic map (
75          r => R1val)
76        port map (
77          n1 => inpp,
78          n2 => inOTA1n);
79
80      R2a : entity work.res(ideal)
81        generic map (
82          r => R2val)
83        port map (
84          n1 => outOTA1p,
85          n2 => inOTA2p);
86
87      R2b : entity work.res(ideal)
88        generic map (
89          r => R2val)
90        port map (
91          n1 => outOTA1n,
92          n2 => inOTA2n);
93
94
95
96      Rdac1a : entity work.res(ideal)
97        generic map (
98          r => Rdac1val)
99        port map (
100         n1 => inRdac1a,
101         n2 => inOTA1n);
102
103     Rdac1b : entity work.res(ideal)
104       generic map (
105         r => Rdac1val)
106       port map (
107         n1 => inRdac1b,
108         n2 => inOTA1p);
109
110     Rdac2a : entity work.res(ideal)
111       generic map (
112         r => Rdac2val)
113       port map (
114         n1 => inRdac2a,
115         n2 => inOTA2n);
116
117     Rdac2b : entity work.res(ideal)
118       generic map (
119         r => Rdac2val)
120       port map (
121         n1 => inRdac2b,
122         n2 => inOTA2p);
123
124
125
126     C1a : entity work.cap(ideal)
127       generic map (
```

```
128         cap_value     => C1val,
129         init_voltage => 0.0)
130       port map (
131         pos => inOTA1n,
132         neg => outOTA1p);
133
134     C1b : entity work.cap(ideal)
135       generic map (
136         cap_value     => C1val,
137         init_voltage => 0.0)
138       port map (
139         pos => inOTA1p,
140         neg => outOTA1n);
141
142     C2a : entity work.cap(ideal)
143       generic map (
144         cap_value     => C2val,
145         init_voltage => 0.0)
146       port map (
147         pos => inOTA2n,
148         neg => outOTA2p);
149
150     C2b : entity work.cap(ideal)
151       generic map (
152         cap_value     => C2val,
153         init_voltage => 0.0)
154       port map (
155         pos => inOTA2p,
156         neg => outOTA2n);
157
158     quantizer : entity work.diffquantizer(ideal)
159       generic map (
160         treshold => 0.0)
161       port map (
162         inpp    => outOTA2n,
163         inpn    => outOTA2p,
164         output => s1,
165         clk     => clk);
166
167   -- the 4 dac blocks
168     dac1a : entity work.dac(ideal)
169       generic map (
170         vrefP       => vrefP,
171         vrefN       => vrefN,
172         slew_up     => dac_rise,
173         slew_down => dac_fall,
174         delay       => dac1delay)
175       port map (
176         input   => s1,
177         clk     => delayedclock,
178         output => inRdac1a);
179
180     dac1b : entity work.dac(ideal)
181       generic map (
182         vrefP       => vrefN,
183         vrefN       => vrefP,
184         slew_up     => dac_rise,
185         slew_down => dac_fall,
186         delay       => dac1delay)
187       port map (
188         input   => s1,
189         clk     => delayedclock,
190         output => inRdac1b);
191
192     dac2a : entity work.dac(ideal)
193       generic map (
194         vrefP       => vrefP,
195         vrefN       => vrefN,
196         slew_up     => dac_rise,
197         slew_down => dac_fall,
198         delay       => dac2delay)
199       port map (
200         input   => s1,
201         clk     => delayedclock,
202         output => inRdac2a);
203
204     dac2b : entity work.dac(ideal)
205       generic map (
206         vrefP       => vrefN,
207         vrefN       => vrefP,
208         slew_up     => dac_rise,
209         slew_down => dac_fall,
210         delay       => dac2delay)
211       port map (
```

```
212         input  => s1,
213         clk    => delayedclock,
214         output => inRdac2b);
215
216
217
218   -- delayed clock generator
219     clkdelaygen : process (clk) is
220     begin
221       if clk'event and clk = '1' then
222         delayedclock <= '1' after delayedclock_delay;
223       else
224         delayedclock <= '0' after delayedclock_delay;
225       end if;
226     end process clkdelaygen;
227
228
229
230     output <= s1;
231   end architecture structural;
```

B.8 2-order CT modulator Model with GBW compensation

```
1   ------------------------------------------------------------------------
2   -- Fctsdm2_struct.vhd
3   -- $Id: .emacs,v 1.417 2002/12/10 19:08:14 rs Exp $
4   -- Description:
5   -- Author:          Jon Helge Nistad<>
6   -- Created at:      Fri Feb  3 10:04:37 2006
7   -- Modified at:     Sat Jun 10 16:08:05 2006
8   -- Modified by:     Jon Helge Nistad<>
9   --
10  ------------------------------------------------------------------------
11  library MGC_AMS;
12  use MGC_AMS.ELDO.all;
13
14
15
16  architecture structural of ctsdm2 is
17
18
19    terminal inOTA1p  : electrical;
20    terminal inOTA1n  : electrical;
21    terminal inOTA2p  : electrical;
22    terminal inOTA2n  : electrical;
23    terminal outOTA1p : electrical;
24    terminal outOTA1n : electrical;
25    terminal outOTA2p : electrical;
26    terminal outOTA2n : electrical;
27    terminal inRdac1a : electrical;
28    terminal inRdac1b : electrical;
29    terminal inRdac2a : electrical;
30    terminal inRdac2b : electrical;
31    terminal inHdacA  : electrical;
32    terminal inHdacB  : electrical;
33    terminal T3       : electrical;
34
35    signal delayedclock : std_logic := '0'
36    signal s1 : std_logic;
37
38
39  begin  -- architecture structural
40
41    -- Here are the delta sigma modulator put together of
42    -- differen sub components
43
44    OTA1 : entity work.diffOTA(gbw)
45      generic map (
46        gain => otagain,
47        GBW  => GBW,
48        vcmn => vcmn)
49
50      port map (
51        inpp  => inOTA1p,
52        inpn  => inOTA1n,
53        outpp => outOTA1p,
54        outpn => outOTA1n);
55
56    OTA2 : entity work.diffOTA(gbw)
```

```
57        generic map (
58          gain => otagain,
59          GBW  => GBW,
60          vcmn => vcmn)
61        port map (
62          inpp  => inOTA2p,
63          inpn  => inOTA2n,
64          outpp => outOTA2p,
65          outpn => outOTA2n);
66
67
68      R1a : entity work.res(ideal)
69        generic map (
70          r => R1val)
71        port map (
72          n1 => inpn,
73          n2 => inOTA1p);
74
75      R1b : entity work.res(ideal)
76        generic map (
77          r => R1val)
78        port map (
79          n1 => inpp,
80          n2 => inOTA1n);
81
82      R2a : entity work.res(ideal)
83        generic map (
84          r => R2val)
85        port map (
86          n1 => outOTA1p,
87          n2 => inOTA2p);
88
89      R2b : entity work.res(ideal)
90        generic map (
91          r => R2val)
92        port map (
93          n1 => outOTA1n,
94          n2 => inOTA2n);
95
96
97
98      Rdac1a : entity work.res(ideal)
99        generic map (
100         r => Rdac1val)
101       port map (
102         n1 => inRdac1a,
103         n2 => inOTA1n);
104
105     Rdac1b : entity work.res(ideal)
106       generic map (
107         r => Rdac1val)
108       port map (
109         n1 => inRdac1b,
110         n2 => inOTA1p);
111
112     Rdac2a : entity work.res(ideal)
113       generic map (
114         r => Rdac2val)
115       port map (
116         n1 => inRdac2a,
117         n2 => inOTA2n);
118
119     Rdac2b : entity work.res(ideal)
120       generic map (
121         r => Rdac2val)
122       port map (
123         n1 => inRdac2b,
124         n2 => inOTA2p);
125
126
127
128     C1a : entity work.cap(ideal)
129       generic map (
130         cap_value    => C1val,
131         init_voltage => 0.0)
132       port map (
133         pos => inOTA1n,
134         neg => outOTA1p);
135
136     C1b : entity work.cap(ideal)
137       generic map (
138         cap_value    => C1val,
139         init_voltage => 0.0)
140       port map (
```

```
141         pos => inOTA1p,
142         neg => outOTA1n);
143
144     C2a : entity work.cap(ideal)
145       generic map (
146         cap_value     => C2val,
147         init_voltage => 0.0)
148       port map (
149         pos => inOTA2n,
150         neg => outOTA2p);
151
152     C2b : entity work.cap(ideal)
153       generic map (
154         cap_value     => C2val,
155         init_voltage => 0.0)
156       port map (
157         pos => inOTA2p,
158         neg => outOTA2n);
159
160     quantizer : entity work.diffquantizer(ideal)
161       generic map (
162         treshold => 0.0)
163       port map (
164         inpp    => outOTA2n,
165         inpn    => outOTA2p,
166         output => s1,
167         clk     => clk);
168
169   -- the 4 dac blocks
170     dac1a : entity work.dac(ideal)
171       generic map (
172         vrefP       => vrefP,
173         vrefN       => vrefN,
174         slew_up     => dac_rise,
175         slew_down => dac_fall,
176         delay       => dac1delay)
177       port map (
178         input   => s1,
179         clk     => delayedclock,
180         output => inRdac1a);
181
182     dac1b : entity work.dac(ideal)
183       generic map (
184         vrefP       => vrefN,
185         vrefN       => vrefP,
186         slew_up     => dac_rise,
187         slew_down => dac_fall,
188         delay       => dac1delay)
189       port map (
190         input   => s1,
191         clk     => delayedclock,
192         output => inRdac1b);
193
194     dac2a : entity work.dac(ideal)
195       generic map (
196         vrefP       => vrefP,
197         vrefN       => vrefN,
198         slew_up     => dac_rise,
199         slew_down => dac_fall,
200         delay       => dac2delay)
201       port map (
202         input   => s1,
203         clk     => delayedclock,
204         output => inRdac2a);
205
206     dac2b : entity work.dac(ideal)
207       generic map (
208         vrefP       => vrefN,
209         vrefN       => vrefP,
210         slew_up     => dac_rise,
211         slew_down => dac_fall,
212         delay       => dac2delay)
213       port map (
214         input   => s1,
215         clk     => delayedclock,
216         output => inRdac2b);
217
218     -- EXTRA ELEMENTS IN THIS HRZ VERSION----
219     HdacA : entity work.res(ideal)
220       generic map (
221         r => Hdacval)
222       port map (
223         n1 => inHdacA,
224         n2 => inOTA2n);
```

```
225
226     HdacB : entity work.res(ideal)
227       generic map (
228         r => Hdacval)
229       port map (
230         n1 => inHdacB,
231         n2 => inOTA2p);
232
233     HrzdacA : entity work.hrz_dac(ideal)
234       generic map (
235         vrefP      => vrefP,
236         vrefN      => vrefN,
237         slew_up    => dac_rise,
238         slew_down => dac_fall,
239         delay      => hrzdacdelay)
240       port map (
241         input  => s1,
242         clk    => delayedclock,
243         output => inHdacA);
244
245     HrzdacB : entity work.hrz_dac(ideal)
246       generic map (
247         vrefP      => vrefN,
248         vrefN      => vrefP,
249         slew_up    => dac_rise,
250         slew_down => dac_fall,
251         delay      => hrzdacdelay)
252       port map (
253         input  => s1,
254         clk    => delayedclock,
255         output => inHdacB);
256
257     -- END OF EXTRA HRZ ELEMENTz--
258
259
260   -- delayed clock generator
261     clkdelaygen : process (clk) is
262     begin
263       if clk'event and clk = '1' then
264         delayedclock <= '1' after delayedclock_delay;
265       else
266         delayedclock <= '0' after delayedclock_delay;
267       end if;
268     end process clkdelaygen;
269
270
271
272     output <= s1;
273   end architecture structural;
```

B.9 VHDL testbench

```
1   ------------------------------------------------------------------------------
2   -- Ttb_ctsdm2.vhd
3   -- Project     :
4   ------------------------------------------------------------------------------
5   -- File        : tb_deltasigma.vhd
6   -- Author      : Jon Helge Nistad
7   -- Company     :
8   -- Created     : 2006-01-17
9   -- Last update: 2006-06-10
10  -- Platform    :
11  -- Standard    : VHDL'93, VHDL-AMS, Math Packages
12  ------------------------------------------------------------------------------
13  -- Description: Deltasigma testbench
14  ------------------------------------------------------------------------------
15  -- Copyright (c) 2006
16  ------------------------------------------------------------------------------
17  -- Revisions   :
18  -- Date        Version   Author   Description
19  -- 2006-01-17  1.0       jonhelge          Created
20  ------------------------------------------------------------------------------
21
22  library ieee;
23  use ieee.math_real.all;
24  use ieee.std_logic_1164.all;
25
26  use ieee.std_logic_arith.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.Numeric_std.all;
29  use STD.TEXTIO.all;
```

```
30  use ieee.numeric_bit.all;
31
32
33  library IEEE_proposed;
34  use IEEE_proposed.electrical_systems.all;
35
36  library MGC_AMS;
37  use MGC_AMS.eldo_parameters.all;
38
39  entity tb_deltasigma_ct2 is
40
41  end entity tb_deltasigma_ct2;
42
43  architecture bhv of tb_deltasigma_ct2 is
44
45    terminal A          : electrical;
46    terminal B          : electrical;
47    quantity Ain across iAin through A to electrical_ref;
48    quantity Bin across iBin through B to electrical_ref;
49    quantity anaop      : voltage := 0.0;
50    signal   op         : std_logic;
51    signal   clk        : std_logic;
52    signal   anaoptemp  : real    := 0.0;
53    ------------------------------------------------------------------------
54
55    --eldo parameters set in the .cmd top level file-----------------------
56    constant amp                : real := param("amp");
57    constant fbin               : real := param("fbin");
58    constant OSR                : real := param("OSR");
59    constant fin                : real := param("fin");
60    constant GBW                : real := param("GBW");
61    constant otagain            : real := param("otagain");
62    constant vcmn               : real := param("vcmn");
63    constant R1val              : real := param("R1val");
64    constant R2val              : real := param("R2val");
65    constant Rdac1val           : real := param("Rdac1val");
66    constant Rdac2val           : real := param("Rdac2val");
67    constant Hdacval            : real := param("Hdacval");
68    constant C1val              : real := param("C1val");
69    constant C2val              : real := param("C2val");
70    constant vrefP              : real := param("vrefP");
71    constant vrefN              : real := param("vrefN");
72    constant dac1delay          : real := param("dac1delay");
73    constant dac2delay          : real := param("dac2delay");
74    constant delayedclock_delay : real := param("delayedclock_delay");
75    constant hrzdacdelay        : real := param("hrzdacdelay");
76    constant GBWcomp            : real := param("GBWcomp");
77    constant dac_rise           : real := param("dac_rise");
78    constant dac_fall           : real := param("dac_fall");
79    ------------------------------------------------------------------------
80
81
82    file rapport : text open append_mode is "rapport.txt";
83
84  begin -- architecture bhv
85
86
87    -- initiate the delta sigma modulator
88    dut : entity work.ctsdm2(structural)
89      generic map (
90        GBW                 => GBW,
91        otagain             => otagain,
92        vcmn                => vcmn,
93        R1val               => R1val,
94        R2val               => R2val,
95        Rdac1val            => Rdac1val,
96        Rdac2val            => Rdac2val,
97        Hdacval             => Hdacval,
98        C1val               => C1val,
99        C2val               => C2val,
100       vrefP               => vrefP,
101       vrefN               => vrefN,
102       dac1delay           => (dac1delay*1.0 ns),
103       dac2delay           => (dac2delay*1.0 ns),
104       delayedclock_delay  => (delayedclock_delay*1.0 ns),
105       hrzdacdelay         => (hrzdacdelay * 1.0 ns),
106       dac_rise            => dac_rise,
107       dac_fall            => dac_fall)
108
109     port map (
110       inpp   => A,
111       inpn   => B,
112       output => op,
113       clk    => clk);
```

```
114
115
116     -- write the header of the output file
117     -- this is used to parse the results in the octave script
118     wr_header : process is
119       variable L    : line;
120       variable temp : integer := 0;
121     begin
122       write(L, OSR);
123       writeLine(rapport, L);
124       write(L, fbin);
125       writeLine(rapport, L);
126       write(L, GBWcomp);
127       writeLine(rapport, L);
128       write(L, GBW);
129       writeLine(rapport, L);
130       write(L, otagain);
131       writeLine(rapport, L);
132       write(L, amp);
133       writeLine(rapport, L);
134       write(L, fin);
135       writeLine(rapport, L);
136       wait;
137     end process wr_header;
138
139     -- Generate the input signal to the modulator
140     Ain ==vcmn+amp*sin(math_2_pi*fin*NOW);
141     Bin ==vcmn-amp*sin(math_2_pi*fin*NOW);
142
143     -- Process for input clock generation
144     -- f_clk=6.4 MHz
145
146     stimulus : process is
147     begin  -- process stimulus
148       clk <= '1';
149       wait for 78.125*1 ns;
150       clk <= '0';
151       wait for 78.125*1 ns;
152     end process stimulus;
153
154
155     -- The process that writes the output value to a file.
156     writefile : process(clk) is
157       variable L    : line;
158       variable temp : integer := 0;
159     begin
160       if falling_edge(clk) then           -- do a sample at a stable moment
161         if op = '1' then temp := 1;
162         else temp             := 0;
163         end if;
164         write(L, temp);                   -- write it to the file
165         writeLine(rapport, L);
166       end if;
167     end process writefile;
168
169     -- A process for converting the digital output to a analog signal.
170     -- Nice for doing measurements on the output in Ezwave
171     writeanaop : process(op) is
172     begin
173       if op = '1' then anaoptemp <= 1.0;
174       else anaoptemp             <= -1.0;
175       end if;
176     end process writeanaop;
177
178
179     anaop == anaoptemp'ramp(1.0e-15);     -- ramp the analog version of ouput.
180   end architecture bhv;
```

Appendix C

Matlab Code

C.1 Code for calculation of compensated feedback coefficents

```
%% Author Jon Helge Nistad 2006
%% GE1, k2 and kh must be set to
%% "something" be fore running script"

fs=6.4e6;
c=0.015
GBW_norm=fs*2*pi;
GBW=c*GBW_norm;

%% Poles
w2=(GBW+fs/(GE1)+k2*fs+kh*fs);
w1=(GBW+0.25*fs+0.25*fs);

%% Delays
tau1=((1-exp(-w2/fs))/(w2))*fs
tau2=(((w1^2)*(1-exp(-w2/fs))-(w2^2)*(1-exp(-w1/fs)))/(w1*w2*(w1-w2)))*fs

%% Scaling coefficients
k2=(6*tau1+4*tau1*tau2-tau2^2)/(8*tau1)
kh=(3*tau1+2*tau1*tau2-tau2^2)/(4*tau1)

% Gain errors
GE1=GBW/(w1)
GE2=GBW/(w2)
```

C.2 Code for simulation of DT filter and DT simulation of CT loop filter

```
% Author: Jon Helge Nistad, 2005
% Simulation and comparision of 2.order DT and CT
% delta sigma modulator. CT simulation is done by
% first doing CT-DT transform of the CT loop filter.

OSR=64;
fB=100e3;
fs=fB*OSR;

z=zpk('z',1)

%% The optimal DT NTF a1=a2=0.5
NTFcifb=1/(1-(-0.5/(z-1)-(0.5*0.5)/(z-1)^2))

%% The results from the imp. inv. transform
%% was k1=0.25 k2=0.375 kv=1. This is a representation
%% of the CT loop filter
Ac=[0 0; 1 0];
Bc=[1 -0.25; 0 -0.375];
```

```
20  Cc=[0 1];
21  Dc=[0 0];
22
23  %% Making a state space description
24  LFc=ss(Ac,Bc,Cc,Dc);
25  LF0c=ss(Ac,Bc(:,1),Cc,Dc(1));
26  L0c=zpk(LF0c);
27
28  %% Do a CT-DT conversion
29  LF=mapCtoD(LFc,[0 1])
30  ABCD=[LF.a LF.b; LF.c LF.d];
31  %% Find the new NTF
32  H=calculateTF(ABCD)
33
34  %%Plot a pole-zero map of the two NTF's
35  figure(1);
36  pzmap(H,NTFcifb);
37
38  %%Simulate the two NTFs with different input amplitudes.
39  amp=[-140:5:-15 -12 -10:0];
40  snr1=simulateSNR(H,OSR,amp,0,2,1/(4*48),16);
41  snr2=simulateSNR(NTFcifb,OSR,amp,0,2,1/(4*48),16);
42  figure(2)
43  hold on;
44  plot(amp,snr1,'g',amp,snr2,'b');
```

Appendix D

Octave code

D.1 Octave script for SNDR calculation

```
1  ## A script for reading the output file from
2  ## the vhdl testbench.
3  ## Author: Jon Helge Nistad
4
5  ## Read the values from rapport.txt
6  f=fopen("result.txt","r","ieee-le");
7  invec=fscanf(f,"%f");
8  fclose(f);
9
10
11 ##Find the length of the bitstreams
12 ##The first value in the header is OSR
13 cnt=2;
14 while(invec(cnt) != invec(1))
15   cnt++;
16 end
17 bLength=cnt-1; ## The length of the bitstreams
18
19 ## Calculates how many sim. runs done
20 nRuns=(length(invec)/bLength)
21
22
23 ## Create the file octres and write a header into it
24 octres=fopen("octres","w");
25 fprintf(octres,"### RESULTS FROM OCTAVE RUNS ######### \n");
26 klokk=clock;
27 fprintf(octres,"### Date: %d.%d.%d Time: %d:%d:%d ### \n",klokk(3),klokk(2),klokk(1),klokk(4),klokk
       (5),klokk(6));
28 fprintf(octres,"### Total number of runs = %d ######## \n",nRuns);
29
30 ## create the two .csv files for the plots
31 spectra=fopen("spectra.csv","w");
32 integ=fopen("spectrainteg.csv","w");
33
34 for runde=1:nRuns
35     newVec=invec((bLength*(runde-1)+1):(bLength*runde));
36     OSR=newVec(1);
37     fbin=newVec(2);
38     GBWcomp=newVec(3);
39     GBWact=newVec(4);
40     otagain=newVec(5);
41     amp=newVec(6);
42     fin=newVec(7);
43     result=newVec((8+100):end);
44     result=(result-0.5)/0.5;
45     N=length(result);
46
47
48 ## SNDR calculation as done at page 380 in schreier05 ##
49 w = hann(N); # Hann window
50 nb=3; #Hann
51 w=fliplr(w); #Rotate and flip the vector so it matches V
52 w=rot90(w);
53
```

```
54  w1=norm(w,1);
55  w2=norm(w,2);
56  NBW=(w2/w1)^2;
57
58  V=fft(w.*result)/(w1/2);
59  signal_bins=fbin+[-(nb-1)/2:(nb-1)/2];
60  noise_bins=1:(N/(2*OSR)+1);
61  noise_bins(signal_bins)=[];
62  snr=10*log10(sum(abs(V(signal_bins)).^2) / sum(abs(V(noise_bins)).^2));
63  ## END of SNDR calucalation
64
65  ## Write the calculated SNDR to the octres file
66  fprintf(octres,"\n------ Results run %d ------\n", runde)
67  fprintf(octres,"Number of samples = %d\n", N);
68  fprintf(octres,"fbin = %d\n",fbin);
69  fprintf(octres,"-------------------\n|SNDR = %f|\n-------------------\n",snr);
70  fprintf(octres,"GBWcomp = %d  GBWact=%d  otagain=%d\n amp=%f  fin=%d\n\n" ,GBWcomp, GBWact, otagain
        , amp, fin);
71
72
73
74  ## WRITE the spectra.csv file
75  ## First the header needed by ezwave
76
77  fprintf(spectra,".HEADER\n");
78  fprintf(spectra,"..NAMES\n");
79  fprintf(spectra,"Freq,%d GBWact=%d  otagain=%d\n" , runde,GBWact, otagain );
80
81  fprintf(spectra,"..UNITS\n");
82  fprintf(spectra,"Hz, dBFS\n");
83  fprintf(spectra,"..DATATYPES\n");
84  fprintf(spectra,"double,double\n");
85  fprintf(spectra,"..WAVEFORM_TYPES\n");
86  fprintf(spectra,",analog\n");
87  fprintf(spectra,"..AXIS_SPACING\n");
88  fprintf(spectra,"linear\n");
89  fprintf(spectra,"..FOLDER_NAME\n");
90  fprintf(spectra,"FFT\n");
91  fprintf(spectra,".DATA\n");
92
93  ## Then write the spectrum to the file
94  j=floor(length(V)/2);
95  for i=1:j
96    fprintf(spectra,"%E,%E\n",i,20*log10(abs(V(i))));
97  end
98
99
100 ## WRITE the spectrainteg.csv file
101 ## First the header needed by ezwave
102
103 fprintf(integ,".HEADER\n");
104 fprintf(integ,"..NAMES\n");
105 fprintf(integ,"Freq,%d  GBWact=%d  otagain=%d\n" ,runde, GBWact, otagain );
106
107 fprintf(integ,"..UNITS\n");
108 fprintf(integ,"Hz, dBFS\n");
109 fprintf(integ,"..DATATYPES\n");
110 fprintf(integ,"double,double\n");
111 fprintf(integ,"..WAVEFORM_TYPES\n");
112 fprintf(integ,",analog\n");
113 fprintf(integ,"..AXIS_SPACING\n");
114 fprintf(integ,"linear\n");
115 fprintf(integ,"..FOLDER_NAME\n");
116 fprintf(integ,"FFT\n");
117 fprintf(integ,".DATA\n");
118
119 ## Then write the integrated spectrum
120 j=floor(length(V)/2);
121 temp=0;
122 temp2=0;
123 for i=1:j
124   if(fbin-2<i &&  i<fbin+2) ## separate signal bins
125   temp2=temp2+(abs(V(i))^2);
126   fprintf(integ,"%E,%E\n",i,10*log10(temp2));
127   else
128   temp=temp+(abs(V(i))^2); ## and noise bins
129   fprintf(integ,"%E,%E\n",i,10*log10(temp));
130   end
131 end
132
133
134 end
135
136 ## finally close the files
```

```
137
138 fprintf(octres,"### END ###\n");
139 fclose(octres);
140 fclose(spectra);
141 fclose(integ);
```