NTNU
Innovation and Creativity

# Investigation of errors in open-loop sigma-delta modulators utilizing analog modulo integrators

**Øystein Knauserud**

Master of Science in Electronics
Submission date: June 2006
Supervisor: Trond Ytterdal, IET
Co-supervisor: Carsten Wulff, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

# Problem Description

The student is to design and build a practical open-loop sigma-delta adc to see if it confirms with the theory presented in a preliminary project. The circuit will be made of discrete components and measurements will be done in a laboratory. The student will also model a third order open-loop sigma-delta adc and simulate it with ideal conditions. In addition the adc will be simulated with common circuit imperfections to investigate the effects when there is no feedback present.
Finally the student will model and simulate a high order adc with circuit imperfections to see if it meets the requirements for use in a GSM system.

Assignment given: 19. January 2006
Supervisor: Trond Ytterdal, IET

# Abstract

This thesis is divided into two parts, the design of a practical first order open loop $\Sigma\Delta$ modulator using discrete components, and simulation of a third order OLSD ADC to investigate the consequences of circuit imperfections - and determining circuit requirements if the ADC should be used in a GSM system.

The practical modulator is designed as a first order OLSD ADC, with standard discrete components such as operational amplifiers and switches, and a microcontroller with a built in ADC. The practical circuit uses surface mount capacitors with a tolerance of 20%, resulting in poor matching and inaccurate behavior of the modulo integrator. Despite the poor matching, the OLSD ADC shows a distinct noise shaping, with a slope of about $20dB$ per decade. The quantization noise is not the dominating noise source in the circuit, and the quantizer resolution must to be set to four bits or less to achieve any improvement in performance over the standard ADC.

The third order modulator is modeled and simulated at a behavior level using VHDL-AMS. The ideal circuit confirms the results from the preliminary project [12], where the quantizer resolution had to be equal to or larger than the modulator order to obtain proper noise shaping. The simulations shows that the ideal third order modulator with a four bit quantizer can achieve a SNR of $88.51dB$, and an ENOB of $13.78bits$ within a $200kHz$ band.

The third order modulator is simulated with circuit imperfections to determine the effect of these when there is no feedback present. Introducing finite gain in the integrators results in harmonic distortion at the output. This harmonic distortion is a result of leakage of the internal reset signal in the integrators. By setting the gain in all three integrators to $2OSR = 42dB$, the SNR of the third order modulator sinks to $71.74dB$. The gain in the first integrator is increased to $60dB$, and the SNR raises to $84.52dB$. The first integrator is the most crucial to the performance of the modulator, as is the case for conventional $\Sigma\Delta$ ADCs.

The circuit is also simulated with capacitance mismatch and comparator offset in the modulo integrator. These two imperfections results in the same error - the output voltage from the integrator differs from the ideal case. Simulations show that the total voltage error should be significantly less than $0.5V_{LSB}$ to obtain the noise shaping. If the integrator output error is too large, the noise shaping will totally disappear.

In general, it has been proved that the OLSD modulator with modulo integrators works as intended, the quantization noise is shaped like in conventional $\Sigma\Delta$ modulators. The modulator is very sensitive to capacitor mismatch and parasitics. The effect of these capacitor imperfections will increase as the quantizer resolution increase, because the error will cover more units of $V_{LSB}$. It is important to minimize these capacitor effects, as increased quantizer resolution will allow a greater input signal swing.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

There are a lot of different ways to perform Analog-to-Digital conversion. A very popular architecture is the $\Sigma\Delta$ converter. $\Sigma\Delta$ converters use oversampling and feedback through a loop filter to shape the quantization noise in the wanted signal band. This gives more relaxed requirements for the analog part of the circuit. A variety of different architectures have been developed, but there have been very few attempts to perform $\Sigma\Delta$ modulation without feedback. Without feedback there is no DAC, and the problems with non-linearity in the DAC is gone. This is especially important when using a multibit quantizer because the DAC has to have the same linearity as the wanted output linearity.

One attempt to perform open loop $\Sigma\Delta$ modulation is done in [9], where the integrator has been replaced with a frequency modulator. It is stated that this frequency modulator behaves as a *modulo-n* integrator, and no feedback is needed to prevent saturation. One of the conclusions in [9] is that the problem with non-linearity in the DAC is moved to making a linear frequency modulator.

## 1.2 Thesis outline

In this thesis a solution for a modulo integrator [3] is presented. The modulo integrator is described at a circuit level, and the basics of the modulo operation is explained through equations. This modulo integrator is the fundamental unit of the proposed architecture for performing open-loop $\Sigma\Delta$ modulation.

### 1.2.1 Background theory

A preliminary project was performed during the autumn of 2005. The basic theory of quantization and $\Sigma\Delta$ modulation (standard and open-loop) is revised in appendix A. If the reader is unfamiliar with analog-to-digital converting, and the open-loop architecture, it is recommended to read appendix A before reading this thesis.

### 1.2.2 Design of a practical OLSD with discrete components

To prove that the OLSD modulator with a modulo integration works, a practical modulator is designed using standard, discrete components and a computer running matlab [1].

### 1.2.3 Simulation of an ideal third order OLSD in VHDL-AMS

To further investigate the OLSD architecture a third order modulator was described and simulated using VHDL-AMS. The different building blocks were simulated to ensure correct operation, before simulating the whole modulator.

#### Modeling of errors in the third order modulator

The third order modulator was introduced to circuit imperfections by modeling some of the common imperfections at a behavioral level. The imperfections modeled is given in the following list.

- Finite integrator gain
- Capacitor mismatch and parasitics
- Comparator offset in integrators

### 1.2.4 Specifications for use in a GSM system

Finally the third order modulator was simulated with the worst case errors to see if it could fulfill the requirements in a GSM system. The basic requirement is that is has to have at least $70dB$ SNR within a $200kHz$ band ([10] and [5]).

# Chapter 2

# Notes on open loop $\Sigma\Delta$ modulation

In [12], simulations showed that the ideal OLSD ADC worked properly by the use of a modulo integrator and a modulo differentiator, with a quantizer in between. The OLSD modulator was also shown to be identical in operation because the modulo operations became *transparent*[1] under ideal conditions. The derivation of the proof can be seen in appendix A.4.

In [12], the simulation results also showed that the noise shaping in the OLSD modulator disappeared if the quantizer resolution, $n$, was equal to or less than the order of the modulator. As soon as $n$ got larger than the modulator order, the noise shaping came back (also dependent on the input signal level). An answer to why this happens may be found in [7]. Here it has been calculated that for a $2_{nd}$ order non-feedback $\Sigma\Delta$ DAC there is five possible output levels, hence a resolution of three bits is necessary. It is also stated that it is possible to use a resolution of two bits if one reduces the input swing and adds a little bias.

By using 2.1 and 2.2 it is shown in [7] that the output from a second order non-feedback $\Sigma\Delta$ DAC is as in equation 2.3.

$$y_n = q(\sum_{k=0}^{n}(\sum_{l=0}^{k-1}(x_l - y_l) - y_{k-1})) \tag{2.1}$$

$$q(x) = x - mod_{2^N}(x) \tag{2.2}$$

$$y_n = x_{n-1} - (mod_{2^N}(\sum_{k=0}^{n}\sum_{l=0}^{k-1} x_l) - 2mod_{2^N}(\sum_{k=0}^{n-1}\sum_{l=0}^{k-1} x_l) + mod_{2^N}(\sum_{k=0}^{n-2}\sum_{l=0}^{k-1} x_l)) \tag{2.3}$$

By setting

$$S = \sum_{k=0}^{n-2}\sum_{l=0}^{k-1} x_l \tag{2.4}$$

the $mod_{2^N}$ functions in equation 2.3 may be substituted by

[1]The modulo differentiator cancel out the reset from the modulo integrator, as if no reset happened at all.

$$
\begin{aligned}
mod_{2^N}(S) &= S & (2.5)\\
mod_{2^N}(S) &= S + 2^N & (2.6)\\
mod_{2^N}(S) &= S + 2^{N+1} & (2.7)
\end{aligned}
$$

It is then shown that $y_n \in [0, 2^N, 2 * 2^N, 3 * 2^N, 4 * 2^N]$ using the above results.

This is an understandable result when working with DACs, but in the ADC, things get a bit more complicated due to the infinite amount of possible input levels. If the output from the modulo integrator is reset, and the resulting output voltage is in the same range as the previous output voltage (for example $\frac{V_{ref}}{2} + V_{ref} = \frac{3V_{ref}}{2} - V_{ref} = \frac{V_{ref}}{2}$, the quantizer may end up quantizing those two voltages to the same value. The result from the differentiation will then be zero, even though there was a input voltage larger than zero that caused the integrator to reset. This means that when a high input voltage causes a reset in the integrator, there is a possibility that the resulting voltage will be quantized to the same value. It is therefore necessary with greater resolution in the quantizer to separate two close levels, or one can reduce the input signal swing. It is however impossible to avoid the exact situation where two adjacent samples are exactly the same value, but this may be assumed to be rare for a *busy* and non-zero input signal.

# Chapter 3

# The modulo integrator

The modulo integrator [3] is the fundament of the proposed open-loop ΣΔ ADC. In [9], a open-loop modulator has been constructed by the use of a frequency modulator instead of a modulo integrator. It is however stated that the frequency modulator behaves as a modulo integrator, and that the non-linearities in the frequency modulator adds directly to the signal. By constructing a modulo integrator with switched-capacitor technology it should be possible to obtain better linearity than the frequency modulator.

## 3.1 Description of the modulo integrator

The modulo integrator is an integrator that resets and keeps the remainder if the output voltage exceeds a specified limit. For a single ended integrator, as the one that is proposed here, two basic operations are needed to perform the modulo operation, subtraction of two voltages and detection of the overloading of the integrator. The latter operation can easily be implemented by a comparator, but the subtraction is a bit more difficult. If the maximum value of the output is named $V_r$, we need to add the voltage $-V_r$ to the output, and then start integrating from that new value. Hence, the circuit needs to remember the last voltage after reset. This is difficult when working with voltages and the circuit would become rather complicated.

After conversations with [4], a method to perform modulo integration was derived. In a common switched-capacitor integrator the output voltage is a result of the charge stored in the integration capacitor. This capacitor holds the previous values of the input samples, hence no voltage needs to be remembered. When the integrator overloads, the output voltage is not able to reach the desired value, and one would think that signal information is lost. This is not necessarily true since the information is stored in the integration capacitor. To perform the reset and keep the remainder, one needs a reset circuitry which triggers when the output voltage exceeds the specified limit, and drains a charge that corresponds to the full scale voltage from the integration capacitor. The output voltage will then drop by $V_r$, and the remaining voltage is kept.

The schematic of the proposed modulo integrator is shown in figure 3.1 (based on figure 3 in [3]). The reset circuitry is shown inside the dotted box. An extra non-overlapping clock is needed to be sure that the reset happens at a desired time, not interrupting the normal operation of the integrator. During $\phi_1$, capacitor $C_3$ is charged to $V_{ref}$, with a reverse polarity in respect to $C_2$. When the comparator triggers, the charge stored in $C_3$ is subtracted from $C_2$, leaving the remaining charge behind. It is very important that $C_2$ and $C_3$ are well matched for this operation to be accurate.

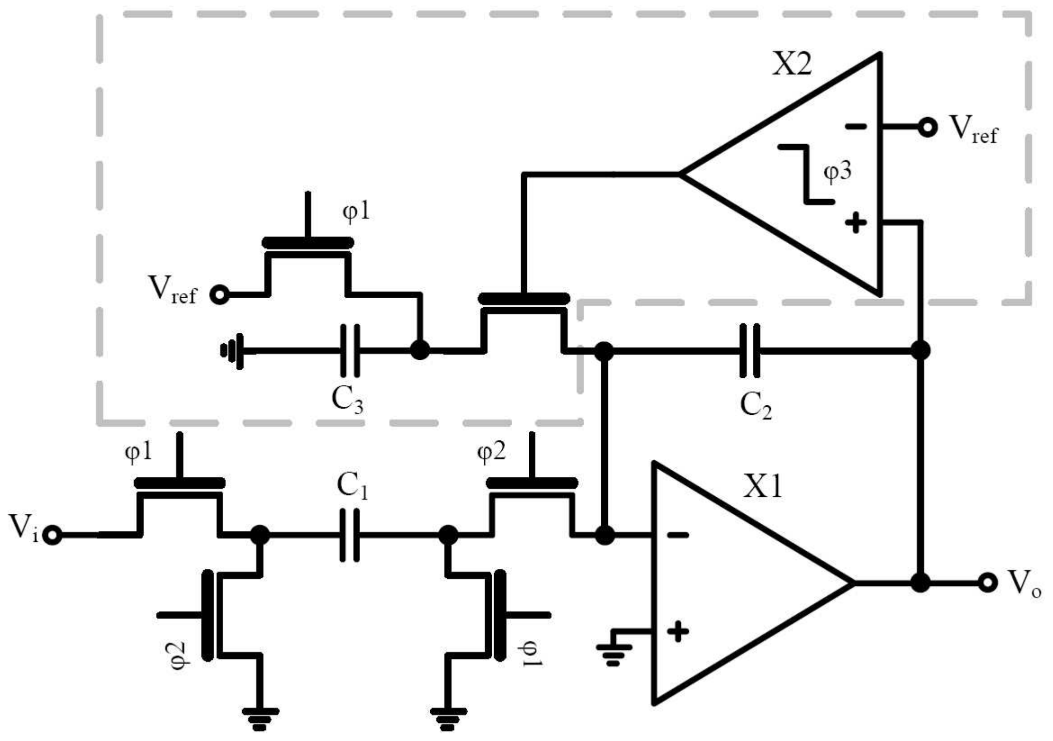


Figure 3.1: Schematic of the proposed modulo integrator

## 3.2 Behavior of the modulo integrator

From figure 3.1 one can see that the integrator itself (apart from the reset circuit inside the dotted box) is a non-inverting delaying integrator (pp404-405 in [8]), with a transfer function as given in equation 3.1 (equation 10.21 in [8]).

$$H(z) = \frac{V_o(z)}{V_i(z)} = \left(\frac{C_1}{C_2}\right)\frac{1}{z-1} \tag{3.1}$$

This means that as long as no reset occurs, the modulo integrator behaves identically to a normal non-inverting delaying integrator. This is also true after a reset, but the charge in capacitor $C_2$ will then be changed. The charge in $C_2$ and $C_3$ right before a reset occurs, is given in equations 3.2 and 3.3.

$$Q_2 = C_2 V'_{out} \tag{3.2}$$

$$Q_3 = C_3 V_{ref} \tag{3.3}$$

When the comparator triggers, the switch opens and connects the negative node of $C_2$ to the positive node of $C_3$. The result is that the charge in $C_2$ will decrease as given in equation 3.4.

$$Q_{2,after\ reset} = C_2 V'_{out} - C_3 V_{ref} = C(V'_{out} - V_{ref}) \tag{3.4}$$

which implies that the output voltage after the reset is equal to $V'_{out} - V_{ref}$.

If the maximum allowed input voltage is set to $V_{ref}$ (maximum signal swing without corrupting the operation of the modulo integrator), the situation where both the output voltage and the incoming sample are infinitesimally smaller than $V_{ref}$ may occur. This would ideally result in an output voltage of about $2V_{ref}$, and the reset operation would bring it back to a level below $V_{ref}$. This requires a supply voltage of $2V_{ref}$, otherwise the output voltage will saturate too early, and the charge transfer from $C_1$ to $C_2$ will not be able to complete. This is not very favorable when working with low voltage CMOS processes and wanting as high signal swing as possible.

In figure 3.1 the output of the comparator controls one switch only. This switch let the charge flow between $C_3$ and $C_2$. If the charge transfer from $C_1$ to $C_2$ did not complete, as mentioned earlier, the reset would bring the output voltage to a lower level than wanted causing errors in the modulo operation. To solve this problem it is necessary to let the comparator output control the switches clocked by $\phi_2$. This enables $C_1$ to complete its charge transfer at the same time as the reset occurs. By doing this, the supply voltage may be lowered without risking loss of signal information. The supply voltage still have to be slightly higher than the reference voltage to be sure that the comparator triggers even if offset and other non-idealities are present.

### 3.2.1 Non-idealities in the modulo integrator

As all other circuits, the modulo integrator will have non-ideal effects in it. It will have all the effects known from the standard switched-capacitor integrator, and in addition the non-ideal effects from the reset circuitry. The integrator itself, without the reset circuitry will have parasitic capacitances on both sides of $C_1$ and $C_2$. These will not affect the integrator, as described in pp405-406 in [8]. As seen in figure 3.2 the reset capacitor $C_3$ will have two parasitic capacitances, coming from the capacitance between the bottom plate and the substrate, and the metal contacts for the top plate and the substrate. By having the bottom plate grounded, the parasitic capacitance can be disregarded. This is the largest parasitic capacitance (up to approximately 20% of C(p395 in[8])). The top plate parasitic can not be disregarded as this capacitance is charged to $V_{ref}$ during $\phi_1$. When the integrator resets, the charge in the top plate parasitic will be subtracted from $C_2$, introducing an error on the output. The size of the top plate parasitic is about 1-5% of C (p398 in [8]). Assuming the worst case where the parasitic is about 5% of C, the output voltage after a reset would be 5% lower than for the ideal case. Depending of the resolution in the quantizer, this would possibly be quite critical for the operation of the ADC. The higher resolution, the more units of $V_{LSB}$ would be covered by the fault. If the integrator output has an error large enough to trigger a change at the quantizer output, a *false negative* may occur. This can happen if the parasitic capacitance is too large. When a false negative occurs, the modulo differentiator will believe that the input voltage was much higher than it actually was - introducing a severe error at the output of the modulator.

The mismatch between $C_2$ and $C_3$ is also a factor that needs to be kept in mind. This mismatch will result in the same fault that the parasitic capacitance causes. The output voltage after a reset will be different from the ideal case. It is therefore very important to match capacitors in addition to reducing the parasitic capacitances.

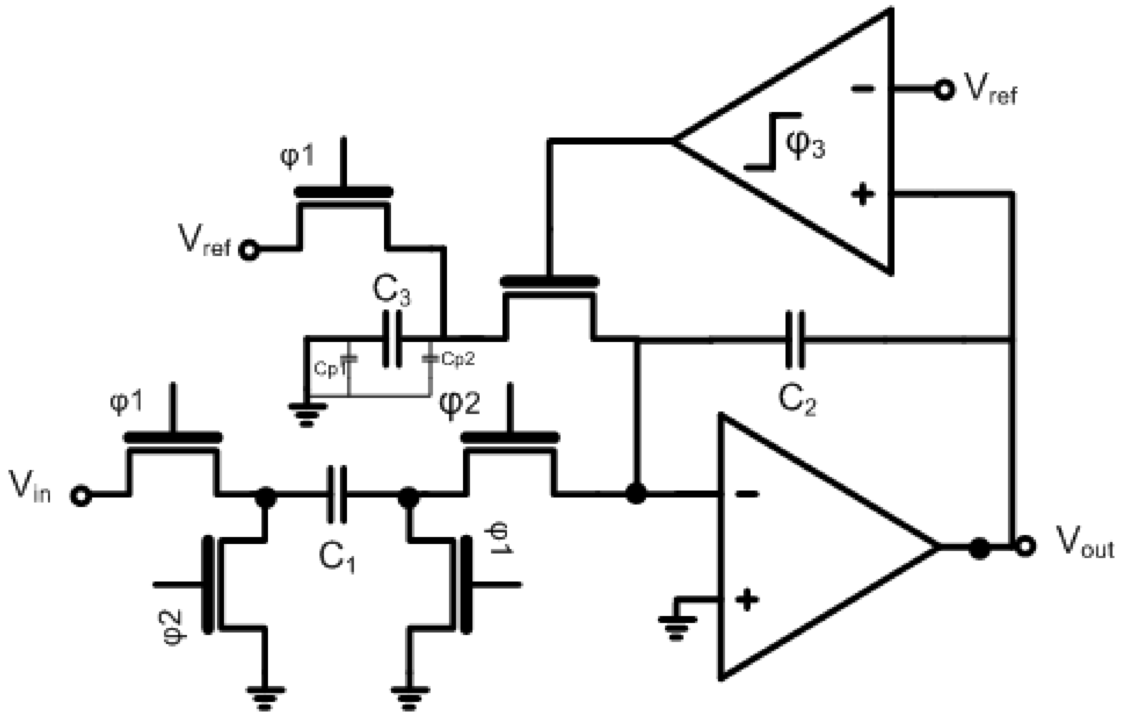


Figure 3.2: Schematic of the proposed modulo integrator with parasitic capacitances

The gain of the integrator is a factor that needs to be thought of during the design. When having finite gain in the integrator, the integrator is not able to drive the output as high as it ideally would be. Since the magnitude of the error will depend on the output voltage, it will in some way be correlated to the input voltage and thus causing distortion.

# Chapter 4

# Practical first order modulator

A practical first order open loop ΣΔ-ADC was constructed to test the open-loop theory in practice. The analog part was made of regular discrete components while the quantizer and the digital part consisted of a microcontroller with an internal ADC. The design also included a RS232-interface to be able to transfer data to a computer and process it in matlab.

## 4.1 Integrator

The integrator was realized as a Switched-Capacitor integrator with discrete components. The operational amplifier is a standard dual opamp, where the second amplifier is used as a comparator for the reset circuitry. The practical circuit does not clock the comparator by $\phi_3$ - the microcontroller is controlling the reset by receiving an interrupt when the comparator toggles. The microcontroller software then resets the integrator if a conversion is not in action. This is possible since the software is generating the two non-overlapping clocks $\phi_1$ and $\phi_2$ which control the integrator and quantization. The part list for the integrator is shown in table 4.1.

## 4.2 Quantizer and PC-interface

The quantizer was realized with an Atmel ATMega48 microcontroller [6]. This microcontroller features a 10-bit successive approximation ADC, where one may choose to use fewer bits by performing right shift operations on the conversion result. The reference voltage may be chosen from an internal $1.1V$ reference or the analog power supply (typically 5V).

The PC-interface use the built in RS232-interface of the AVR microcontroller. The circuit continuously transfer the sampled values as they are finished.

| | |
|---:|---:|
| Operational amplifier | Motorola MC34072 dual opamp (#1) |
| Comparator | Motorola MC34072 dual opamp(#2) |
| Switches | Maxim MAX392 integrated switches |
| Capacitors | General surface mount capacitors (20% tolerance) |
| Power supply | $+10V$ lab supply and decoupling capacitors |
| Printed circuit board | FR4 expoxy laminated circuit board |

Table 4.1: Part list for the modulo integrator

## 4.3 Differentiation in Matlab

To be able to maximize speed, the microcontroller only performs quantization. Every sample is transmitted to matlab where the modulo differentiation is executed. A simple matlab script receives the samples and put them into arrays. When the circuit has finished transmitting samples, the script performs the modulo operation on these arrays. The block diagram of the realized open loop $\Sigma\Delta$ ADC is shown in figure 4.1. The complete schematic can be seen in Appendix C and the software for the microcontroller and matlab can be found in appendices D.1 and D.2. Note that the complete schematics has a lot of wire connections by node names to simplify the drawing.

## 4.4 Measuring method and results

### 4.4.1 The modulo integrator

The modulo integrator was put in a test bench to confirm correct modulo operation. The input was a DC signal at $50mV$ and $100mV$. The results can be seen in figure 4.2 where screen shots from the oscilloscope is presented. It is clear that the output never exceeds the reference voltage, which was set to $1.1V$. The output is not zero after a reset, it is the value that would have been above the reference if the reset did not occur. Note that the reset happens more often when the input voltage is raised to $100mV$.

### 4.4.2 The complete modulator

The open loop converter was put into a test bench with an adjustable power supply, a signal generator and an oscilloscope. For every test the results from the standard quantizer was compared to the result from the open-loop modulator. Every test performed here was run for $2^{14}$ samples.

In figure 4.3 the output frequency spectrum is shown for the standard quantizer with four bits resolution, and for the OLSD modulator. The input voltage was $0.1 + 0.05sin(2\pi f)$, where $f = 5Hz$. The reference voltage was $1.1V$ and the sampling frequency was $f_s = 487Hz$. The noise floor in the two cases both lie on about $-40dB$, but the high frequency noise was shaped a bit in the OLSD modulator. The general noise in the circuit was dominating in the pass band though.

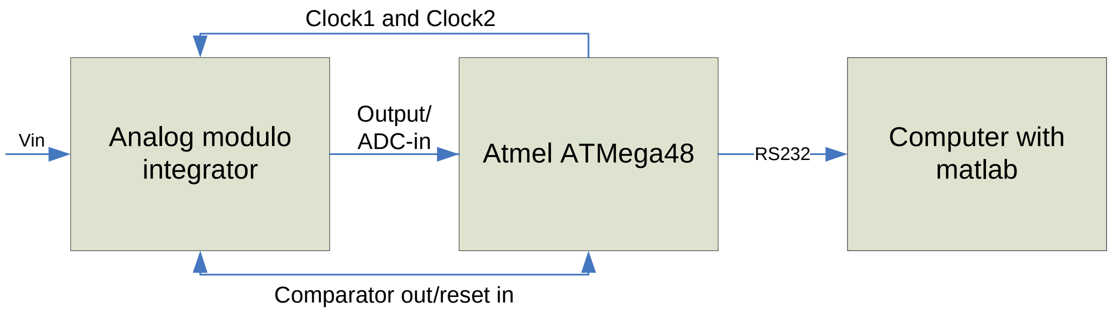


Figure 4.1: Block diagram of the open loop $\Sigma\Delta$ ADC

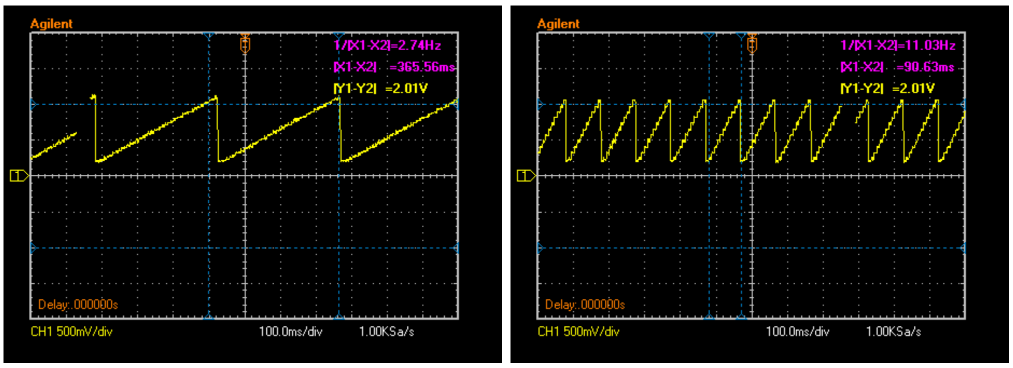


Figure 4.2: Left: Output from the modulo integrator with 50mV input. Right: Output with 100mV input.

In figure 4.4 the two spectrums are shown for a sampling frequency of $122Hz$. The input signal level, frequency and quantizer resolution was the same as in figure 4.3. This time the noise in the OLSD modulator has a more distinct shaping. In figure 4.5, the same spectrum as in figure 4.4 is shown with markings for easier readout of the magnitudes. From the figure one sees that the slope of the noise is approximately $20dB$ per decade, which is the expected value for a first order modulator.

In general, the modulator was very sensitive to different input voltages and quantizer resolution. For high sampling frequencies the noise floor raised quickly. To really see an improvement in SNR from the OLSD, the quantizer resolution had to be set to four bits or less. Otherwise the quantization noise was buried in other noise sources. With the right parameters the OLSD modulator outperformed the standard quantizer, although the performance increase was not very significant. One major source of error was the capacitors used in the modulo integrator. These were surface mount ceramic capacitors with a tolerance of 20%. As mentioned in chapter 3, the capacitor matching can be quite critical for the operation of the integrator.

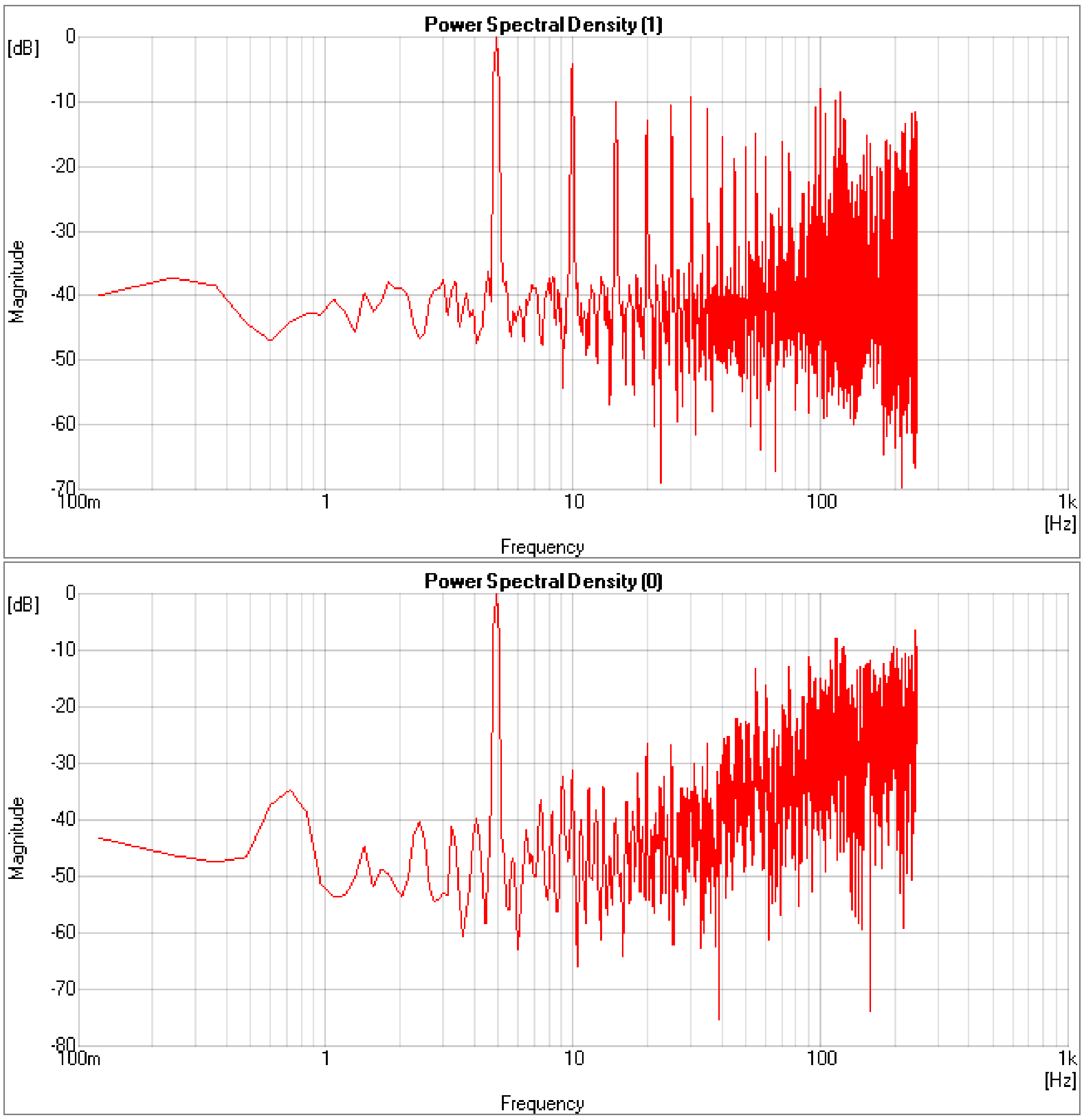


Figure 4.3: Output frequency spectrum of the standard quantizer (top), and from the open loop modulator (bottom). $F_0 = 5Hz$, $F_s = 487Hz$.

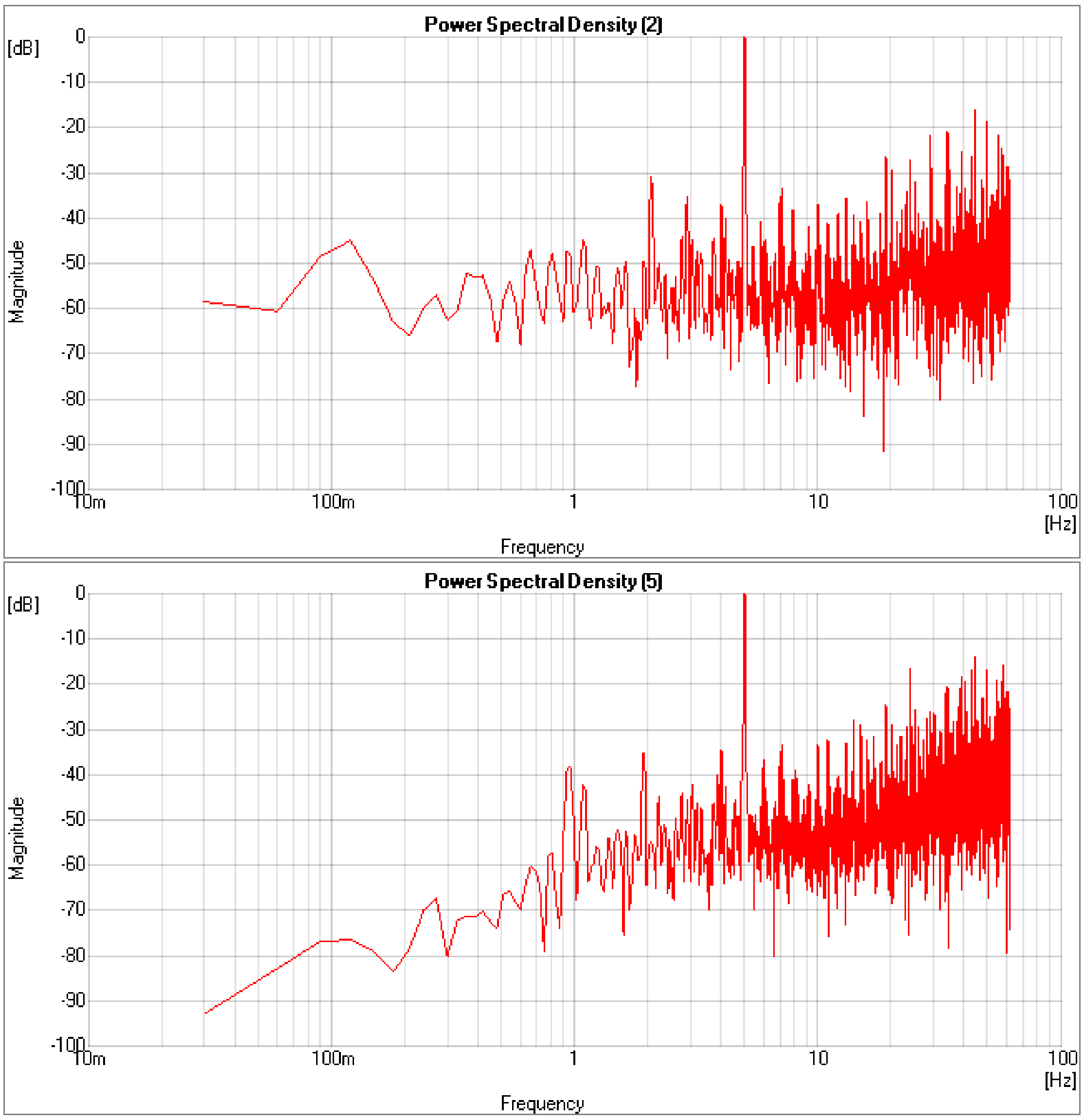


Figure 4.4: Output frequency spectrum of the standard quantizer (top), and from the open loop modulator (bottom). $F_0 = 5Hz$, $F_s = 122Hz$

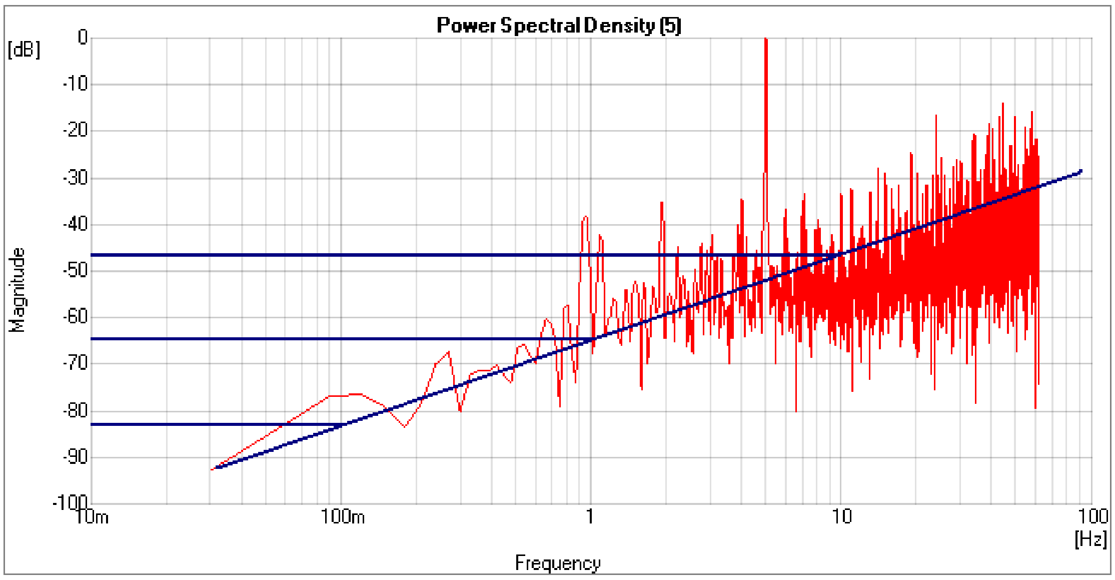


Figure 4.5: Frequency output spectrum with markings for easier readout.

# Chapter 5

# Simulations in VHDL-AMS

The ADC was simulated at a behavioral level in VHDL-AMS, using Advance MS [2]. Each module was implemented in such way that it easily could be replaced by a low level module later. The behavioral model does not take much advantage of VHDL-AMS ability to simulate analog circuitry together with digital. As the first switch in the first integrator samples the signal, everything is in discrete time - and the behavior of the circuit is pure discrete. However, if one should implement the modules at circuit level at a later time it is necessary to simulate them as analog circuits.

## 5.1 The integrator

The modulo integrator is in fact modeled in the same way as a digital accumulator with reset. The only difference is that the signal is represented by floating point numbers instead of integers.

### 5.1.1 Adding non-idealities to the integrator

Introducing non-idealities in the integrator is an important factor when determining the performance of the ADC in a real circuit. The integrator was therefore modeled with adjustable gain and comparator offset. Capacitance mismatch and parasitics were also modeled.

#### Integrator gain

To model the gain of the integrator, the output voltage is multiplied with a constant such that $V_{out} = V'_{out} * k$. The constant is given in equation 5.1.

$$k = \frac{1}{1 + \frac{1}{10^{\frac{gain}{20}}}} \tag{5.1}$$

where *gain* is the dc-gain in decibels.

#### Comparator offset

The comparator offset is fairly easy to implement in the behavioral model. The comparator is implemented as a simple if-then-else case, and adding offset means adding offset to the argument of the if case.

**Capacitor mismatch and parasitics**

The capacitance non-idealities are only modeled for the reset-circuit, since this is the only new and unknown element of the integrator. When the integrator resets, the matching between $C_2$ and $C_3$ in figure 3.1 is very important. Modeling of the mismatch between these two capacitors is done by subtracting a voltage different from $V_{ref}$, when the reset occurs. The same operation is done when adding the parasitic capacitance mentioned in chapter 3.2. The effect of these two non-idealities is the same - the charge removed from $C_2$ is different from $Q = CV_{ref}$.

## 5.2 The quantizer

The quantizer is modeled as a standard flash quantizer. (The model is generic, meaning that the number of bits is easily manipulated. This makes it better suited for simulations where one tries to find the optimal configuration of the ADC.) In figure 5.1, a 3-bit flash ADC is shown. Note that there are 8, or $2^N$ comparators.

## 5.3 The differentiator

The operation of the differentiator is quite simple when using unsigned representation of the numbers. The behavior of the differentiator can be described by the following pseudo code.

```
difference = (current sample - previous sample);
if(difference > modulo mask)
    output = (difference AND modulo mask) -1
else
    output = difference;
end if;
```

The *modulo mask* in the code above is simply an array of ones, the length of $(2^N - 1)$. The subtraction is performed by 1's complement, that is $x - y = x + not(y)$. If the result exceeds the modulo mask, the $(2^N - 1)$ lowest bits, minus one, is set to the output. Otherwise the difference is set directly to the output.

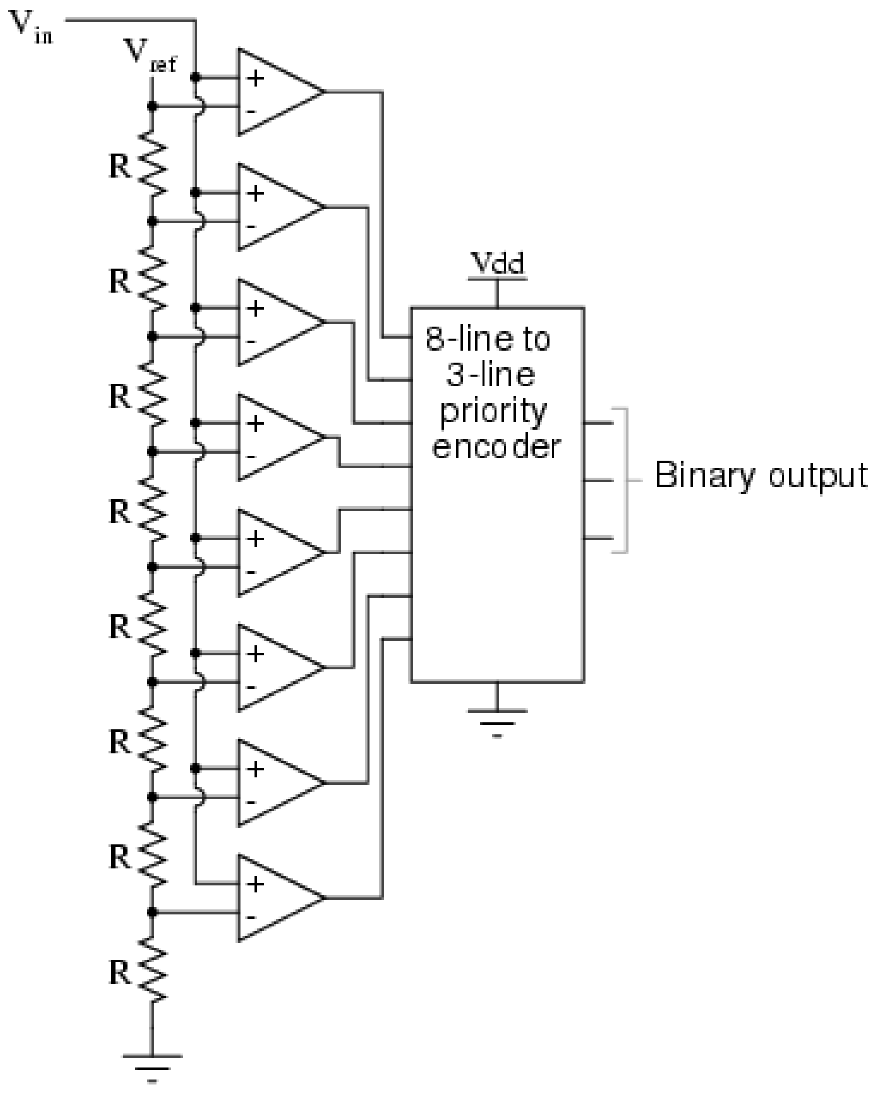


Figure 5.1: Schematic for a 3-bit flash ADC

# Chapter 6

# Simulations results and discussion

The different building blocks of the OLSD ADC was simulated in VHDL-AMS. The output spectrums, SNR, SNDR and ENOB were all calculated with SdnReport, which is a part of SystemDotNet [11].

## 6.1 Block simulations

### 6.1.1 Integrator

The allowed input range of the modulo integrator is $V_{in} \in [0, V_{ref} >$, and it was simulated with different DC and AC voltages to confirm correct modulo operation. In figure 6.1 the input voltage is $0.1\, V_{ref}\ DC$. It is clear that when the output voltage exceeds $V_{ref}$ it is reset to the remainder, which in the figure always is $0.1V_{ref}$. In figure 6.2 the input voltage is raised to $0.5\, V_{ref}\ DC$. In this case the output rise faster, and resets more often. The figure shows that the integrator resets in some cases where $V_{acc} + V_{in} = 1.0$, and in some cases it does not. This may be due to errors in the simulation tool since the two cases of $V_{out} = V_{ref}$ may cause both a reset or not.

In figure 6.3 the input voltage is set to $0.5V_{ref} + 0.5V_{ref}sin(\omega)$. In this case the output becomes more complex due to the more complex nature of the input signal. It is however clear that the output never exceeds the reference voltage. Note that the slope of the output is strongly correlated to the input signal.

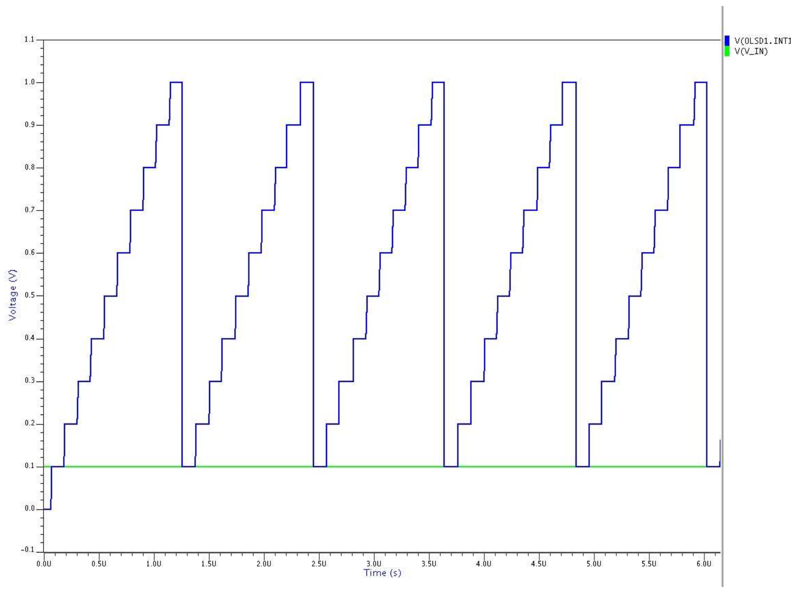


Figure 6.1: Output from modulo integrator (blue) and $0.1V_{ref}$ input (green)

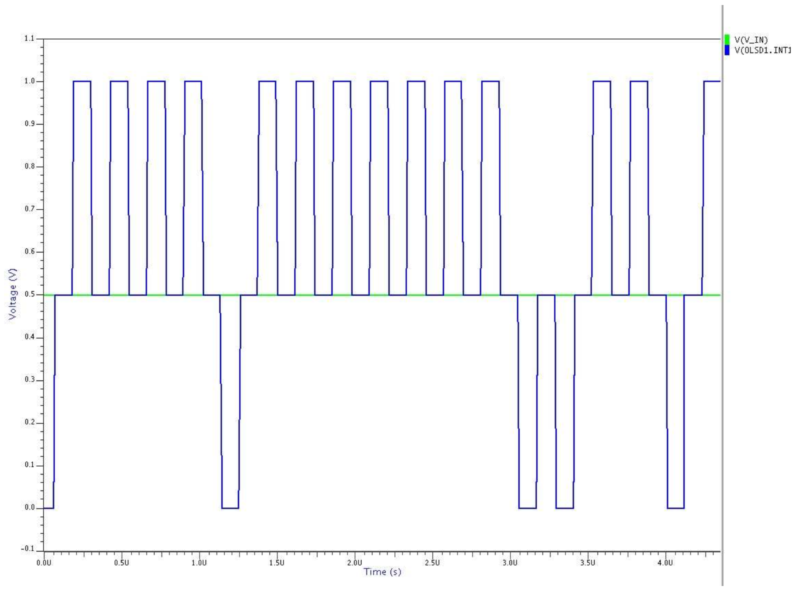


Figure 6.2: Output from modulo integrator (blue) and $0.5V_{ref}$ input (green)

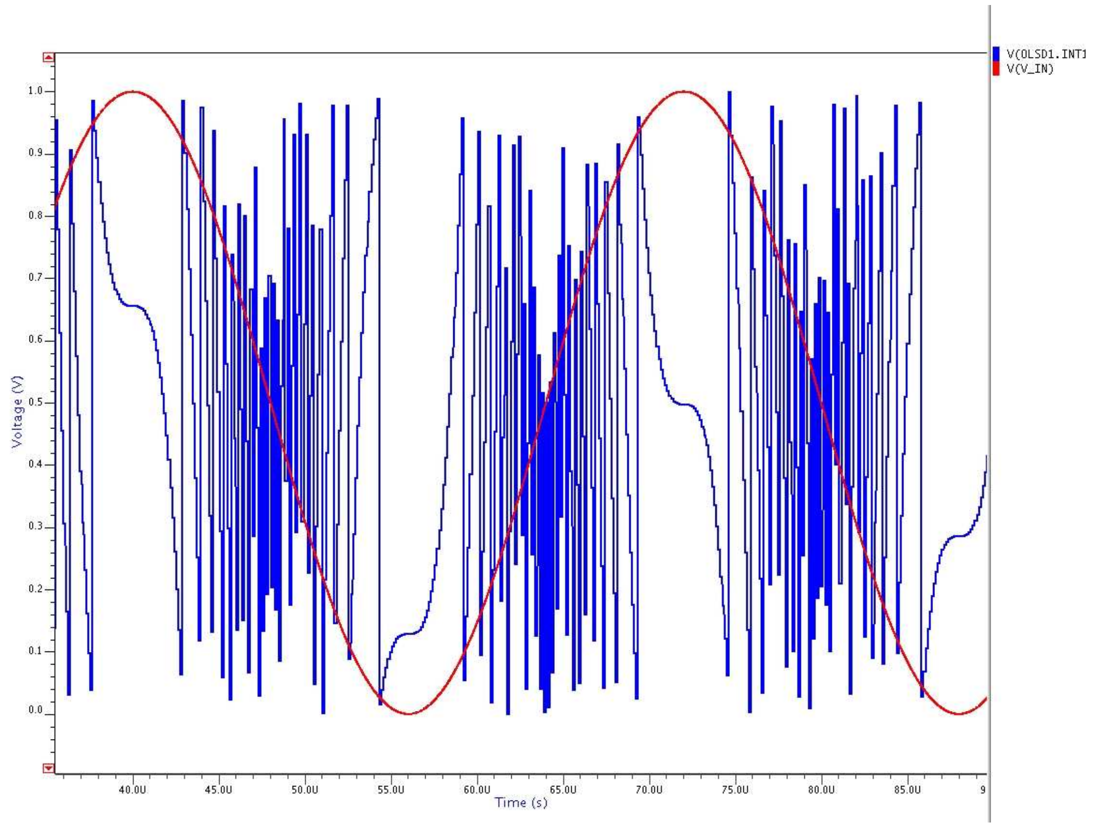


Figure 6.3: Output from modulo integrator (blue) and $0.5V_{ref} + 0.5V_{ref}sin(\omega)$ input (red)

### 6.1.2 Quantizer

The quantizer was fed with a ramp that started from $0V$ and rose evenly up to, and above the reference voltage. In figure 6.4 the result is shown for a resolution of four bits. Note that the input voltage in the figure is multiplied by $2^N - 1$ to easily show the relation to the output. Each output step corresponds to a voltage of $V_{LSB} = \frac{V_{ref}}{2^N} = 0.0625V$, or $0.9375V$ in the figure. In figure 6.5 the output frequency spectrum is shown. It is clear that the white noise assumption is not true, as the noise floor is far from flat. The obtained SNR and SNDR for the bandwidth of $2F_0$ was $54.96dB$, which corresponds to an ENOB of 8.84 bits. The theoretical SNR would be $47.12dB$ according to equation A.9 (quantization with oversampling). The difference is quite large, but if one calculates the SNR for the range from $0 - \frac{f_s}{2}$ the SNR is $25.8dB$, which corresponds very well to equation A.7 (quantization in a nyquist rate converter). This equals an ENOB of 3.99 bits, which is very good when the actual number of bits simulated 4.

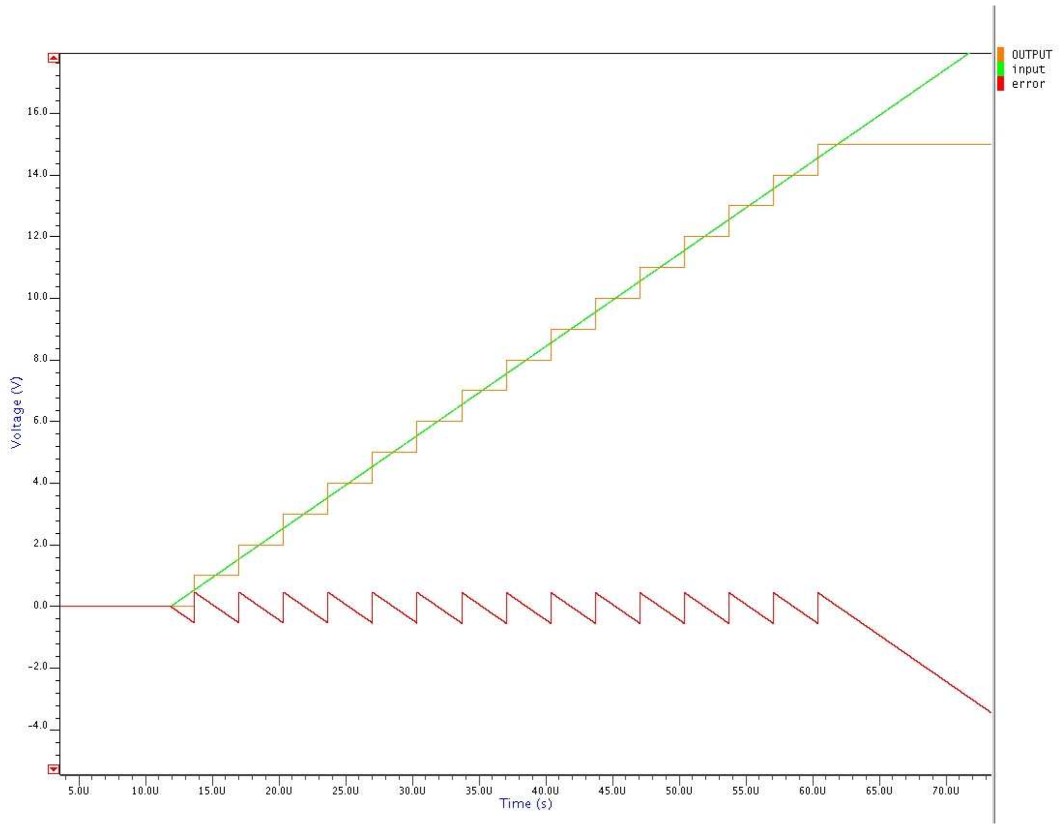


Figure 6.4: Quantizer input (green), output(orange) and quantization error (red)

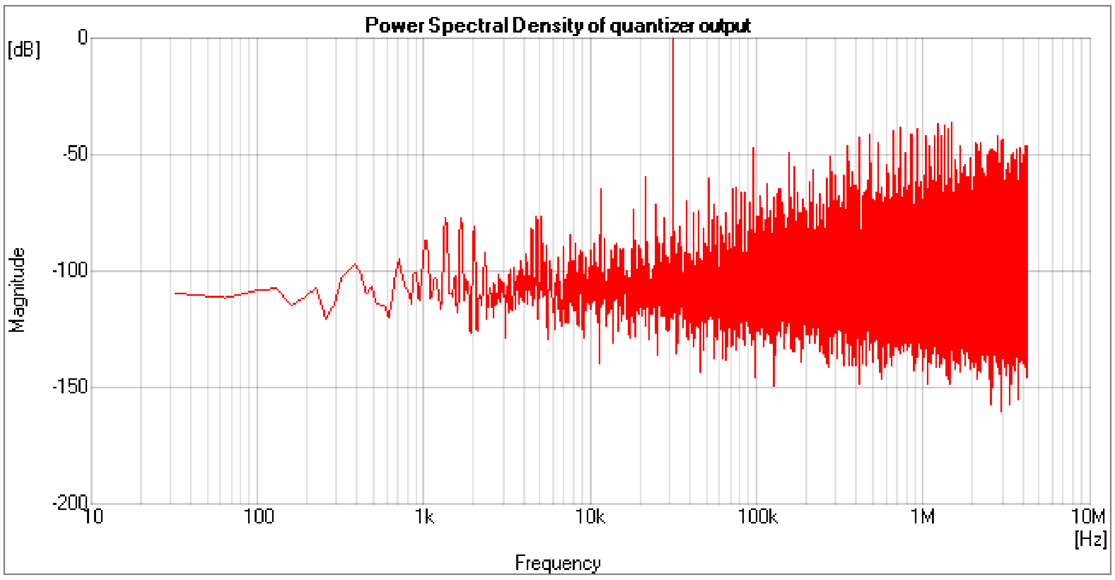


Figure 6.5: Output spectrum for the quantizer. $F_s = 2^{23}$ and $F_0 = 31.232 * 10^3$)

### 6.1.3 Differentiator

The modulo differentiator was first simulated with an input similar to the output from the quantizer when feeding it with a ramp. The result is shown in figure 6.6. It is clear that for each input step the output is equal to the step size. When the input is constant the output is zero.

In figure 6.7 the modulo differentiator was fed with an input signal that corresponds to the output of the modulo integrator when having a constant input voltage. The constant voltage in this case results in a step size of 4 on the output of the four bit quantizer. It is clear from the figure that the differentiator retrieves the constant value of 4 from the input signal.

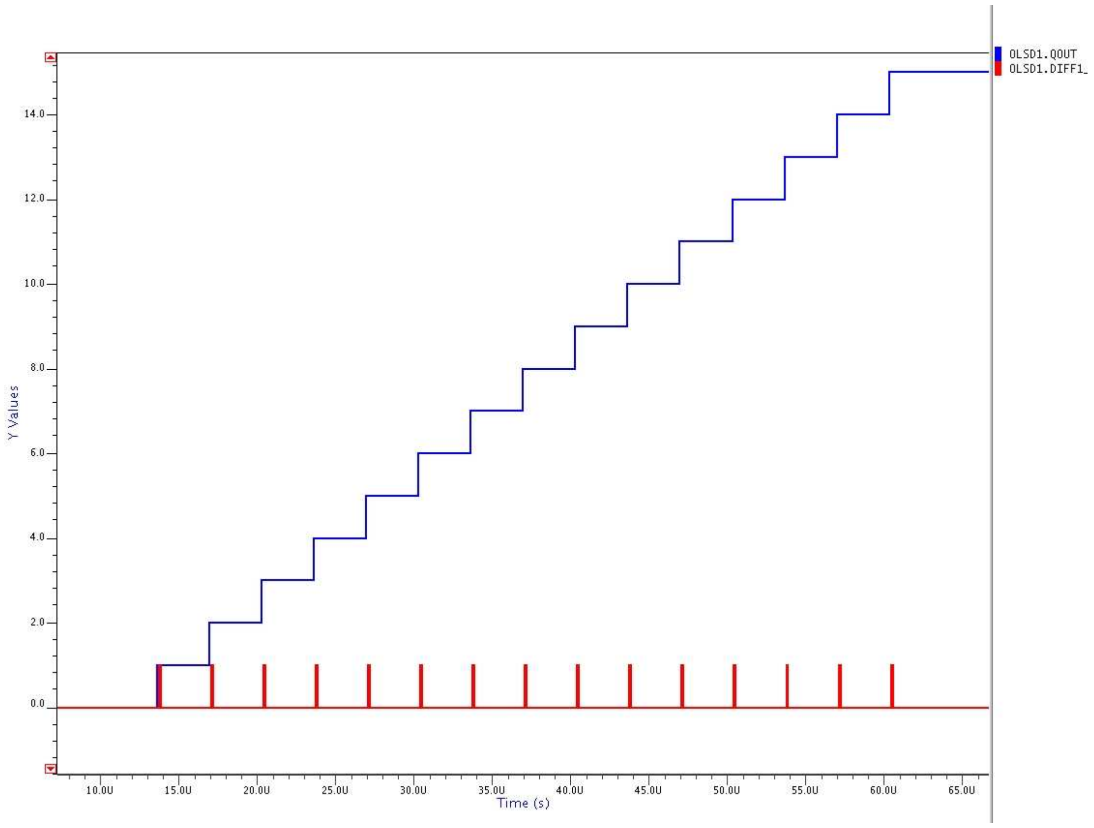


Figure 6.6: Output from the differentiator with a ramp input

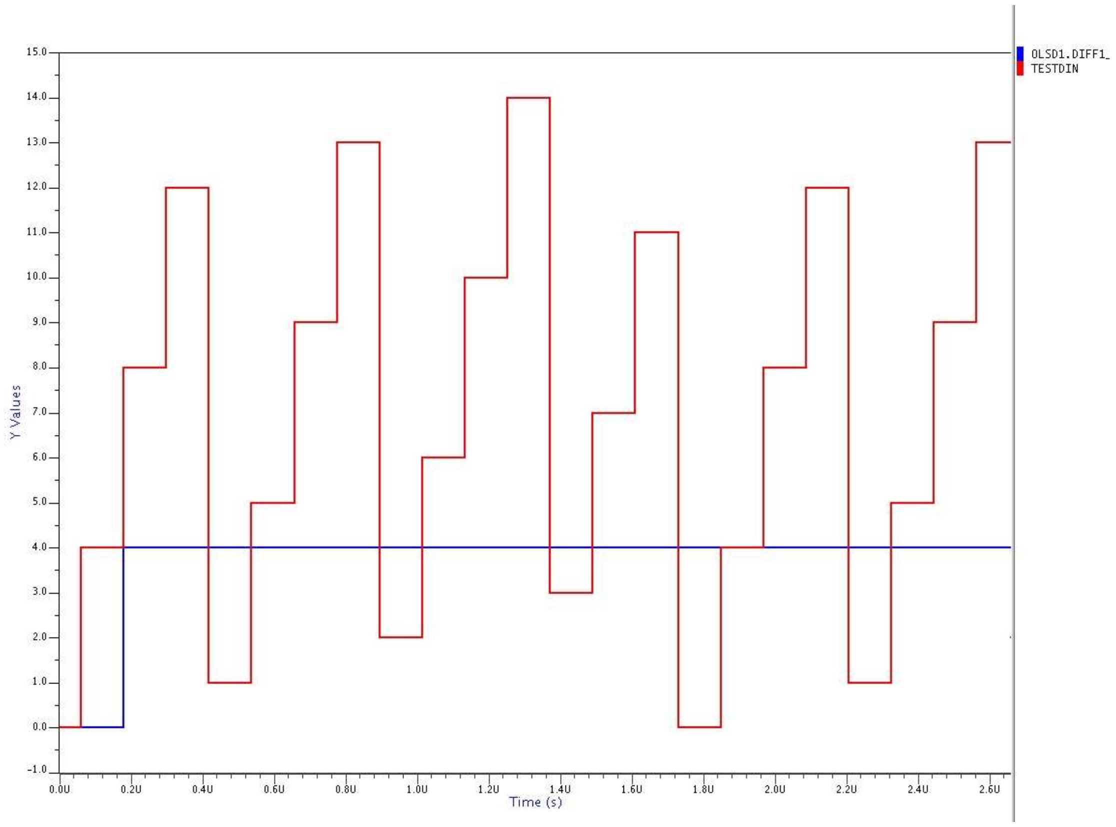


Figure 6.7: Output from the differentiator with a input that corresponds to the output from the modulo integrator with constant input voltage

## 6.2 Ideal system simulations

A third order OLSD ADC made up of the components described earlier was simulated. The goal was to achieve a SNR of at least $71dB$ within a $200kHz$ band, to meet the specifications for an ADC in a GSM system ([10] and [5]). The simulations started with a low input level, rising the level until the noise shaping disappears, to find the maximum input range.

Figure 6.8 shows the output of the third order, four bit modulator for the input voltage with a DC-level of 0.3 and an amplitude of 0.1. The signal frequency is $f = 31.25kHz$ and the sampling frequency is $f_s = 2^{23}Hz = 8.388608MHz$. The obtained SNR was $82.55dB$. It is however desirable to have as much signal power as possible to obtain a good SNR, so the amplitude of the input signal was risen. The result for the input signal $V_{in} = 0.3 + 0.2sin(2\pi f)$ is shown in figure 6.9. Unfortunately, the noise shaping has disappeared. The noise floor is really flat, and the obtained SNR was only $15.34dB$. From the time domain view, one sees that the sinusoidal output has some glitches that rise up from the otherwise low level, up to almost maximum. As soon as these glitches start to appear, the noise shaping disappears. The other output values do also differ, but not as extreme as the glitches.

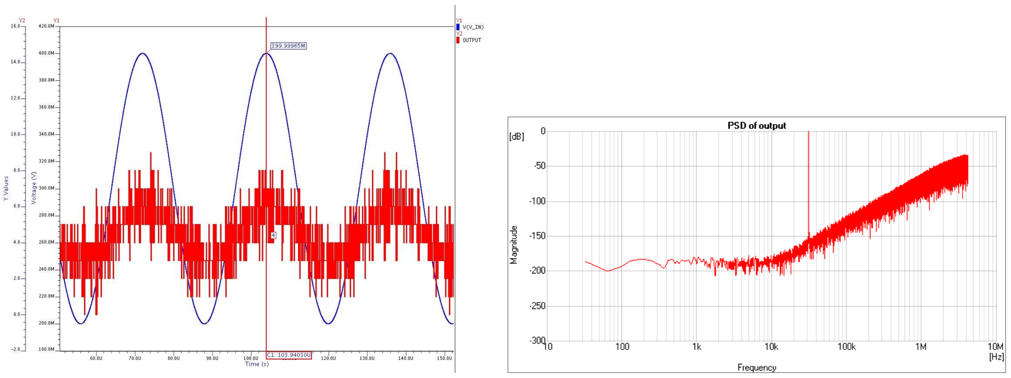


Figure 6.8: Left: input (blue) $=0.3 + 0.1sin(2\pi f)$ and output (red). Right: Output frequency spectrum

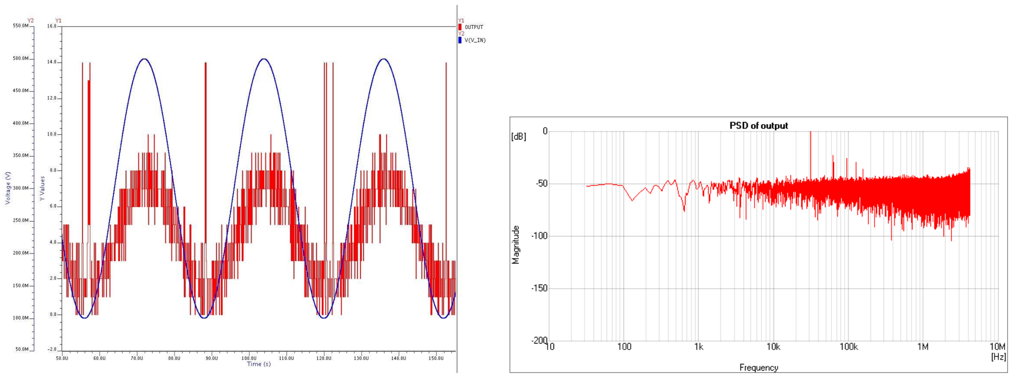


Figure 6.9: Left: input (blue) $=0.3 + 0.2sin(2\pi f)$ and output (red). Right: Output frequency spectrum

In figure 6.10 the input DC-level was risen, and the input AC-level was kept at $0.2V$. As one can see from the figure, the noise shaping is back, and the glitches has disappeared. It seems as if the modulator noise shaping is dependent on the absolute value of the input signal, more than the amplitude disregarding the DC-level. Then what happens if one increases the amplitude one step further, to a total input level of $0.5 + 0.3sin(2\pi f)$? The result is shown in figure 6.11. Once again the noise shaping has disappeared, and the glitches are back. This time there are no glitches from below, but the glitches occur when the input is high. The glitches are almost always samples with zero magnitude. This is what was called a *false negative* in chapter 3. When the input is high, the integrators reset more often, and the possibility for equal (within the same decision area in the quantizer) samples increase. Two equal samples result in a zero output in the last differentiator.

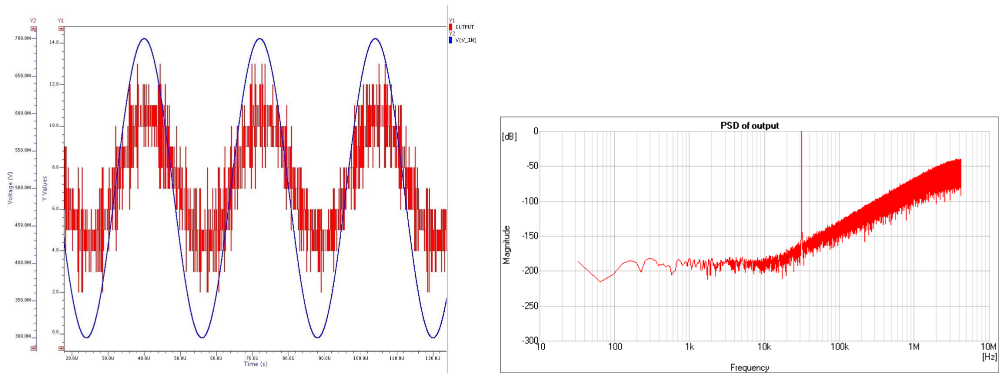
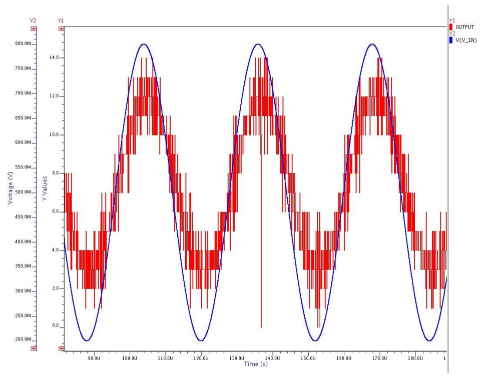


Figure 6.10: Left: input (blue) $=0.5 + 0.2sin(2\pi f)$ and output (red).Right: Output frequency spectrum for input $=0.5 + 0.2sin(2\pi f)$



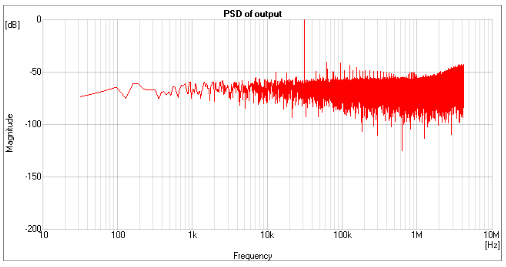


Figure 6.11: Left: input (blue) $=0.5+0.3sin(2\pi f)$ and output (red).Output frequency spectrum for input $=0.5+0.3sin(2\pi f)$

## 6.3 System simulations with circuit imperfections

The third order OLSD was simulated with circuit imperfections to determine the requirements for the analog circuitry. The imperfections were modeled as described in chapter 5.

### 6.3.1 Finite integrator gain

In page 564 in [8] it is stated that the opamp gain of a switched-capacitor integrator in a ΣΔ modulator should be larger than $\frac{OSR}{\pi}$, and that designers typically will ensure that the opamp gain is as least twice as large as the OSR. With this in mind, the opamp gain was first set to $2OSR$, to check how much the performance degraded compared to the ideal modulator. The finite gain was first introduced in the first integrator, then in the second one, and in the end all three integrators got the same gain. The result from the three simulations is shown in table 6.1. In figure 6.12, the output frequency spectrum for the ADC is shown when the first integrator has $42dB$ gain, and when all three integrators have $42dB$ gain.

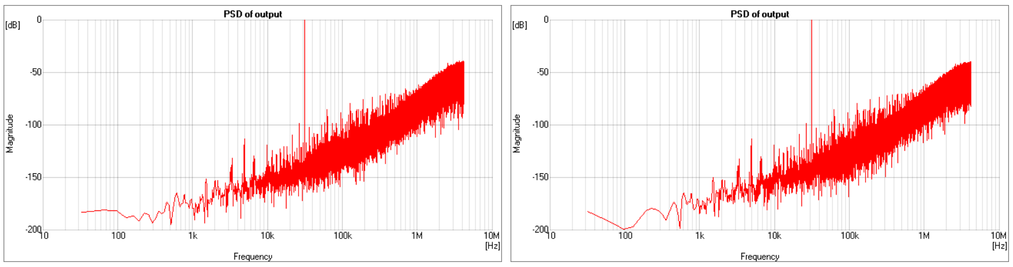


Figure 6.12: Output frequency spectrum for input $=0.5+0.2sin(2\pi f)$ and 60dB gain in all three integrators (left), and 42 dB in first integrator - infinite in the two others (right)

As one can see from table 6.1, it is the gain of the first integrator that is the most crucial to the performance of the modulator. This is because the error in the first integrator adds directly to the signal and propagate through two other integrators.

For the OLSD ADC with $42dB$ gain in all three integrators, there was only $1.74dB$ margin to meet the requirement of $70dB$ gain within a $200kHz$ band. The gain in the first integrator was therefore increased to $60dB$, since the first integrator is the most important when considering gain. The output frequency spectrum in this case is shown in figure 6.13. The obtained SNR was $84.52dB$, a SNDR of $82.53dB$ resulting in an ENOB of $13.42bits$.

From the figures showing output frequency spectrum for the OLSD ADC, it is evident that there has been added distortion compared to the ideal ADC. There are some spikes on the even harmonic frequencies of the input signal, and a lot of spikes that not correlate directly to the input. To find the source of the distortion, the frequency spectrum for the internal reset signal in the integrators were calculated. In figure 6.14 this spectrum is compared to the output of the

| Integrator gain ⇒ | IDEAL | 42 – inf – inf | 42 – 42 – inf | 42 – 42 – 42 |
|---:|---:|---:|---:|---:|
| SNR | $88.51dB$ | $71.79dB$ | $71.77dB$ | $71.74dB$ |
| SNDR | $84.74dB$ | $71.28dB$ | $71.32dB$ | $71.24dB$ |
| ENOB | $13.78bits$ | $11.55bits$ | $11.55bits$ | $11.54bits$ |

Table 6.1: Table of SNR, SNDR and ENOB versus integrator gains of $42dB$

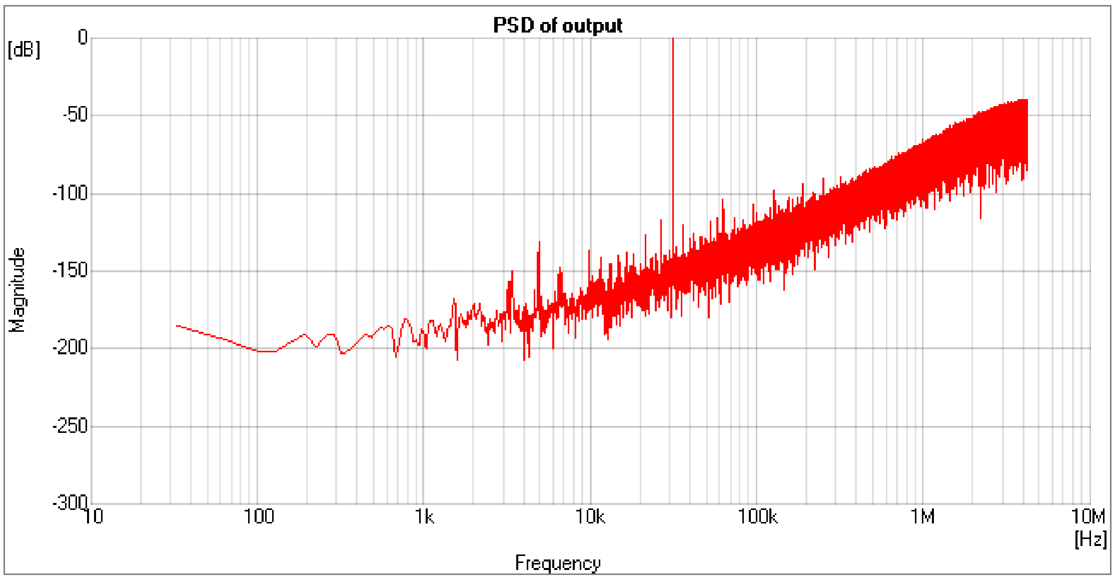


Figure 6.13: Output frequency spectrum for input $=0.5+0.2sin(2\pi f)$ and 60dB gain in the first integrator, 42dB in the two others

OLSD ADC with $60dB$ gain in the first integrator. The figure shows a close relation between the spectrum of the reset signal and the distortion in the output of the ADC. The SNR of the reset signal was in fact $29.81dB$, the SNDR was $29.41dB$, resulting in an ENOB of $4.59bits$.

Note that the output frequency spectrum for the comparator resembles the output of a conventional first order $\Sigma\Delta$ ADC.

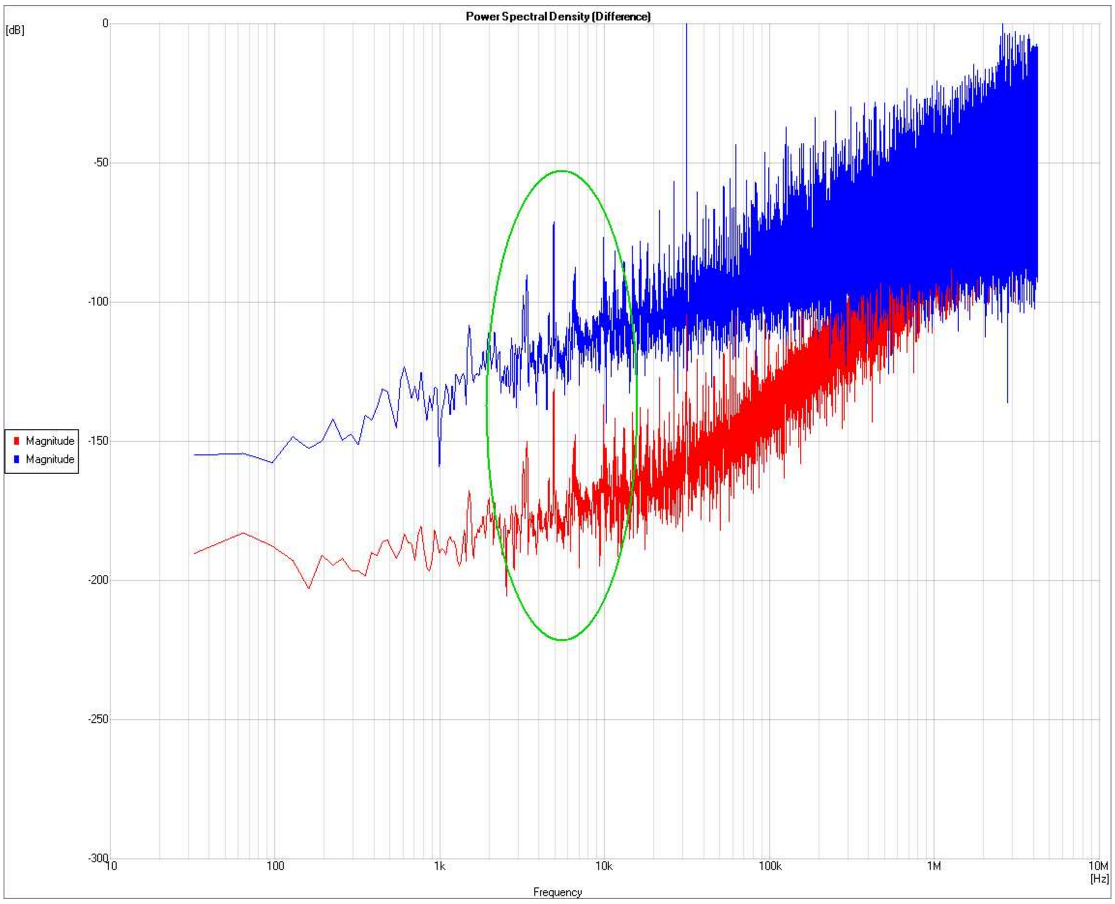


Figure 6.14: The output frequency spectrum for a third order modulator with $60dB$ gain in the first integrator (red) and the frequency spectrum for the reset signal in the first integrator (blue)

### 6.3.2 Capacitor mismatch and parasitics in integrators

The capacitor mismatch and parasitics will as described in chapter 3.2.1 result in the same changes in the behavioral model. During a reset, a voltage different from $V_{ref}$ will be subtracted from the accumulated value. Since the combination of mismatch and parasitics result in the same fault, the combined fault of the two non-idealities was simulated.

The first simulation used the worst case of a 5% parasitic capacitance at the positive side of the reset capacitor. This would result in a voltage error of $2^N k = (16 * 0.05) = 0.8V_{LSB}$.The output frequency spectrum for the modulator is shown in figure 6.15. The input voltage is the same as during the gain testing, $0.5 + 0.2sin(2\pi f)$. It is clear that the noise shaping is gone. The reason to this is the magnitude of the output voltage error after a reset. When this error is $0.8V_{LSB}$, the quantized value of the output voltage will very often be one less than it should have been. (Assuming that the quantizer is adjusted such that the quantization error is between $-\frac{V_{LSB}}{2}$ and $+\frac{V_{LSB}}{2}$. When a sample is one less than it should be, there is a probability that the difference between that sample and the previous one is falsely negative. If that is the case, the modulo differentiator will wrap around and the result would be greater than the actual input value.

The OLSD ADC was simulated with a 1% parasitic on the reset capacitor. The result can be seen in figure 6.16. The difference in the output spectrum is hardly noticeable compared to the ideal modulator. The obtained SNR was $88.22dB$, the SNDR $84.52dB$, resulting in an ENOB of $13.76bits$. In this case the integrator output error after a reset is only $0.16V_{LSB}$. The probability of a false negative is therefore greatly reduced.

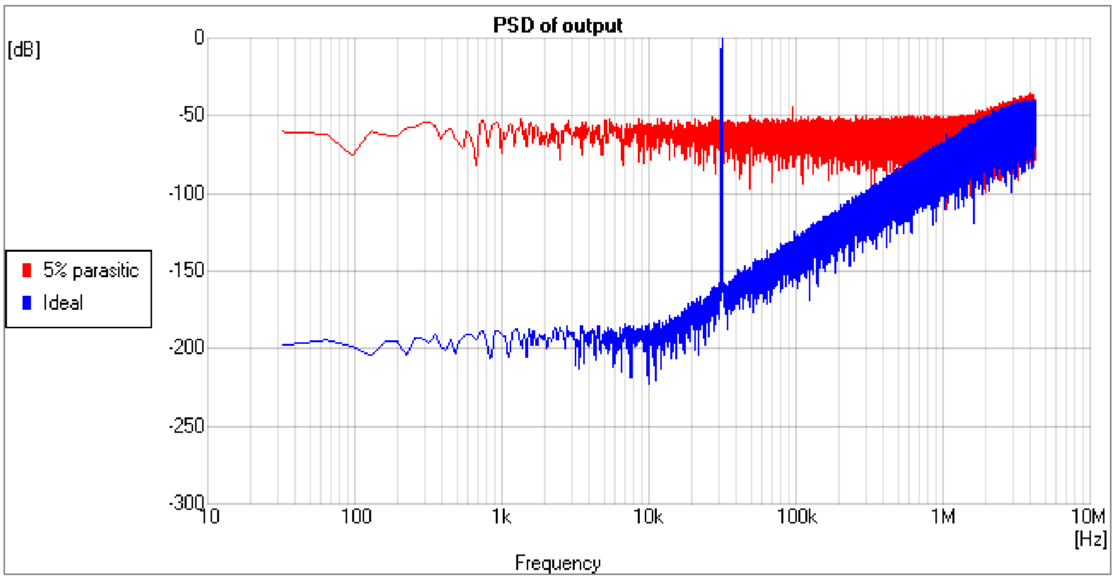


Figure 6.15: The output frequency spectrum for the ideal modulator (blue) and for the modulator with a 5% parasitic capacitance (red)

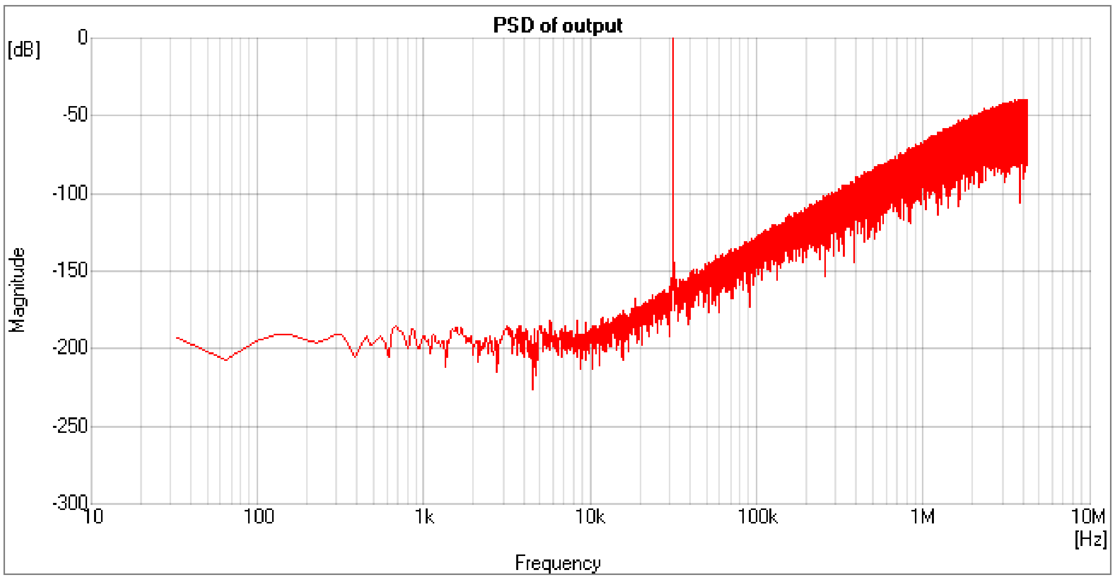


Figure 6.16: The output frequency spectrum for the modulator with a 1% parasitic capacitance (red)

### 6.3.3 Comparator offset in integrators

A $5mV$ offset voltage (relative to $V_{ref} = 1V$),was added to the comparator inn all three integrators, and no performance degradation was visible. The offset voltage was slowly increased until degradation appeared. As long as the comparator offset was less than $|0.35V_{LSB}|$, there was no noticeable performance drop at all. When the error exceeded $|0.35V_{LSB}|$, a few glitches appeared, but the difference in SNR was insignificant. When the offset voltage was in the order of $|0.45V_{LSB}|$ and larger, the noise shaping disappeared. When there is an offset in the comparator, the output voltage after a reset would be erroneous because the reset happens at the wrong level. If the error is large enough, the quantizer will output a different value than for the ideal case. If this leads to a false negative, the output value will be much larger than expected. A higher quantizer output will not be that critical, but still there will be an error.

The comparator offset will add to the error from the capacitance imperfections. It is therefore important that the combined fault of the capacitors and the comparator is small enough to not cause errors at the output of the quantizer ($V_{err} < |0.45V_{LSB}|$).

## 6.4 Specifications for use in a GSM system - summary

The simulations with circuit imperfections showed that the capacitor matching, parasitics and comparator offset are quite critical for the operation of the modulator. To see wether or not the modulator could be used in a GSM system, the capacitor errors were set to maximum $0.4V_{LSB}$ ($2.5\% of\ V_{ref}$ with a four bit quantizer). This includes mismatch and parasitics, and should be achievable in a common CMOS process when putting some effort in layout. The gain in the integrators were set to $60dB$, $42dB$ and $42dB$. The results for different matching and comparator offset can be seen in table 6.2. The results show that when both capacitor imperfections and comparator offset is present, a positive offset is less harmful than a negative offset. This is because the capacitor effects will lower the integrator output, and with a positive offset the lowered value will be canceled out by the amount of offset. When the offset is negative, it adds to the error and makes the result even worse.

It is possible to achieve a SNR of $70dB$ within the wanted $200kHz$ band, but it is necessary to be careful during layout to minimize the parasitic capacitance and the mismatch. Table 6.2 shows that the requirement is met for a capacitance error causing a total of 5% $(0.8\,V_{LSB})$ error on the output of the integrators and a comparator offset of $+50mV$ in all integrator comparators.

| Units of $V_{LSB}$ error | comp. offset | SNR | ENOB |
|---:|---:|---:|---:|
| 0.4 | $+50mV$ | $84.61dB$ | 13.45bits |
| 0.4 | $-50mV$ | $16.96dB$ | 2.51bits |
| 0.8 | $+50mV$ | $83.91dB$ | 13.38bits |
| 0.8 | $-50mV$ | $13.12dB$ | 1.88bits |

Table 6.2: SNR and ENOB for different parameters in the third order modulator

# Chapter 7

# Future work

The OLSD ADC should be implemented in CMOS to see how it performs, and how accurate it can be made when considering the sensitivity to capacitance imperfections and comparator offset in the integrators. The problem of $n + 1$ bits should also be further investigated. The answer is given for a DAC in [7], but for a ADC it may be necessary to use statistics to model this problem. A more accurate circuit with discrete components should also be made. The matching of the capacitors can be done much more accurate than in this thesis.

Chapter 8

# Conclusion

This thesis was divided into two parts, design of a practical first order open loop $\Sigma\Delta$ modulator using discrete components, and simulation of a third order OLSD ADC to investigate the consequences of circuit imperfections - and determining circuit requirements if the ADC should be used in a GSM system.

The practical modulator was designed as a first order OLSD ADC, with standard discrete components such as operational amplifiers and switches, and a microcontroller with a built in ADC. The practical circuit used surface mount capacitors with a tolerance of 20%, resulting in poor matching and inaccurate behavior of the modulo integrator. Despite the poor matching, the OLSD ADC showed a distinct noise shaping, with a slope of about $20dB$ per decade. The quantization noise was not the dominating noise source in the circuit, and the quantizer resolution had to be set to four bits of less to achieve any improvement in performance over the standard ADC.

The third order modulator was modeled and simulated at a behavior level using VHDL-AMS. The ideal circuit confirmed the results from the preliminary project [12], where the quantizer resolution had to be equal to or larger than the modulator order to obtain proper noise shaping. The simulations showed that the ideal third order modulator with a four bit quantizer could achieve a SNR of $88.51dB$, and an ENOB of $13.78bits$[1].

The third order modulator was simulated with circuit imperfections to determine the effect of these when there was no feedback present. Introducing finite gain in the integrators resulted in harmonic distortion at the output. This harmonic distortion was a result of leakage of the internal reset signal in the integrators. By setting the gain in all three integrators to $2OSR = 42dB$, the SNR of the third order modulator sunk to $71.74dB$. The gain in the first integrator was increased to $60dB$, and the SNR raised to $84.52dB$. The first integrator was the most crucial to the performance of the modulator, as is the case for conventional $\Sigma\Delta$ ADCs.

The circuit was also simulated with capacitance mismatch and comparator offset in the modulo integrator. These two imperfections resulted in the same error - the output voltage from the integrator differed from the ideal case. The total voltage error should be significantly less than $0.5V_{LSB}$ to obtain the noise shaping. If the integrator output error was too large, the noise shaping would totally disappear.

In general, it has been proved that the OLSD modulator with modulo integrators works as intended, the quantization noise is shaped like in conventional $\Sigma\Delta$ modulators. The modulator is very sensitive to capacitor mismatch and parasitics. The effect of these capacitor imperfections will increase as the quantizer resolution increase, because the error will cover more units of $V_{LSB}$.

[1]Calculated from the SNDR which was $84.74dB$

It is important to minimize these capacitor effects, as increased quantizer resolution will allow a greater input signal swing. The comparator offset in the integrators will add to the errors from the capacitors. A positive offset will cancel some of the capacitance error, while a negative offset will make it even worse.

The requirements for use in a GSM system were met with integrator gains of $60dB - 42dB - 42dB$, a total capacitance error causing an integrator output error of $+0.8V_{LSB}$ and a comparator offset of $+50mV$ in all three integrators. The obtained SNR was $84.61dB$. This should be possible to implement in a standard CMOS process.

# Bibliography

[1] http://www.mathworks.com/products/matlab/.

[2] http://www.mentor.com/products/ic_nanometer_design/simulation/advance_ms/.

[3] Analog modulo integrator for use in open-loop sigma-delta modulators. http://folk.ntnu.no/wulff/olsd.pdf.

[4] Teaching supervisor carsten wulff.

[5] Thomas Burger and Qiuting Huang. A 13.5-mw 185-msample/s $\delta\sigma$ modulator for umts/gsm dual-standard if reception. *IEEE Journal of Solid-State Circuits*, 36(12), 2001.

[6] Atmel Corporation. Atmega48 datasheet.

[7] Mats E. Høvin Dag T. Wisland and Tor S. Lande. A second-order non-feedback delta-sigma modulator for d/a conversion. *IEEE*, 2002.

[8] Ken Martin David A. Johns. *Analog Integrated Circuit Design*. John Wiley and sons inc., 1997.

[9] Tor Sverre Lande Chris Toumazou Mats Høvin, Alf Olsen. Delta-sigma modulators using frequency-modulated intermediate values. 1997.

[10] Ana Rusu and Hannu Tenhunen. A third-order sigma-delta modulator for dual-mode receivers.

[11] Carsten Wulff. A brief introduction to systemdotnet, http://www.sf.net/projects/systemdotnet. 2005.

[12] Øystein Knauserud. Simulation of an open loop sigma-delta analog-to-digital converter. 12 2005.

# Appendix A

# ADC theory

This chapter presents the most basic theory about quantization of analog signals and quantization with noise shaping. Theory about the open-loop modulator is also presented. The open-loop modulator is shown to be identical in operation to the conventional modulator.

## A.1 Quantization of analog signals

When performing quantization, one converts an analog signal with an infinite amount of possible amplitudes, to a digital value. This digital value has an finite number of possible values determined by the resolution of the quantizer. The number of values is given by Equation A.1.

$$n = 2^N \tag{A.1}$$

where N denotes the number of bits in the quantizer. A 1-bit quantizer will for example have two possible values, 0 and 1. For a 8-bit quantizer the number of possible values is 256.

The quantizer also needs a reference voltage to convert the signal. This could be either a single voltage or two voltages, dependent of the type of quantizer. If a single voltage is used this often specifyes the maximum input voltage. The required input voltage to increase the output by one LSB (Least Significant Bit) is given by equation A.2.

$$V_{LSB} = \frac{V_{ref}}{2^N} \tag{A.2}$$

### A.1.1 Quantization error

Equation A.2 gives the smallest change in input voltage that triggers a change on the digital output. Then there must be an interval where the analog input can change while the digital output remains the same. The input voltage range where the output will not change is given by Equation A.3

$$-\frac{1}{2}V_{LSB} < V_x < \frac{1}{2}V_{LSB} \tag{A.3}$$

where $V_x$ is in the range where the output will not change. The quantization error will then be

$$V_e = \pm\frac{V_{LSB}}{2} \tag{A.4}$$

As one can see from the equation above the quantization error is reduced when the number of bits in the quantizer is increased. An example of the output from a 3-bit quantizer is shown in Figure A.1. The quantization error for the same converter is also shown. If the input voltage exceeds the limits of the quantizer the quantizer will be *overloaded* and the quantization error will be larger than the value given in Equation A.4.

### A.1.2 White noise assumption

The quantization error can under certain circumstances be treated as additive white noise. To treat the quantization error as additive white noise one has to do the assumption that the quantization error is a random value independent of the input signal, and uniformly distributed between $\frac{-V_{LSB}}{2}$ and $\frac{V_{LSB}}{2}$ (p 450 in [8]). This can be assumed if the input signal is *busy* or rapidly changing in common words.

When doing this, one can treat the quantization error as additive noise and make a linear model of the quantizer. This does the computing of transfer functions easier. A linear model of a quantizer is shown in Figure A.2.

The white noise assumption will be used in all calculations in this report to simplify the calculations.

### A.1.3 Quantization noise

The quantization error described in A.1.1 will appear as noise on the output of the quantizer. The power of this noise is an important factor in the calculation of the Signal-to-Noise Ratio (SNR)of the quantizer. The SNR value gives the Signal-to-Noise ratio within the band of interest. It should be mentioned that this value excludes any harmonics of the quantization noise, if present. This can give a wrong impression of the performance, and to get a better impression one should use Signal-to-Noise and Distortion Ratio (SNDR). This includes the harmonics and gives a better view of the performance.

The average value of the quantization noise can be shown to be zero. This is shown in Equation A.5 (equation 11.13 in [8]). Even though the average value of the quantization error is zero, the RMS value is not insignificant. The rms value differs from zero and will add to the signal. The RMS value of the quantization noise is given in Equation A.6(equation 11.14 in [8]). The power spectral density (PSD) of an analog signal with frequency of 1kHz is shown in Figure

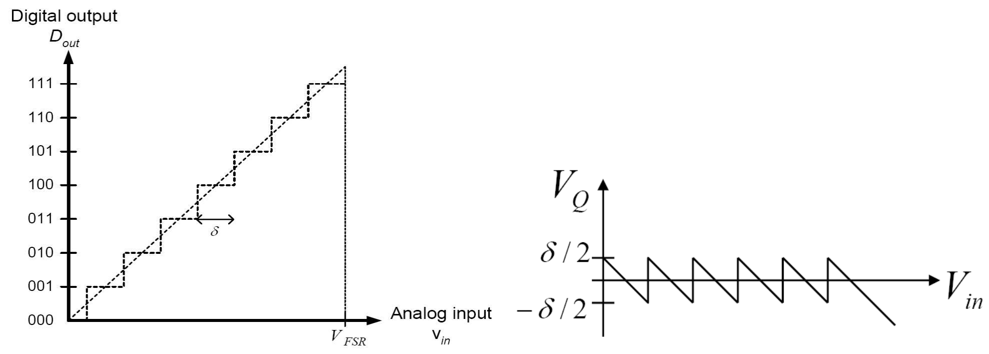


Figure A.1: (Left): Analog input signal and digital output word. (right): Quantization error

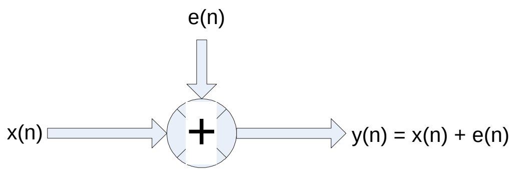


Figure A.2: Linear model of a quantizer

A.3. The PSD of the same signal quantized with a 1-bit quantizer is also shown. It is clear that the noise floor is raised due to the quantization error. Harmonics of the quantization noise is also present.

One way to dampen the harmonics of the quantization noise is by using dithering (pp 563-564 in [8]. Dithering can be done by adding a random to the signal right before the quantizer. In this way the dither signal is shaped in the same manner as the quantization noise. Since the dither signal behaves randomly the harmonics will be reduced because the quantization noise becomes less correlated with the input signal. The result is reduced harmonics, but the dither signal acts like noise and adds about $3dB$ to noise in the signal band.

$$V_{Q(avg)} = \int_{-\infty}^{\infty} x f_Q(x) dx = \frac{1}{V_{LSB}} \left( \int_{-\frac{V_{LSB}}{2}}^{\frac{V_{LSB}}{2}} x dx \right) = 0 \tag{A.5}$$

$$V_{Q(rms)} = \left[ \int_{-\infty}^{\infty} x^2 f_e(x) dx \right]^{\frac{1}{2}} = \left[ \frac{1}{V_{LSB}} \left( \int_{-\frac{V_{LSB}}{2}}^{\frac{V_{LSB}}{2}} x^2 dx \right) \right] = \frac{V_{LSB}}{\sqrt{12}} \tag{A.6}$$

If one assumes that the input to the quantizer is a sawtooth signal with maximum amplitude of $V_{ref}$, and disregard any DC-level the SNR is given in Equation A.7 (equation 11.15 in [8]). Here one see that if one increase the resolution of the quantizer by one bit the SNR is increased by approximately 6dB. If the input signal is a sinusoidal waveform between 0 and $V_{ref}$ it is shown that the SNR is $6.02N + 1.76dB$ (equation 11.16 in [8]).

$$SNR = 20log\left(\frac{V_{in(rms)}}{V_{Q(rms)}}\right) = 20log\left(\frac{\frac{V_{ref}}{\sqrt{12}}}{\frac{V_{LSB}}{\sqrt{12}}}\right) = 20log\left(2^N\right) = 6.02NdB \tag{A.7}$$

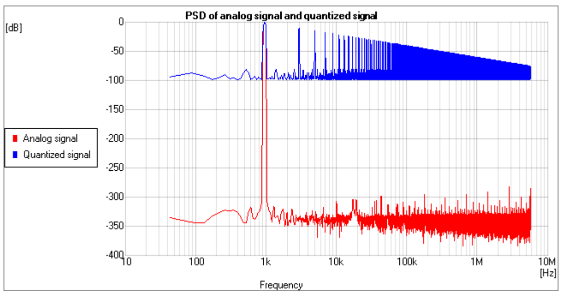


Figure A.3: PSD of analog signal (red) and 1-bit quantized signal (blue)

## A.2 Oversampling Analog-to-Digital converters

In oversampling ADCs the analog circuitry meets more relaxed requirements. This leads to a more complex digital circuitry to compensate for the losses in the analog part. This is getting more important when the technology for realizing integrated circuits evolve. The trend is that the supply voltages decrease when a new step in the technology is released. The threshold voltage does not decrease as much as the supply voltage, and the analog circuitry is getting harder to design.

When oversampling one samples the input signal at a higher rate than the Nyquist-rate. The Nyquist-rate is given by $f_s = 2f_0$ where $f_0$ is the signal bandwidth. Then one can define the OverSampling Ratio (OSR) as given in Equation A.8.

$$OSR \equiv \frac{f_s}{2f_0} \tag{A.8}$$

It is then shown on pp 535-536 in [8] that the maximum SNR for an oversampled ADC is as given in Equation A.9.

$$SNR_{max} = 10log\left(\frac{P_s}{P_e}\right) = 10log\left(\frac{3}{2}2^{2N}\right) + 10log(OSR) \tag{A.9}$$

The first part of this equation is the same as in the calculation of SNR without oversampling, whereas the last part is the gain from the oversampling. The oversampling adds 3dB to the SNR for each doubling of the sampling frequency, or equivalently 0.5 bits per octave.

## A.3 Oversampling with noise shaping

It is clear from Equation A.9 that using only oversampling is not enough if one wants a high SNR without sampling at really high frequencies. To obtain a higher SNR, the noise in the pass band has to be suppressed. To do this one can use a $\Sigma\Delta$-modulator. A general $\Sigma\Delta$-modulator and its linear model is shown in Figure A.4. In the linear model the quantization noise is assumed to be white.

To derive the signal transfer function ($S_{TF}$) and the noise transfer function ($N_{TF}$) one can use superposition since the quantization noise is assumed to be independent of the input signal. The two transfer functions is given in Equation A.10 and A.11 (equations 14.15 and 14.16 in [8]).

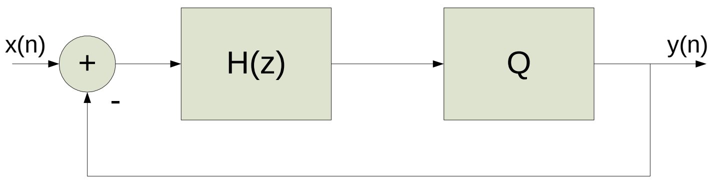
$$S_{TF}(z) \equiv \frac{Y(z)}{U(z)} = \frac{H(z)}{1+H(z)} \tag{A.10}$$

$$N_{TF}(z) \equiv \frac{Y(z)}{E(z)} = \frac{1}{1+H(z)} \tag{A.11}$$

The total output signal Y(z) will then be as in Equation A.12 (Equation 14.17 in [8]).

$$Y(z) = S_{TF}(z)U(z) + N_{TF}(z)E(z) \tag{A.12}$$

It is clear from the equations above that the magnitude of the loop filter should be large for frequencies inside the pass band $(0 - f_0)$. With such a loop filter the $S_{TF}$ will be approximately one in the pass band, while the $N_{TF}$ will reach zero in the pass band. This will suppress the noise and the SNR will increase. If the order of the loop filter increases the noise will be more suppressed in the signal band, and even greater SNR will be achieved.



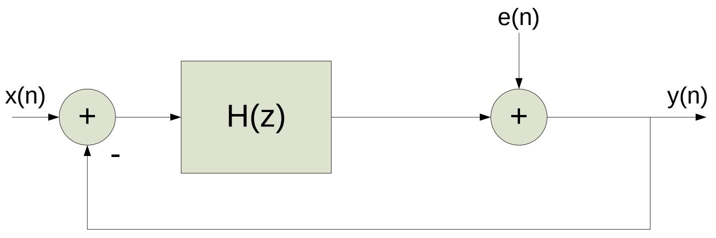


Figure A.4: A $\Delta\Sigma$ modulator and the linear model of the modulator

The quantizer in a $\Sigma\Delta$-modulator can in practice be almost any Nyquist-rate converter, but it is preferred to use a converter that is capable of running at high speeds because of the high sampling frequencies. It is very common to use a 1-bit quantizer. The advantage of using a 1-bit quantizer appears when looking at a real model of a $\Delta\Sigma$-modulator. In a real modulator there has to be a Digital-to-Analog Converter (DAC) in the feedback loop. This DAC has to have the same resolution as the quantizer to obtain full resolution of the ADC. The 1-bit DAC is said to be *inherently linear* since it has only two output values, and the shortest path between those values is a straight line. In this way one can obtain a good linearity, which is important since the $\Delta\Sigma$-modulator does not improve linearity.

### A.3.1 First order noise shaping

In a first order $\Sigma\Delta$-modulator the $H(z)$ can be a discrete-time integrator. This has a zero at DC-level so the signal will be un-shaped. The poles of the integrator is equal to the zeros of the $N_{TF}$. The transfer function of this integrator is shown in Equation A.13

$$H(z) = \frac{1}{z-1} \tag{A.13}$$

It is shown in pp 540-541 in [8] that the $S_{TF}$ and $N_{TF}$ of the modulator will be as in Equation A.14 and A.15.

$$S_{TF}(z) = \frac{\frac{1}{z-1}}{1+\frac{1}{z-1}} = z^{-1} \tag{A.14}$$

$$N_{TF}(z) = \frac{1}{\frac{1}{1+\frac{1}{z-1}}} = (1-z^{-1}) \tag{A.15}$$

It is clear that the signal is delayed by one sample while the noise is high-pass filtered. This dampens the noise in the signal-band and give a higher SNR. In page 542 and 544 in [8] it is shown that the maximum SNR of a first order ΣΔ-modulator is as i Equation A.16, and for a second order modulator in Equation A.17.

$$SNR_{max} = 6.02N + 1.76 - 5.17 + 30log(OSR) \tag{A.16}$$

for a sinusoidal input signal. Doubling the sampling frequency adds 9dB to the SNR, or 1.5 bits equivalently.

$$SNR_{max} = 6.02N + 1.76 - 12.9 + 50log(OSR) \tag{A.17}$$

for a sinusoidal input signal. Doubling the sampling frequency adds 15dB to the SNR, or 2.5 bits equivalently.

## A.4 Noise shaping in the open-loop modulator

By having a modulo integrator followed by a quantizer and a modulo differentiator, there is no need for feedback to achieve noise shaping. The modulo integrator accumulates values until the output voltage exceeds the maximum voltage, and then subtracts this maxium voltage to keep the remainder. To prove this, one has to look at the basic of integration/accumulation and differentiation. The following two cases explains why the OLSD should function like a conventional ΣΔ modulator.

**Case1: No reset has occurred**

The integrator has accumulated some values over the last time, $t$, but the output voltage is still below the reference, hence no reset has occurred. Without the reset, no modulo operation that can complicate the transfer function has been executed.

It is a mathematical proof that the derivative of an integral of a function is the function itself, as in Equation A.18.

$$\frac{d}{dt}\left[\int x(t)dt\right] = x(t) \tag{A.18}$$

This is also valid in discrete time:

$$\left[\left(\sum_{i=0}^{n} x_i\right) - \left(\sum_{i=0}^{n-1} x_i\right)\right] = x_n \tag{A.19}$$

Equation A.19 shows that the last input sample $x_n$ is retrieved at the output. As long as $x_n$ is bonded by 0 and $(2^N - 1)$, $x_n mod(2^N - 1)$ equals $x_n$.

**Case 2: A reset has just occurred**

When a reset has occurred in the integrator the *real* accumulated value (call it $x^{'}(i)$) has been subtracted the maximum value. This maximum value corresponds to the value $n = 2^N - 1$ from the quantizer. The output from the modulo differentiator is now as shown in Equation A.20.

$$mDiff_{out} = \Big(x(i) - x(i-1)\Big)\ mod\ n \tag{A.20}$$

Now, the difference in the equation above is negative. This is always true after a reset since the input is bounded by $max$, and $max$ is subtracted from the real accumulated value.

The result from Equation A.20 when the left hand side is negative is given in Equation A.21.

$$mDiff_{out} = (x(i) - x(i-1) + n) \;\; mod \; n \tag{A.21}$$

when using 2's complement representation of negative numbers. Now, remember that $x(i)$ actually is $x^{'}(i) - max$, (or $x^{'}(i) - n$) Combining that with Equation A.21 gives

$$mDiff_{out} = x^{'}(i) - x(i-1) \; mod \; n \tag{A.22}$$

which shows that the result after reset is the same as if the reset never occurred.

# Appendix B

# List of abbreviations

| | |
|---|---|
| **ADC** | Analog to Digital Converter |
| **DAC** | Digital to Analog Converter |
| **ENOB** | Effective Numer Of Bits |
| **LSB** | Least Significant Bit |
| **OLSD** | Open Loop Sigma Delta |
| **OSR** | OverSampling Ratio |
| **PSD** | Power Spectral Density |
| **SNR** | Signal to Noise Ratio |
| **SNDR** | Signal to Noise and Distortion Ratio |

Appendix C

# Schematic for the practical first order modulator

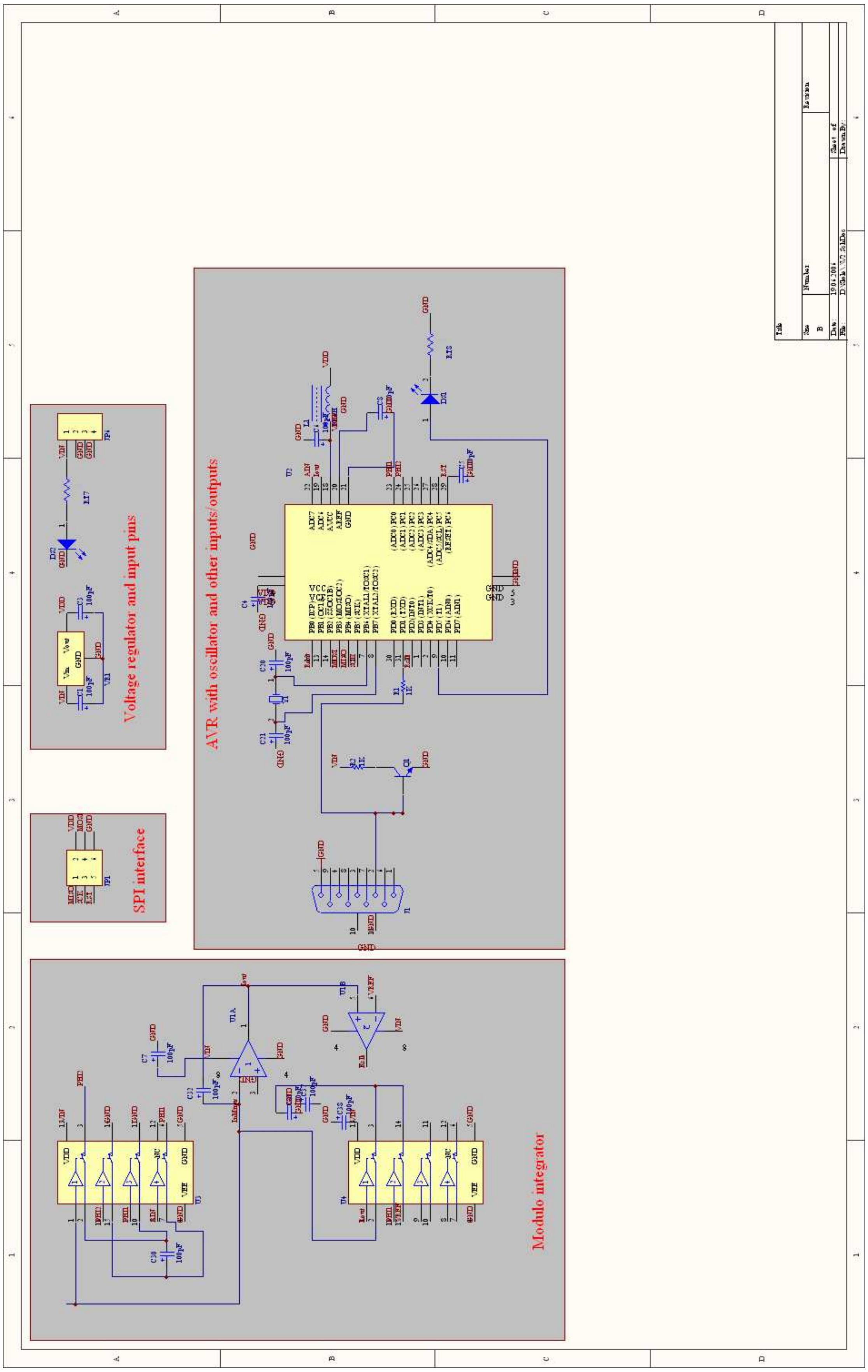

Voltage regulator and input pins
SPI interface
AVR with oscillator and other inputs/outputs
Modulo integrator

Figure C.1: Schematic for the open loop modulator

# Appendix D

# Software for practical modulator

## D.1 Software for the AVR microcontroller

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#define SAMPLE_DELAY 30
#define TESTC 20
#define VANLIG 40
#define OL     50

void send(unsigned char d);
void delay();
void delay2();

volatile unsigned char t_sample=0;
volatile unsigned char p_sample=0;
volatile unsigned char out=0;
volatile unsigned int testing=0;
unsigned char s=0;
volatile unsigned int N=0;
volatile unsigned int M=0;
unsigned int i=0;
unsigned char busy=0;

volatile char FAULT =2;
volatile char MODE = VANLIG;
volatile char RESET=0;
volatile unsigned char POWER = 14;
volatile long int samples=0;
SIGNAL(SIG_INTERRUPT0)
{
        //**********************************
        //This interrupt triggers on the
        //output from the comparator.
        //Set PORTB high to reset integrator
        //**********************************
        PORTB = 0xff;
        delay();
        delay();
        PORTB = 0x00;


}
SIGNAL(SIG_OVERFLOW0)
{
        //**********************************
        //This interrupt triggers on the
        //timer0 overflow. It controls the
        //two clocks and the quantizer
        //**********************************

        PORTB = 0x00; //output for clocks
        TCNT0 = 0; //reset counter
        s = s+1;



        //in PHI2, invert the clocks without overlapping
        if(s==1)//PHI2
        {


                PORTC &= ~(1<<0);
                delay();
                PORTC |=(1<<1);
```

```
                    delay();


          }

          //in PHI1, invert the clocks and start quantization
          else if(s==2)//PHI1
          {



                    PORTC &= ~(1<<1);
                    delay();
                    //delay();

                    PORTC |=(1<<0);
                    delay();
                    ADCSRA |= (1<<ADSC);
                    s=0;
          }


}

SIGNAL(SIG_ADC)
{


          busy =0x01;
          //store the sample from the adc
          //and right shift four times
          t_sample = (ADCH>>4);

          //send the sample via RS232
          send(t_sample);

          N++;




}
void init()
{
          //clock output
          DDRD = 0xff;
          DDRD &= ~(1<<PD2);
          PORTD = 0x00;
          //clock generation
          TCCR0B |= (4<<CS00);
          TIMSK0 |= (1<<TOIE0);

          //AD
          ADCSRA |= (1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(0<<ADPS0)|(1<<ADEN);
          //SFIOR |= (1<<ADTS0)|(1<<ADTS1);
          ADMUX |= (1<<MUX0)|(1<<MUX2)|(1<<MUX1)|(1<<REFS0)|(1<<REFS1)|(1<<ADLAR);
          //UART
          UCSR0A |= (1<<U2X0);
          UCSR0B |= (1<<TXEN0);
          UCSR0C |= (1<<UCSZ00)|(1<<UCSZ01);

          UBRR0 = 0x10; //0x22: 56700, 0x10: 115200

          //external interrupt
          EICRA |= (1<<ISC00)|(1<<ISC01);
          EIMSK |= (1<<INT0);
          DDRC = 0xff;
          DDRB = 0xff;
          PORTB = 0xff;
          delay2();
          delay2();
          delay2();
          delay2();

          PORTB = 0x00;
          MODE = VANLIG;
          sei();
          samples = (1<<POWER);
}
void delay()
{
          int i;
          for(i=0; i<SAMPLE_DELAY; i++)
          {
                    asm("nop␣\n\t");
          }
}

void delay2()
{
          int i;
          for(i=0; i<32000; i++)
          {
                    asm("nop␣\n\t");
          }
}

void send(unsigned char data)
```

```
{
        while ( !( UCSR0A & (1<<UDRE0)) );
        UDR0 = data;
        M++;


}
int main()
{

        int a=0;

        //initialze everything
        init();

        //turn on LED
        PORTD = 0xff;
        //send POWER, number of samples = 2^POWER
        send(POWER);
        //send number of quantizer levels
        send(15);



        //run for N samples with normal quantization
        while(N<samples)
        {

        }
        //Switch to open loop sigma-delta
        //and run N new samples
        ADMUX &= ~(1<<MUX0);
        N=0;
        MODE = OL;
        while(N<samples);
        //Finished, clear interrupts and turn off LED
        N=0;
        cli();


        PORTD = 0x00;

        while(1);
        return 0;
}
```

## D.2 Matlab script for reading out data an performing modulo operations

```
%read out data from the OLSD
clear

%initialize serial port
port = serial('COM1');
port.BaudRate = 115200;
port.InputBuffer = 800000;
port.Timeout = 10000;
fopen(port);

%read first two bytes (2^power number of samples)
power = fread(port,2)

%read next byte (number of quantizer levesl)
n = fread(port,1)

%read 2^power samples times two
data = fread(port,(2.^power)); %regular quantization
data2 = fread(port,(2.^power)); %OPEN LOOP
fclose(port);

%open the output files
fil = fopen('diffout.csv','wt');
fil3 = fopen('kvantisert.csv','wt');

%run modulo operation on data2 and write to files
for i=2:length(data2)

    d=mod(data2(i)-data2(i-1),n)
    ddata(i)=d;

    fprintf(fil,'%i\n',d);
    fprintf(fil3,'%i\n',data(i));
end

%close files
 fclose(fil);
 fclose(fil3);
```

# Appendix E

# VHDL source for the third order modulator

## E.1 The modulo integrator

```
-------------------------------------------------------------------------------
-- Author:          Øystein Knauserud <>
-- Created at:      Wed Jan 11 13:20:54 2006
-- Modified at:     Tue Jun 13 18:45:13 2006
-- Modified by:     Øystein Knauserud <knauseru@stud.ntnu.no>
-------------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
library IEEE_proposed;
use IEEE_proposed.electrical_systems.all;


entity mint is

  generic (
    vmax: real  := 1.0;
    damp : real:=1.0;
    cpm: real:=1.0;
    gainerror: real:=1.0);


  port (
    terminal input  : electrical;
    signal clk : in std_logic;
    signal rout: out integer:=0;
    --terminal ref: electrical;
    --signal r:out  real;
    terminal  output: electrical);

end entity mint;

-------------------------------------------------------------------------------
architecture ideal of mint is
  quantity v_in across input;
  quantity v_out across i_out through output;
  signal temp: real := 0.0;
begin  -- architecture ideal

  run: process(clk) is
         variable temp2: real :=0.0;

       begin
         if(clk'event and clk='1') then
           rout <= 0; --reset signal
            if((temp+(v_in)) > (vmax)) then --comparator trigger
              temp <= (temp + (v_in)) - (vmax*cpm); --output remainder
              rout <= 1; --reset signal

            else
              temp <= (temp + (v_in)); --no reset, output unchanged
              rout <= 0;

            end if;
          else
         end if;

       end process run;


    v_out == temp*gainerror;











 end architecture ideal;
--------------------------------------------------------------------------------
```

## E.2 The flash quantizer

```
-------------------------------------------------------------------------------
-- Author:         Øystein Knauserud <>
-- Created at:     Wed Jan 11 13:20:54 2006
-- Modified at:    Tue Jun 13 22:37:13 2006
-- Modified by:    Øystein Knauserud <knauseru@stud.ntnu.no>
-------------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library IEEE_proposed;
use IEEE_proposed.electrical_systems.all;

entity FLASH is
  generic(
    vmax: real :=1.0;
    ok: real:=0.00;
    N: integer:=4);
  port(
    terminal input : electrical;
    signal clk: in std_logic:='0';
    signal dout: out integer :=0);
end entity FLASH;


architecture behave of FLASH is
  quantity v_in across input;
  constant nN: integer := (2 ** N)-1;
  constant Rstep: real:= real(vmax/((real(nN-1)*2.0)+2.0));

  signal tempout: unsigned(nN downto 0) := (others=>'0');
  signal tempout2: unsigned(nN-1 downto 0) := (others =>'0');
begin

  run: process(clk) is
          variable hit:integer:=0;
       begin
        if(clk'event and clk='1') then

         --iterate through the comparators
         for i in 0 to nN loop
           if(v_in <= (vmax-(2.0*Rstep*real(i))+(Rstep)) then
             tempout(i) <= '1';
           else
             tempout(i) <= '0';
           end if;
         end loop;

         for ii in 0 to nN loop
           if(tempout(ii)='0') then
             hit := hit+1;
           end if;

           dout <= hit-1;

           if(ii=(nN)) then
             hit:=0;
           end if;
         end loop;

         for iii in 0 to nN-1 loop




         end loop;
        end if;
       end process run;

end architecture behave;
```

## E.3 The modulo differentiator

```

-------------------------------------------------------------------------------
-- Author:         Øystein Knauserud <>
-- Created at:     Wed Jan 11 13:20:54 2006
-- Modified at:    Tue Jun 13 22:41:26 2006
-- Modified by:    Øystein Knauserud <knauseru@stud.ntnu.no>
-------------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library IEEE_proposed;
use IEEE_proposed.electrical_systems.all;


entity DIFF is

  generic (
    N : integer :=4;
    mult : integer:=1);


  port (
    signal clk     : in  std_logic:='0';
    signal diffout : out integer :=0;--unsigned(N-1 downto 0) := (others =>'0');
    signal diffin  : in  integer:=0);--unsigned(N-1 downto 0) := (others => '0'));

end entity DIFF;

-------------------------------------------------------------------------------
architecture ideal of DIFF is


begin  -- architecture ideal

  run: process(clk) is
                    variable current, prev : unsigned(N downto 0) := (others => '0');
                    variable diff          : unsigned(N downto 0) := (others => '0');
                    variable check: unsigned(N-1 downto 0) := (others => '1');


  begin
    if(clk'event and clk = '1') then
      --hold previous value
      prev := current;
      current := (to_unsigned(diffin,N+1));
       --subtract
      diff   := (current +((not prev)+1));

      if(diff(N)='1' ) then
       diffout <= to_integer(diff(N-1 downto 0)-1);
      else
       diffout <= to_integer(diff(N downto 0));

      end if;
    else

    end if;
  end process run;
end architecture ideal;
-------------------------------------------------------------------------------
```

## E.4 The complete third order modulator

```
-------------------------------------------------------------------------------
-- Author:         Øystein Knauserud
-- Created at:     Wed Jan 11 13:42:25 2006
-- Modified at:    Tue Jun 13 18:13:34 2006
-- Modified by:    Øystein Knauserud <knauseru@stud.ntnu.no>
-------------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library IEEE;
use IEEE.MATH_REAL.all;
library IEEE_proposed;
use IEEE_proposed.electrical_systems.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
library MGC_AMS;
use MGC_AMS.ELDO.all;
use MGC_AMS.eldo_parameters.all;

entity olsd is
  generic(
    cpm,ge1,ge2,ge3: real:=1.0;
    nbit: real:=4.0);
 port(
   terminal v_in: electrical;
   signal clk : in std_logic:='0';
   signal testdin : in integer:=0;
   signal output : out integer:=0);
end entity olsd;


```

```
architecture behave of olsd is



  constant damp : real := 1.0;           -- dempeledd mellom integratorer
  constant mult: integer := 1;



  constant N : integer := integer(nbit);

  terminal int1_out: electrical;

  terminal int2_out: electrical;

  terminal int3_out:electrical;


  signal qout : integer:=0

  signal diff1_out :  integer:=0

  signal diff2_out :  integer:=0;
  signal diff3_out,dummy,dummy2 :  integer:=0;

  signal ir1,ir2,ir3: integer:=0;



   begin  -- architecture testbench


  integrator1: entity work.mint(ideal)
    generic map(
      vmax => 1.0,
      cpm => cpm,
      gainerror => ge1,
      damp => 1.0)
    port map(
      input => v_in,
      output => int1_out,
      rout => ir1,
      clk => clk);

  integrator2: entity work.mint(ideal)
   generic map(
     vmax => 1.0,
     cpm => cpm,
     gainerror => ge2,
     damp => damp)
   port map(
     input => int1_out,
     output => int2_out,
     rout => ir2,
     clk => clk);

  integrator3: entity work.mint(ideal)
   generic map(
     vmax => 1.0,
     cpm => cpm,
     gainerror => ge3,
     damp => damp)
   port map(
     input => int2_out,
     output => int3_out,
     rout => ir3,
     clk => clk);

  quantizer: entity work.FLASH(behave)
    generic map(
      vmax => 1.0,
      ok => 0.0150,
      N => N)
    port map(
      input => int3_out,
      clk => clk,
      dout => qout);

  mdiff1: entity work.diff(ideal)
    generic map(
      N => N,
      mult => 1)
    port map(
      clk => clk,
      diffin => qout,
      diffout => diff1_out);

  mdiff2: entity work.diff(ideal)
   generic map(
     N => N,
     mult => 1)
   port map(
     clk => clk,
     diffin => diff1_out,
     diffout => diff2_out);

   mdiff3: entity work.diff(ideal)
   generic map(
      N => N,
      mult => 1)
    port map(
      clk => clk,
      diffin => diff2_out,
      diffout => output);


end architecture behave;
```

## E.5 Test bench

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library IEEE;
use IEEE.MATH_REAL.all;
library IEEE_proposed;
use IEEE_proposed.electrical_systems.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
library MGC_AMS;
use MGC_AMS.ELDO.all;
use MGC_AMS.eldo_parameters.all;

entity tb_olsd is

end tb_olsd;

architecture test of tb_olsd is
 --input parameters from cmd-file
 constant N : real := param("nbit");
 constant ampl : real :=param("aa");
 constant dc : real:=param("pdc");
 constant gain1: real:=param("gain1");
 constant gain2: real:=param("gain2");
 constant gain3: real:=param("gain3");
 constant pcpm : real := param("pcpm");
 constant ge1: real:=1.0/(1.0+(1.0/(10**(gain1/20.0))));
 constant ge2: real:=1.0/(1.0+(1.0/(10**(gain2/20.0))));
 constant ge3: real:=1.0/(1.0+(1.0/(10**(gain3/20.0))));

 signal clk: std_logic :='0'; -- global clock
 signal output,output2,opdiff: integer:=0; -- output from OLSD
 terminal v_in: electrical; -- input voltage to OLSD
 terminal ground: electrical;

 quantity v_inp across  v_in;
 signal  test : voltage := 0.0;
 signal testdin : integer := 0;
   --*********************************
  --*Input signal generator
  --*********************************
   component ELDO_stimuli is
     generic(f:real;a:real;dc:real);
     port(terminal ground,output:electrical);
   end component ELDO_stimuli;
   attribute Eldo_device
     of ELDO_stimuli : component is Eldo_subckt;
   attribute Eldo_subckt_name
     of ELDO_stimuli : component is "stimuli";
   attribute Eldo_file_name
     of ELDO_stimuli : component is "stimuli.ckt";

 --**************************************
 --OLSD ADC
 --**************************************
  component olsd is
     generic(cpm,ge1,ge2,ge3:real;nbit:real);
     port(terminal v_in:electrical;
          signal clk: std_logic;
          signal testdin : in integer;
          signal output: integer);
  end component olsd;

 --***************************
 --Log file
 --***************************
 file outputfile : TEXT open WRITE_MODE is "simres/outputfile1.csv";

begin
    --generate stimuli from spice
    stim1: ELDO_stimuli
      generic map(f=>31.264e3,
                  dc => dc,
                  a => ampl )
      port map(ground => ground,
               output => v_in);
    --modulator one
    olsd1: olsd
      generic map(
        ge1=>1.0,
        ge2=>1.0,
        ge3=>1.0,
        cpm=>1.0,
        nbit=>N)
      port map(
        v_in=>v_in ,
        clk=>clk ,
        testdin=>testdin ,
        output=>output);

     --modulator 2
     olsd2: olsd
      generic map(
        ge1=>ge1 ,
        ge2=>ge2 ,
        ge3=>ge3 ,
        cpm=>pcpm ,
        nbit=>N)
      port map(
        v_in=>v_in ,
        clk=>clk ,
        testdin=>testdin ,
        output=>output2);

    clk <= not clk after 59604.64478 ps;
    opdiff <= output - output2; --calcultate diffrence between 1 and 2

   --v_inp == test'ramp(1.0e-4,0.0);


    --log process
    log: process(clk) is
            variable L: LINE;
            variable cnt : integer := 0;
        begin
          if(clk'event and clk='1') then
            write(L,opdiff);
            write(L,';');
            write(L,output);
            write(L,';');
            write(L,output2);
            write(L,';');
            write(L,v_inp);
             writeline(outputfile,L);
          end if;
        end process log;

    --log process that writes signal names - executes first
    firstlog: process is
                variable line2: LINE;
              begin
                write(line2,string'("Ideal_output;Gainerror_output;Difference;input"));
                writeline(outputfile,line2);
                wait;

              end process firstlog;
end test;
```