

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

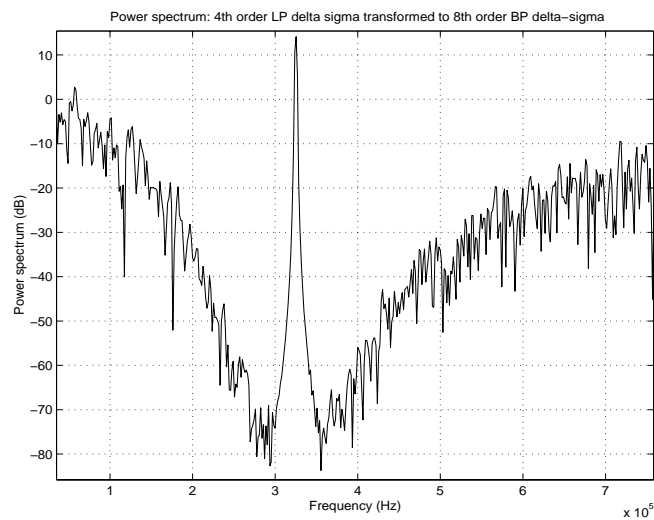
REALIZATION OF A SIGMA-DELTA MODULATOR IN FPGA

Bachelor thesis 1999 at Ericsson Microwave Systems AB

by

Michael Melin

Magnus Nilsson



Supervisor, EMW: Rune Olsson

Supervisor, CTH: Lars Bengtsson

Examiner: Bert Lanne

Institution of electronics

Chalmers university of technology

Gothenburg 1999

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

Abstract

Ericsson Microwave Systems develops radarsystems also for military applications. In these environments high radar resolution and long range are desired, thus high demands must be met by the generated and transmitted radar signal.

In this Bachelor thesis the design of sigma-delta modulators for use in onebit waveform generators are described.

A theoretical model for an eight-order sigma-delta modulator has been developed and simulated in Matlab. A hardware description has been made in VHDL for realization in FPGA. Simulations of the VHDL-code and the Matlab-code gave identical results.

The VHDL code was simulated and synthesized in Synopsis environment, which resulted in 670 clb. The design was downloaded into a FPGA, type XC4028EX, and the output was identical with the Matlab and VHDL simulations, thus proving the theory.

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

Preface

This bachelor thesis is the final task of our Bachelor of Science in Electronics at Chalmers University of Technology. The work has been done at Ericsson Microwave Systems AB in Mölndal Sweden, at the department FM/D.

We would like to thank the following people who has been of great help to us during the work.

Our supervisor Rune Olsson, EMW.

Our manager Håkan Enskog.

Lars Bengtsson, supervisor CTH.

Lennart Mathe, for all the ideas and helpful hints.

Fredrik Johansson and Daniel Wallström for help with VHDL.

Andreas Hilvarsson, for help with the realization.

Thomas Lundgren, for help with pattern generator and logic analyzer

Leif Glenvall for help with practical details.

We would also like to thank all the others at FM who has been helpful to us.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

Table of contents

1 INTRODUCTION6

1.1 BACKGROUND6

1.2 TASK6

1.3 GOAL6

2 ORIENTATION7

2.1 INFORMATION SEARCH7

2.2 D/A CONVERTERS7

2.2.1 MULTIBIT CONVERTERS7

2.2.2 ONEBIT CONVERTERS7

3 SIGMA-DELTAMODULATION8

Fig.1. : First order Sigma-Deltamodulator8

3.1 THE TWO TRANSFERFUNCTIONS8

3.2 THE LINEAR MODEL8

Fig.2. : General block diagram9

Fig.3. : General block diagram10

3.3 MODULATOR STRUCTURES13

Fig.4. : Modulator that allows spread root locus13

Fig.5. : Second order modulator13

Fig.6. : Second order modulator with quantizer replaced14

3.4 THE LOWPASS-PROTOTYPE METHOD15

Fig.7. : Lowpass to bandpass transformation15

3.5 MATLAB SIMULATION18

Fig.8. : Simulation second-order LP, Hanning window18

Fig.9. : Simulation fourth-order BP, Hanning window18

4 REALIZATION19

4.1 ADAPTATION OF 2LP -> 4BP TO INTEGER19

4.1.1 THEORY19

4.1.2 METHOD21

Fig.10. : Modulator with a_0 22

Fig.11. : Modulator without a_0 22

Fig.12. : Insertion of constant23

Fig.13. : Modulator with new coefficients23

Fig.14. : Implementation of H-block24

Fig.15. : Implementation of 'a'25

Fig.16. : Implementation of '2a'25

4.2 ADAPTATION OF 4LP -> 8BP TO INTEGER26

Fig.17. : Rescale large coefficients27

Fig.18. : Structure that only uses integer coefficients27

5 HARD QUANTIZATION vs. SIGMA-DELTA28

Fig.19. : Hard quantization, Hanning window28

Fig.20. : Eight-order Sigma-deltamodulator, Hanning window28

6 VHDL29

Fig.21. : General block diagram for a logic block29

7 ANALYSIS30

Fig.22. : Analysis of the results from FPGA, VSIM and Matlab30

8 CONCLUSION31

9 IDEAS FOR FURTHER STUDIES31

10 REFERENCES32

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

APPENDIX

App A: Matlabcode

App B: VHDL-code

App C: Circuitboard

App D: Realization

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

1 INTRODUCTION

1.1 BACKGROUND

Radarsystems transmits and receives pulsed signals. These pulses looks different depending on the application. To create an outgoing puls or waveform a DDS (Direct Digital Synthesis) might be used. One drawback with commercial DDS systems is that they are unnecessarily complicated for Ericssons applications. It is therefor interesting to realize the same function in a FPGA or DSP and a D/A converter.

1.2 TASK

The task of this thesis has been to investigate if it's possible to realize a sigma-delta algorithm in hardware. The following steps has been included:

- Information search and studies of sigma-delta modulators.
- Simulate and evaluate different sigma-delta algorithms.
- Realize a suitable algorithm in hardware with a FPGA.
- Evaluate the performance of the realized modulator.

1.3 GOAL

The goal with this bachelor thesis is to develop a suitable sigma-delta algorithm and there after implement the design in a FPGA.

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog	1999-06-10	A	

2 ORIENTATION

2.1 INFORMATION SEARCH

As a start we searched for information about sigma-deltamodulators and found a lot of articles regarding the subject. Most of these articles focused on low-pass A/D modulators. There were only a few interesting articles, regarding our subject, bandpass D/A modulators. A good place to start the search was at the IEEE database[4]. We also found an interesting book, Delta-Sigma Data Converters[6].

2.2 D/A CONVERTERS

One of the most important parts of a DDS is the D/A converter. It's crucial for the quality of the output signal. There are different types of D/A converters and these can be divided into onebit and multibit converters.

2.2.1 MULTIBIT CONVERTERS

Multibitconversion is handled in Andersson, Ekström [1].

2.2.2 ONEBIT CONVERTERS

In a onebit converter there are only two levels in the conversion, therefore to obtain high resolution the sampling rate must be increased. This is called oversampling. To measure the Over Sampling Ratio, OSR, the following equation is used:

$$OSR = \frac{f_s}{2B}$$

Oversampling

(EQ 1)

To reach an even higher resolution a sigma-delta modulator might be used. This modulator moves the quantization noise away from the band of interest. Then the out of band frequencies may be reduced by filters. One of the advantages with this method is that it's easily made linear. Another advantage is that it can be made from ordinary digital circuits.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

3 SIGMA-DELTAMODULATION

The theories in this chapter are partly the same as in Northsworty, Shreier and Temes[6] (pp 143 and 287).

A D/A converter based on hard quantization witch only looks at the signbit does not become very effective, this is discussed in Andersson, Ekström [1]. There are more effective methods, one of these is the sigma-delta modulator.

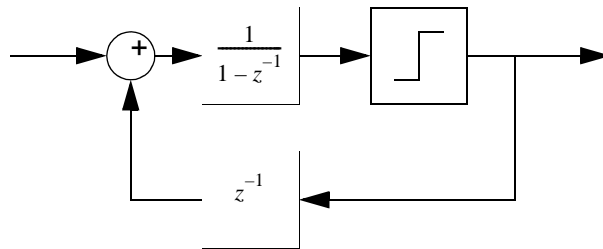


Fig. 1. First order Sigma-Deltamodulator

3.1 THE TWO TRANSFERFUNCTIONS

When you perform D/A conversion using sigma-delta modulation the modulator will move the noise away from the signal. This does not mean that the sum of the noise decreases, only that it is moved out from the band of interests.

One can create a model of the modulator by separating the input signal and the quantization noise into two arbitrary transfer functions, NTF = Noise Transfer Function, STF = Signal Transfer Function.

3.2 THE LINEAR MODEL

Figure 2 shows a general block diagram for a single-quantizer sigma-delta modulator. The modulator is split into a linear block and a nonlinear block, with the linear block having arbitrary transfer functions from its two inputs U and V to its single output Y.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

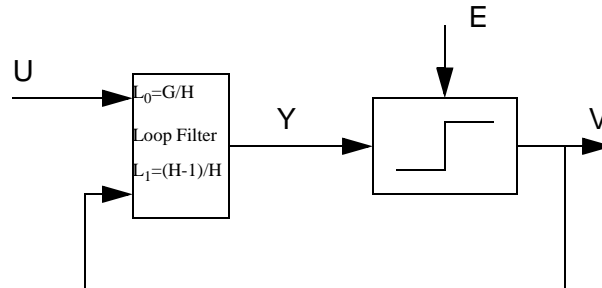


Fig.2. General block diagram

Here the transfer functions has been labeled as:

$$L_0 = \frac{G(z)}{H(z)} \text{ and } L_1 = \frac{H(z-1)}{H(z)}$$

(EQ 2)

With these assignments, the output of the linear block is:

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

(EQ 3)

By defining the error signal E as $E(z) = V(z) - Y(z)$, the output can be written as:

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

(EQ 4)

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

By showing that the output consists of independently filtered signal and noise components, eq.4 captures the essence of noise-shaping loops.

Eq.4 indicates that a sigma-delta modulator with a signal transfer function $G(z)$ and an input $U(z)$ is equivalent to a sigma-delta modulator with a signal transfer function of unity and an input $G(z) * U(z)$. This allows us to focus on the NTF $H(z)$ in the discussions of loop stability since $G(z)$ merely acts as a prefilter on the input.

One drawback of Eq.4 is that it hides the fact that the noise is signal dependent. This omission can lead to serious modeling errors. As one example of such an error, consider the linear model in figure 3.

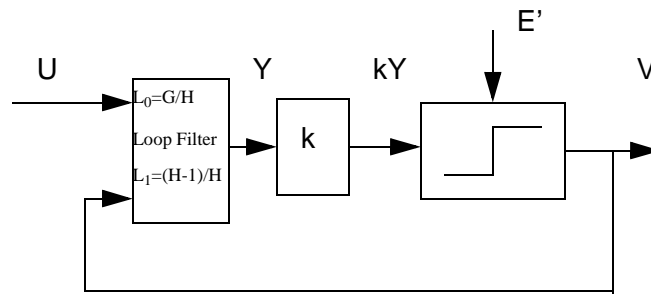


Fig.3. General block diagram

This model is identical to the previous one except for the addition of an arbitrary gain $k > 0$ at the input of the quantizer.

$$V(z) = G'(z) \times U(z) + H'(z) \times E'(z)$$

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

Since the quantizer is binary, this addition does not affect the operation of the modulator, but it does affect the linear model.

The new equations becomes:

$$E'(z) = V(z) - kY(z)$$

(EQ 6)

$$G'(z) = \frac{kG(z)}{k + (1 - k) \times H(z)}$$

(EQ 7)

$$H'(z) = \frac{H(z)}{k + (1 - k) \times H(z)}$$

(EQ 8)

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev File
EMW FM/DC Håkan Enskog	1999-06-10	A	

Thus the NTF and STF are different and may even be unstable!

The issue at hand is "What is the best definition of $E(z)$?" or equivalently "What is the gain of the quantizer?"

This question can be answered by defining the value of k that minimizes the error signals power. This optimum value, k_{opt} decorrolates the error and signal components and is given by:

$$k_{opt} = \frac{cov\langle y, v \rangle}{var\langle y \rangle} = \lim_{N \rightarrow \infty} \frac{\sum_{n=0}^N y(n) \times v(n)}{\sum_{n=0}^N y(n)^2}$$

(EQ 9)

This formula clearly shows that k_{opt} depends on y , which in turn depends on the modulator input U . Consequently one must have a prior knowledge of signal statistics in order to find the optimum linear model, and this model varies as the input varies.

A fixed value of k would be preferable since the designer could then talk about the noise and signal transfer functions, without having to qualify such statements by specifying the input. Unfortunately, this is not possible. This variability of k_{opt} can be viewed as being a cause of instability in high order modulators.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

3.3 MODULATOR STRUCTURES

There are a lot of different ways to realize a sigma-delta modulator. At first we looked at the structure in figure 4. It has some advantages, for example the ability to spread the position of the zeros, which in our case could have been interesting since in this study Ericsson also cares about the out of band noise. This structure however would be impossible to realize with the hardware currently at our disposal, as a large number of multipliers consumes too much space.

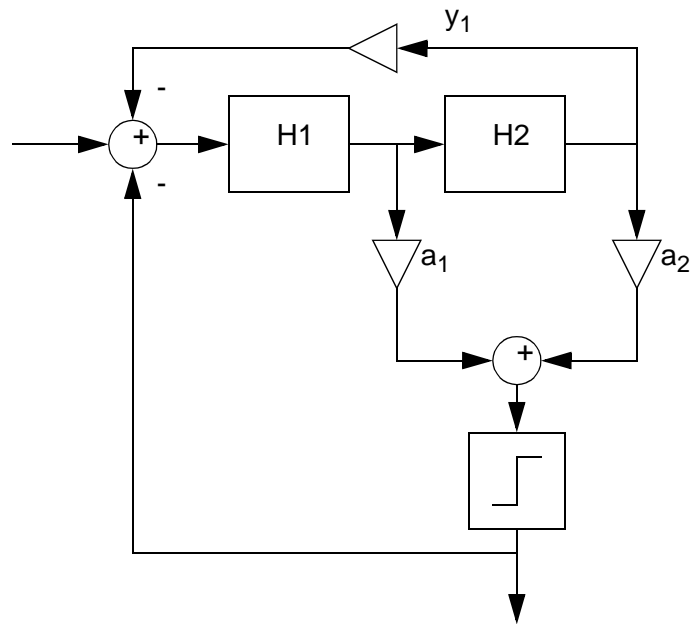


Fig.4. Modulator that allows spread root locus

For this reason we concentrated on another structure with fewer multipliers and finally came up with the structure below.

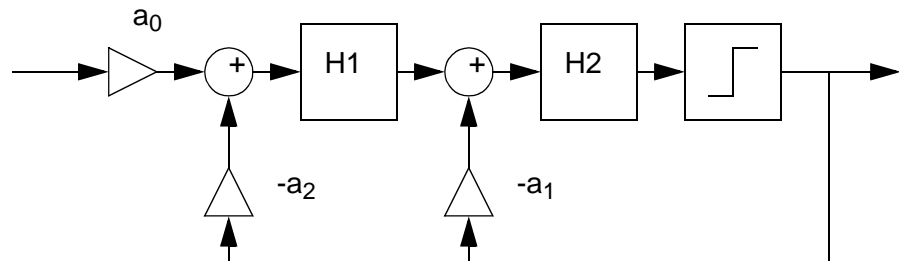


Fig.5. Second order modulator

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

To calculate the feedback constants, a suitable filter family for NTF is chosen. In our case a Butterworth highpass filter was selected. The filter is calculated in Matlab so that the out_of_band_gain becomes about 1,5 which has been proved to be a reasonable value according to Norsworthy, Schreier, Temes [6].

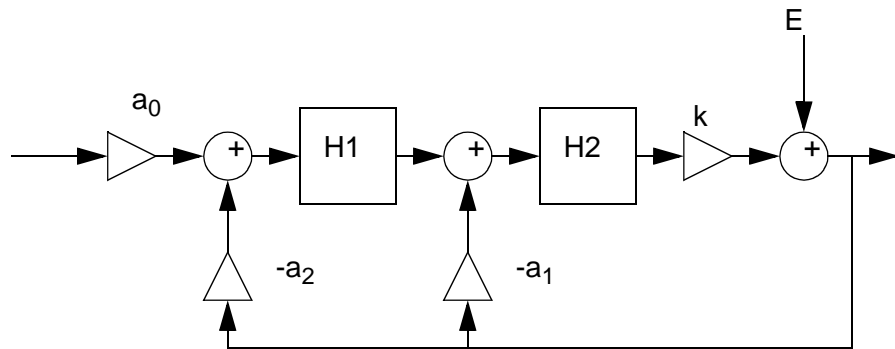


Fig.6. Second order modulator with quantizer replaced

For the modulator in figure 6, the noise transfer function becomes:

$$NTF = \frac{1}{1 + ka_1H + ka_2H^2}$$

NTF Second order modulator

(EQ 10)

The values of the feedback constants are given by comparing the filter coefficients of the poles for NTF and the butterworthfilter. According to Norsworthy, Schreier, Temes [6] the approximation k=1 could be useful, which is confirmed by simulations.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

3.4 THE LOWPASS-PROTOTYPE METHOD

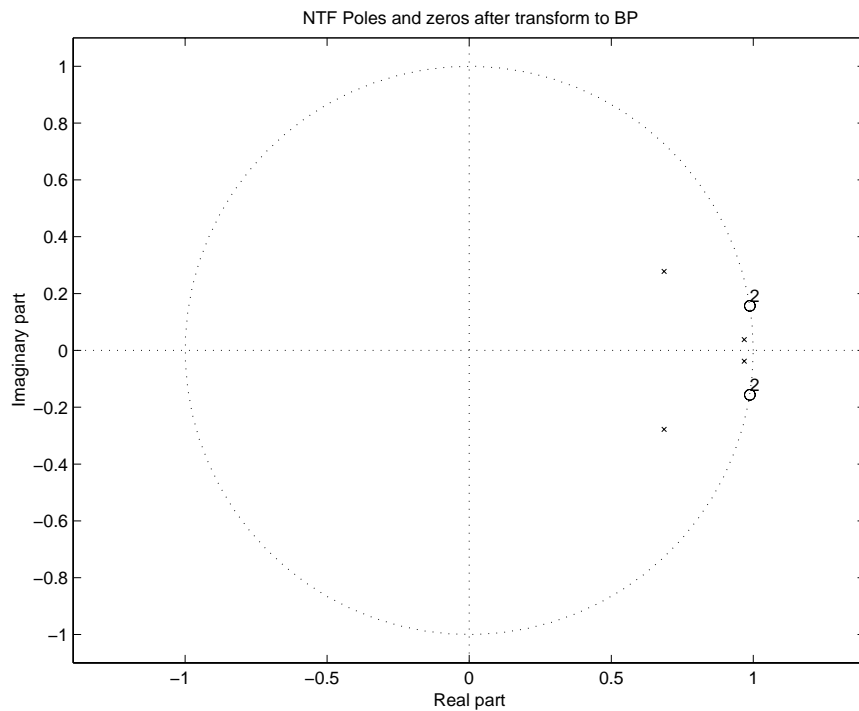
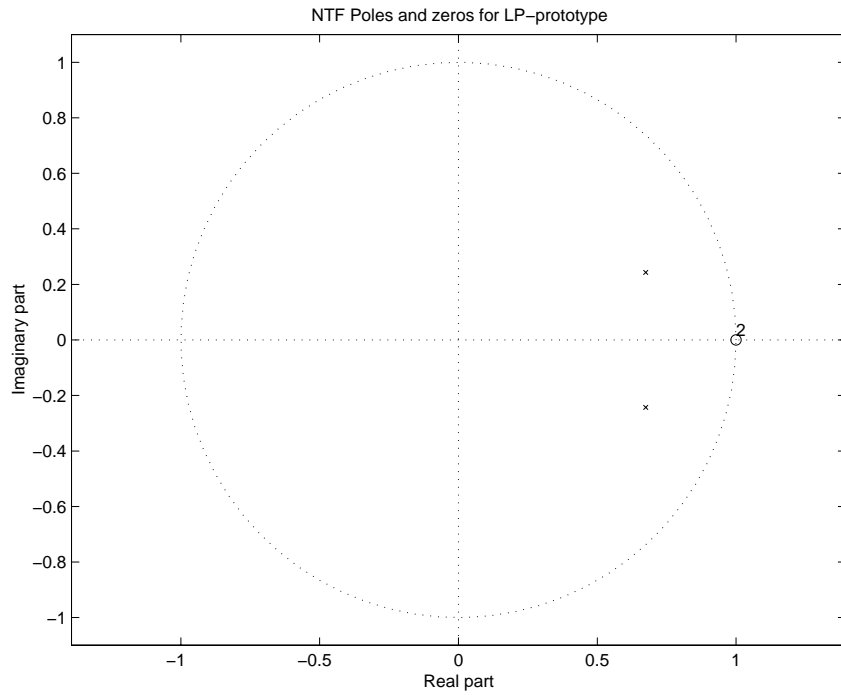


Fig.7. Lowpass to bandpass transformation.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

The simplest way to design $H(z)$ for a bandpass modulator is to start with a suitable lowpass modulator and apply a lowpass to bandpass transformation on it. Such transformations of necessity must increase the order of the modulator. The lowpass prototype must be chosen to satisfy the SNR (Signal to Noise Ratio) specifications with an oversampling ratio that is a function of both the oversampling ratio of the bandpass modulator and the transformation employed.

For example, if one were to apply the transformation $z \rightarrow -z^2$ to a lowpass prototype, the zeros of $H(z)$ would be mapped from dc to $\pm \pi/2$. This transformation places the center frequency at $\omega_0 = \pi/2$, and thus for a fixed center frequency the sampling frequency is dictated by the relation $f_s = 4f_0$. Also, since this transformation preserves the oversampling ratio, the oversampling ratio of the prototype modulator is again determined by the signal parameters:

$$R = \frac{2f_0}{B}$$

(EQ 11)

The $z \rightarrow -z^2$ transformation is a particular attractive one since it does not affect the dynamics of the prototype.

Specifically, the modulator behaves as a pair of multiplexed lowpass modulators with alternate samples of each modulator negated. As a result, the bandpass modulator is stable if and only if the lowpass modulator is stable and the SNR curves of the modulators are identical.

Other transformations, such as generalized N-path transformations and lowpass to bandpass transformations, are possible but does not possess all the advantages of the $z \rightarrow -z^2$ transformation. Generalized N-path transformations $z \rightarrow -z^N$ preserve modulator dynamics but increase the modulator order unnecessarily for $N > 2$ (putting unnecessary passbands) or result in a passband centered at $f_s/2$ (aliasing problems occur). On the other hand, generalized second-order lowpass to bandpass transformations give full control over the passband location but do not preserve modulator dynamics. Since the discrete-time lowpass to bandpass transformation:

$$z \rightarrow -z \times \frac{z+a}{az+1}$$

(EQ 12)

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

Preserves both the realizability and Lee stability constraints[9]. The case $a=0$ degenerates to $z \rightarrow -z^2$; negative a gives systems closer to dc; positive a gives systems with passbands closer to $f_s/2$.

Most of the present bandpass modulators are centered at $\pi/2$, our task however has been to place the center frequency closer to dc.

The effect on a conventional first-order modulator $H_p(z) = 1 - z^{-1}$ is

$$H(z) = H_p\left(-z \times \frac{z+a}{az+1}\right) = 1 + \frac{az+1}{z(z+a)} = \frac{z^2 + 2az + 1}{z(z+a)}$$

(EQ 13)

For a second order modulator $H_p(z) = (1 - z^{-1})^2$.

$$H(z) = \left(\frac{z \times 2az + 1}{z(z+a)}\right)^2$$

(EQ 14)

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

3.5 MATLAB SIMULATION

To verify these theories, they were simulated in Matlab. The output spectrums below shows the simulations for a second order lowpass transformed to a fourth-order bandpass modulator. The over sampling ratio for this design is 45.

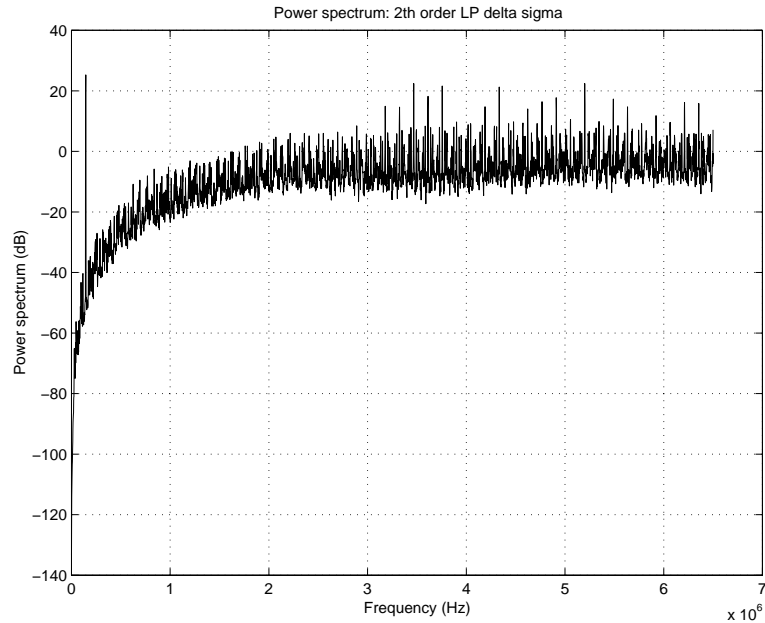


Fig.8. Simulation second-order LP, Hanning window

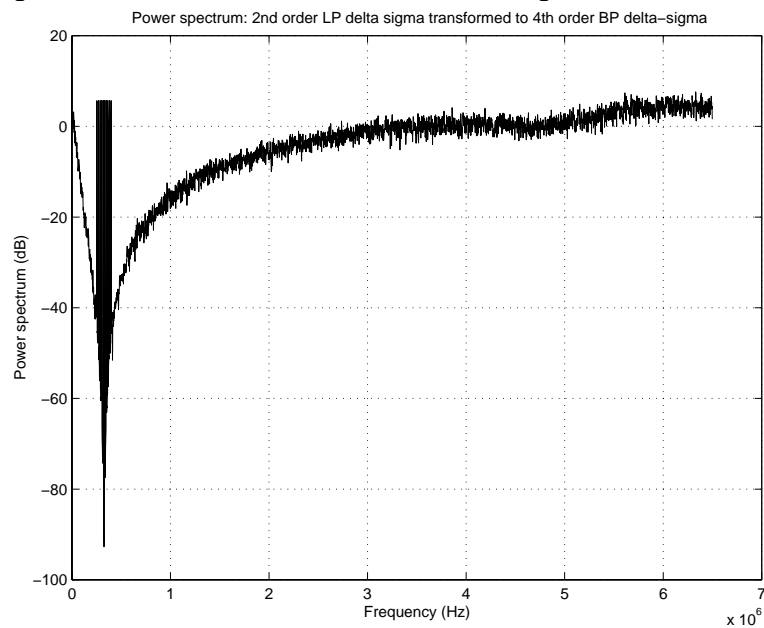


Fig.9. Simulation fourth-order BP, Hanning window

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

4 REALIZATION

In this chapter the transformation theory is discussed further. A method for realization is also discussed.

4.1 ADAPTATION OF 2LP -> 4BP TO INTEGER

4.1.1 THEORY

Lowpass to bandpass transformation by Eq.13 gives:

$$H_{LP}(z) = \frac{1}{z-1} \Rightarrow H_{BP}(z) = \frac{1}{\left(-z \times \frac{z+a}{az+1}\right) - 1} = -\frac{az^{-1} + z^{-2}}{1 + 2az^{-1} + z^{-2}}$$

(EQ 15)

Equations 10 and 15 gives us the NTF of the transformed modulator as:

$$NTF_{BP} = \frac{(z^2 + 2az + 1)^2}{(z^2 + 2az + 1)^2 - ka_1(az + 1)(z^2 + 2az + 1) + ka_2(az + 1)^2}$$

Fourth-order bandpass NTF

(EQ 16)

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

To obtain the zeros at the passband center, the constant 'a' has to be calculated. This is done by solving the following equation.

$$z^2 + 2az + 1 = 0$$

$$(z - e^{j\varphi})(z - e^{-j\varphi}) = z^2 - z(e^{j\varphi} + e^{-j\varphi}) + 1 = z^2 - z2\cos(\varphi) + 1 \Rightarrow a = -\cos(\varphi)$$

Calculating a

(EQ 17)

This gives the output from the H-block as:

$$Y_n = -2aY_{n-1} - Y_{n-2} - aX_{n-1} - X_{n-2}$$

Output from H-block

(EQ 18)

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

4.1.2 METHOD

Step 1: Calculate a_0

a_0 is chosen for a “suitable” STF-gain.

The STF inband gain is measured or calculated with $a_0 = 1$

- a_0 is then selected for a desired gain

Desired gain should be:

- As high as possible for a high SNR
- Low enough to maintain stability for FS (full scale) –input

If its assumed that STF-gain = 1 and NTF out_of_band_gain = 1,5 then usually the sigma-delta modulator will be stable for signals up to 50% of the quantizer feedback. Therefor select STF-gain = 0.5 so that a full scale input will produce a half scale output.

For this particular modulator:

$$a_1 = 0,8570$$

$$a_2 = 0,2698$$

$$\text{NTF out_of_band_gain} = 1,56$$

STF-gain is then measured in simulations to be 11.4 dB = 3.71 times.

Wanted STF gain = 0,5 -> $a_0 = \frac{1}{2} * 1/3,71 = 0,1348$, simulate again and verify STF-gain.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

Step 2: Get rid of a_0

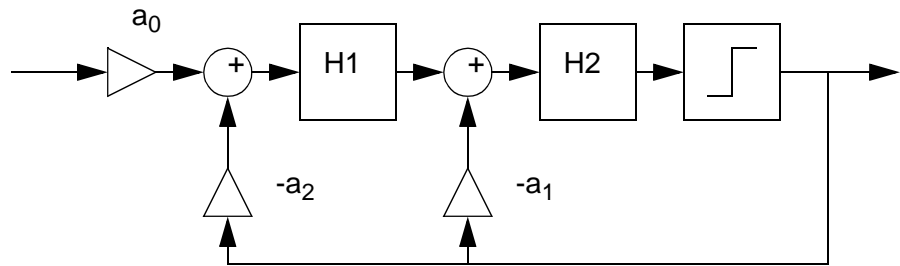


Fig.10. Modulator with a_0

New values for a_1 and a_2 are:

$$a_{2\text{new}} = 0,2698/0,1348 = 2,0000$$

$$a_{1\text{new}} = 0,8570/0,1348 = 6,3577$$

Note that $a_{2\text{new}} = 2,00 = 1/\text{STF-gain}$

This turns out to be true generally, therefore in order to rescale a modulator for a desired STF-gain:

- Select a scalefactor k_s so that $a_{2\text{new}} = k_s * a_{2\text{old}} = 1/\text{STF-gain}$
- Multiply all coefficients with the same scalefactor

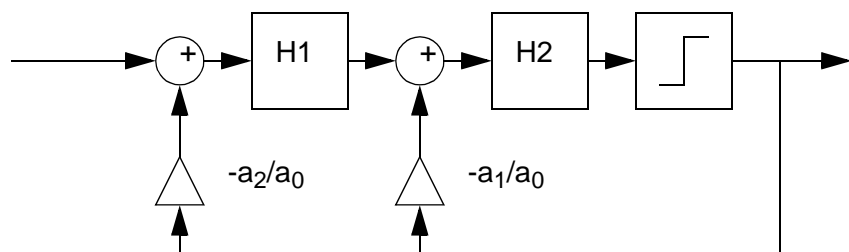


Fig.11. Modulator without a_0

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/ Godkänd - Doc respons/ Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

Step 3: Rescale to fit input wordlength

To rescale the modulator for a 16-bit input, a constant, $k = 2^{15}$ is inserted. The insertion of this constant does not affect the modulators function.

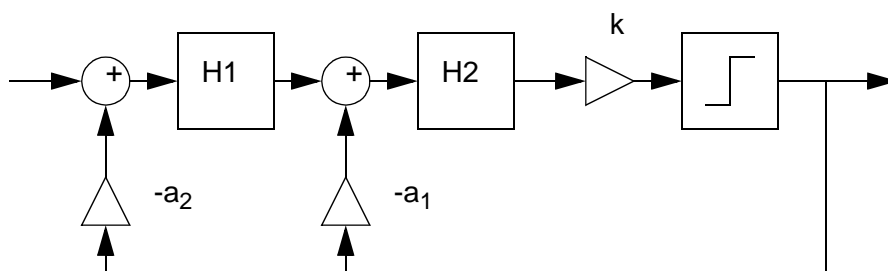


Fig. 12. Insertion of constant.

The coefficients can now be rounded of.

$$a_{2\text{round}} = \text{round}(a_2 * 2^{15}) = 65536$$

$$a_{1\text{round}} = \text{round}(a_1 * 2^{15}) = 208159$$

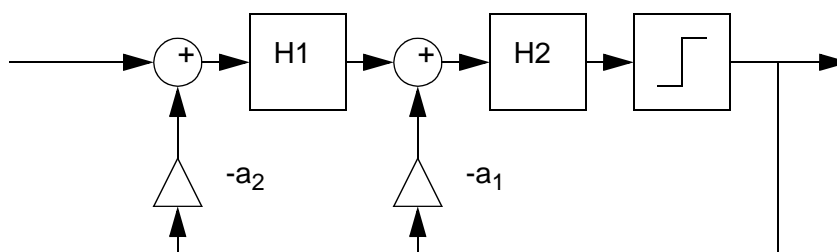


Fig. 13. Modulator with new coefficients.

Observe that $a_1 = a_{1\text{round}}$ and $a_2 = a_{2\text{round}}$

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

Step 4: Implementation of H-block

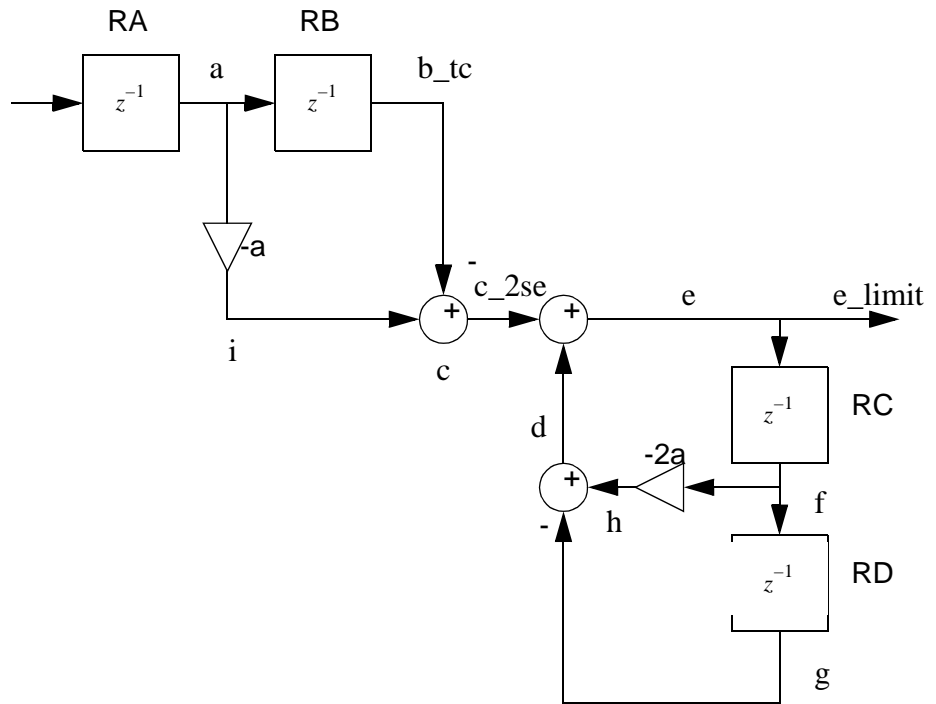


Fig.14. Implementation of H-block

The wordlength in the A,B,C and D registers are determined through simulations and are dependent on the input.

In our case with a center frequency of X MHz and a sampling frequency of Y MHz, the formula becomes:

$$a = -\cos\left(\frac{X\pi}{0,5Y}\right) = -0,9877\dots$$

For simple realization, the constant 'a' was selected to:

$$-a = 253/256 = 1 - 1/128 - 1/256 = 0,9883\dots$$

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

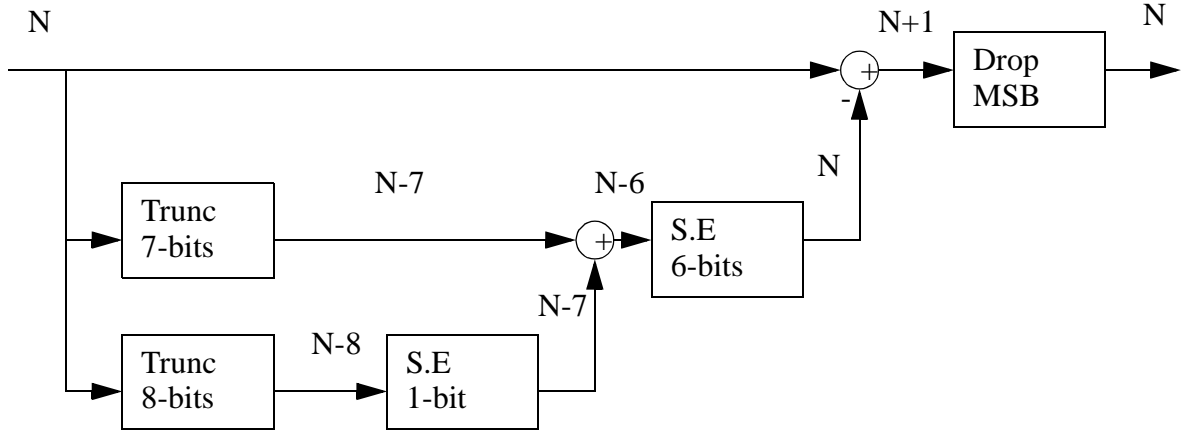


Fig.15. Implementation of 'a'

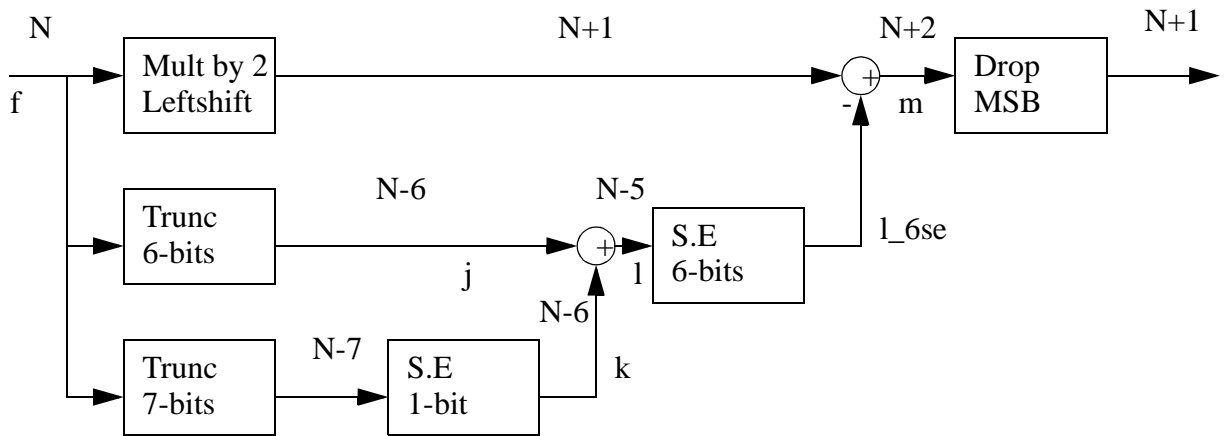


Fig.16. Implementation of '2a'

The names of the variables refers to the VHDL-code.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

4.2 ADAPTATION OF 4LP -> 8BP TO INTEGER

According to the 2LP -> 4BP example, one can:

- Change the STF-gain by performing a multiplication between the a-coefficients and a constant, STF-gain = $1/a_4$
- With a 16-bit input and STF-gain = $\frac{1}{2}$, a_4 then becomes 65536 ($=2^{16}$).

Therefore rescale the a-coefficients so that $a_4 = 65536$.

In this particular case:

$$a_4 = 65536$$

$$a_3 = 671699$$

$$a_2 = 3139146$$

$$a_1 = 8055955$$

Note that some of these constants will become very large, for example, 8055955 is a 24 bit word.

This can be simplified in the following way, see Figure 17, pp 27.

The coefficients are chosen:

- As powers of two for easy realization
- As small as possible to save hardware
- Large enough to maintain performance

Through Matlab simulation the coefficients are chosen:

$$k_1 = 2^{-6}, k_2 = 2^{-3}, k_3 = 2^{-1}$$

These are implemented by truncating 6, 3 and 1 bit after the H-block.

This gives us: $a_4 = 65536$, $a_3 = 10495$, $a_2 = 6131$, $a_1 = 7367$

According to Andersson, Ekström [1] the result will only improve slightly for modulators of higher order than eight. Therefore we have chosen an eight-order modulator for our application.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

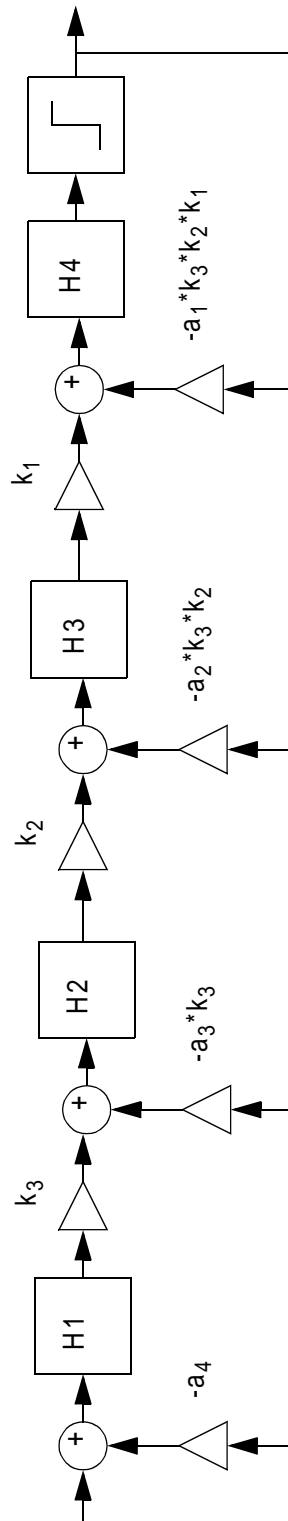


Fig.17. Rescale large coefficients

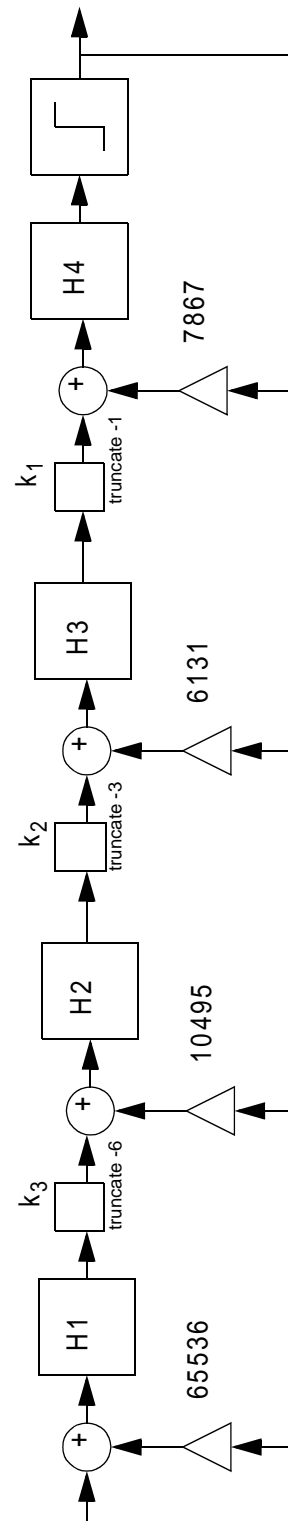


Fig.18. Structure that only uses integer coefficients

Fig.17. Rescale large coefficients

Fig.18. Structure that only uses integer coefficients

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

5 HARD QUANTIZATION vs. SIGMA-DELTA

The most simple way to perform onebit D/A conversion is through hard quantization (see Andersson, Ekström [1]). Obviously it would be interesting to compare this method with sigma-delta and check the performance of both methods.

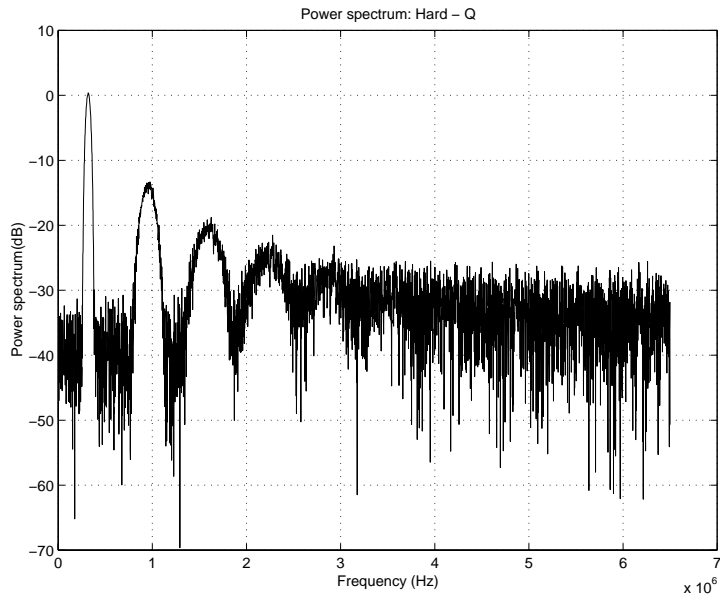


Fig.19. Hard quanzization, Hanning window

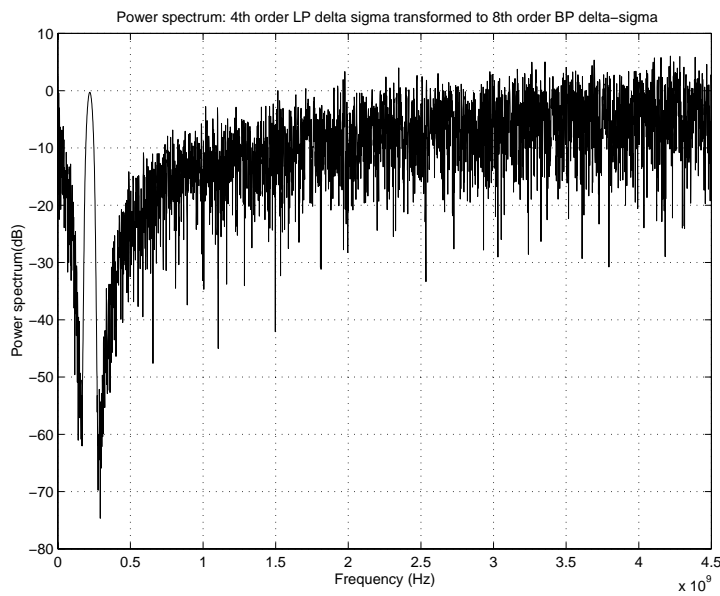


Fig.20. Eight-order Sigma-deltamodulator, Hanning window

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

6

VHDL

The realization of the modulator was made in VHDL (Very high speed integrated circuit Hardware Description Language).

We decided to split the design hierarchically in two levels. One top block and four logic blocks.

The top block only contains connections between the input, the four logicblocks and the output.

Each logic block contains one H-block, one feedback coefficient, one adder between these and a truncation of the output. The fourth logic block does not contain a truncation, instead it makes a quantization of the output.

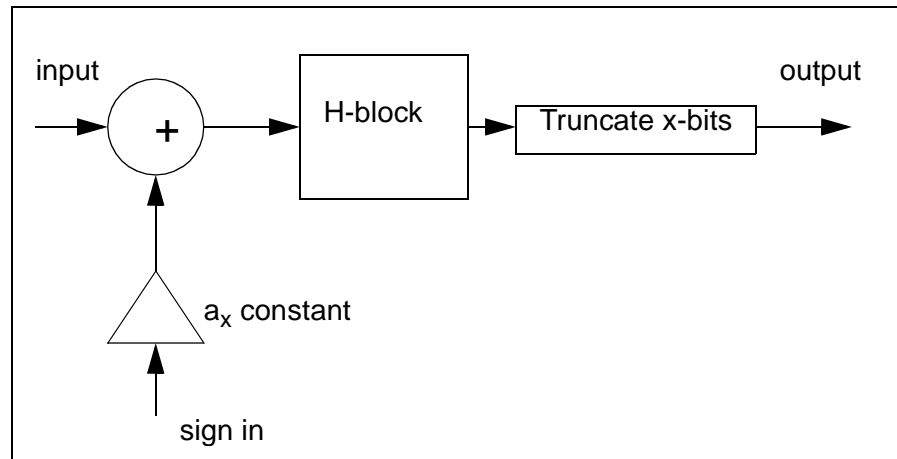


Fig.21. General block diagram for a logic block

The logicblocks are divided into five processes: *register process*, *multiplication with a*, *multiplication with 2a* and a *process for the rest of the logic*. Only the register process is clocked, the rest are combinatorial.

All negative values are represented as two-complement.

The design was synthesized in Synopsis environment, for a FPGA type XC4028EX-3. The clockrate is limited to 13 MHz. This limit is set by the speed of the adders. The design used 670 clb.

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

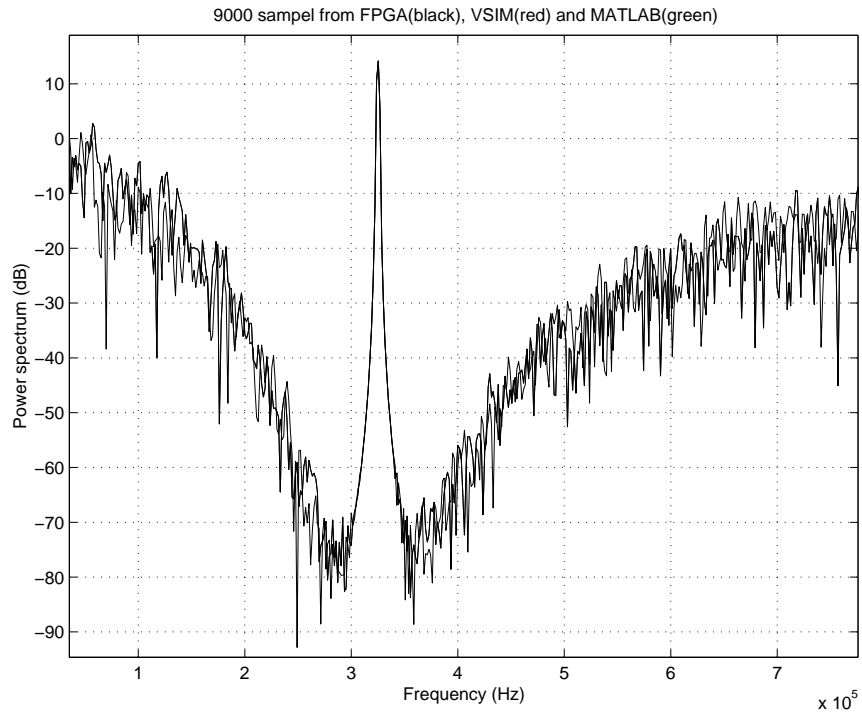


Fig.22. Analysis of the results from FPGA, VSIM and Matlab

The design was downloaded into a FPGA and the analysis of the output was identical with the simulations of both the Matlab and the VHDL-code.

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

8

CONCLUSION

- Eight-order sigma-delta modulation is a better modulation method, then hard quantization, regarding the inband noise.
- Sigma-delta is particularly interesting when a high over-sampling ratio is possible.
- It is possible to realize a bandpass modulator with a center frequency separated from $F_s/4$.
- For high clockrates the limit for the overall modulator is set by the speed of the adders in the design.
- For easy implementation in FPGA, a modulator structure that uses few and simple multipliers is preferable.
- Our eight-order sigma-delta modulator used 670 clb.

9

IDEAS FOR FURTHER STUDIES

In order to reach higher clockrates a larger FPGA might be used. When synthesized for a FPGA type XC40150, the clockrate could be increased to over 30 MHz. For even higher clockrates the VHDL-code has to be modified.

A different structure of the modulator, for example the structure presented in figure 4, that allows spread zeros, could be interesting for reducing the out of band noise. One should however note that the phase response is affected.

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

REFERENCES

- [1] **Andersson, L., Ekström, P.**: 'Enbits Vågformsgenerering', Ericsson Microwave System AB, 1998
- [2] **Bazarjani, S., Snelgrove, M.**: 'A 40 Mhz IF fourth-order double-sampled SC bandpass sigma-delta modulator', IEEE International Symposium on Circuits and System, June 9-12, 1997, pp 73-76
- [3] **Fremrot, P., Frännhagen, M.**: 'Sigma-Delta D/A-converters', Ericsson Mobile Communications AB, 1996
- [4] **IEEE database**: <http://www.iel.ihs.com> (Acc 1999-03-25)
- [5] **Jantzi, S. A., Snelgrove, M. Furguson, P.F.**: 'A 4-th order bandpass sigma-delta modulator', IEEE 1992 custom integrated circuit conference
- [6] **Norsworthy, S.R, Schreier, R., Temes, C.**: 'Delta-Sigma data converters, Theory, Design, and Simulation', IEEE Press, 1997, ISBN: 0-7803-1045-4
- [7] **Schreier, R.**: 'An empirical study of high-order single-bit delta-sigma modulators', IEEE Analog and digital processing, Vol. 40, No. 8, August 1993, pp 461-466
- [8] **Schreier, R.**: 'Bandpass delta-sigma converters', IEEE, 1996, pp 94-97
- [9] **W.L.Lee**: 'A novel higher order interpolative modulator topology for high resolution oversampling A/D converters', Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1987

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.		
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev File
EMW FM/DC Håkan Enskog		1999-06-10	A

Appendix A: Matlabcode

Comments and descriptions of Matlab-code.

m_2lp.m

Description:

Simulation of a 2:nd order lowpass delta-sigma. Simulates the whole process. Plots desired figures, prints out_of_band_gain and a-values.

m_4lp.m

Description:

Simulation of a 4:th order lowpass delta-sigma. Simulates the whole process. Plots desired figures, prints out_of_band_gain and a-values.

m_2lp_4bp.m

Description:

Simulation of a 2:nd order lowpass delta-sigma transformed into a 4:th order bandpass. Simulates the whole process. Plots desired figures, prints out_of_band_gain and a-values.

p_4lp_8bp.m

Description:

Shows that the transformation 4:th order lowpass to 8:th order bandpass is possible. Plots interesting data.

fchirp.m

Description:

Function that generates a chirp. Returns a sinus and a cosinus vector.

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

r_4lp_8bp_hard.m

Description:

Simulation of a 4:th order lowpass delta-sigma transformed into a 8:th order bandpass and plots a comparison of hard quantization. Simulates the whole process. Plots desired figures, prints out_of_band_gain and a-values.

int_8bp.m

Description:

Simulation of a 8:th order bandpass delta-sigma with integer constants. Simulates the whole process. Plots desired figures, prints out_of_band_gain, a-values and the required number of bits in the registers.

output.m

Description:

Loads data from VHDL-code simulation, FPGA-realisation and Matlab simulation. Plots the result in one diagram.

Table of contents

1	m_2lp.m	3
2	m_4lp.m	5
3	m_2lp_4bp.m	8
4	p_4lp_8bp.m	11
5	fchirp.m	13
6	r_4lp_8bp_hard.m	14
7	int_8bp.m	17
8	output.m	24

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

1 **M 2LP.M**

%% Example 2th order delta-sigma LP%

%%%%%%%%%% SETUP input Data Data %%%%%%%%%

npoints = 100000;

% Number of data points in simulation

nfft = 8192;

% FFT Resolution

Fs = XXX;

% Sample rate

f = 100e6;

% Test tone

A = 0.5;

% Amplituder

% Bin align to fft.

Hzbin = Fs / nfft;

binno = round(f / Hzbin);

f = binno * Hzbin;

x = A * sin(2*pi*f/Fs*[0:npoints-1]);

%%%%%%%%%% SETUP NTF FILTER %%%%%%%%%

[B, A] = butter(2, 0.18,'high');

B=B/B(1); A=A/A(1);

H = abs(freqz(B,A, [0 pi]));

out_of_band_gain = H(2)

%%%%%%%%%% CALCULATE a0 a1 a2 a3 a4 %%%%%%%%%

a1 = A(2)+2;

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

```
a2 = A(3)-1+a1;
```

```
a0 = a2;
```

```
%%%%%%%%%%%%% DISPLAY CALCULATED COEFICIENTS %%%%%%%%%%%%%%
```

```
disp(sprintf('[a0 a1 a2] = [%1.3f %1.3f %1.3f]', a0, a1, a2));
```

```
%%%%%%%%%%%%% Perform 4:th order sigma-delta modulation %%%%%%%%%%%%%%
```

```
y = zeros(size(x));
```

```
R1=0; R2=0;
```

```
%Integrator registers, R2 is closest to the quantizer
```

```
Vectors R1v - R2v are only for inspecting the register contents
```

```
R1v = zeros(1, length(x));
```

```
R2v = zeros(1, length(x));
```

```
disp('Starting sigma-delta');
```

```
for k = 1:length(x)
```

```
% Calculate output as a function of current register content..
```

```
Y = 2 * (R2 >= 0) - 1;
```

```
% Y=1 if R2 >= 0, and -1 otherwise
```

```
(k) = Y;
```

```
% Calculate what next reg contents will be
```

```
    R1next = R1 - a2 * Y + a0 * x(k);
```

```
    R2next = R2 - a1 * Y + R1;
```

```
% Save it (for inspection later)
```

```
    R1v(k) = R1;
```

```
    R2v(k) = R2;
```

```
% Apply clock to registers..
```

```
    R1 = R1next;
```

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

```

R2 = R2next;

end

%%%%%%%%%% Calculate output power spectrum %%%%%%%%%%

disp('Starting power spectrum calculation');

figure;

[Pxx F] = psd(y, nfft, Fs);

plot(F, 10*log10(Pxx)); grid on;

ylabel('Power spectrum (dB)');

title('Power spectrum: 2th order LP delta sigma');

xlabel('Frequency (Hz)');

%%%%%%%%%% Calculate Z-plane %%%%%%%%%%

disp('Starting calculating Z-plane');

figure;

zplane(B,A);

title('Z-plane pole zero plot');

```

2 **M 4LP.M**

```

%% Example 4th order delta-sigma%

%%%%%%%%%% Setup input Data %%%%%%%%%%

npoints = 100000;

% Number of data points in simulation

nfft = 8192;

% FFT Resolution

Fs = XXX;

% Sample rate

```

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

f = 100e6;

% Test tone

A = 0.5;

% Amplituder

% Bin align to fft.

Hzbin = Fs / nfft;

binno = round(f / Hzbin);

f = binno * Hzbin;

x = A * sin(2*pi*f/Fs*[0:npoints-1]);

%%%%%%%%%% SETUP NTF FILTER %%%%%%%%%%%

[B, A] = butter(4, 0.10, 'high');

B=B/B(1); A=A/A(1);

H = abs(freqz(B,A, [0 pi]));

out_of_band_gain = H(2)

%%%%%%%%%% CALCULATE a0 a1 a2 a3 a4 %%%%%%%%%%%

a1 = +4 + A(2);

a2 = -6 + 3*a1 + A(3);

a3 = +4 - 3*a1 + 2*a2 + A(4);

a4 = -1 + a1 - a2 + a3 + A(5);

a0 = a4;

%%%%%%%%%% DISPLAY CALCULATED COEFICIENTS %%%%%%%%%%%

disp(sprintf('[a0 a1 a2 a3 a4] = [%1.3f %1.3f %1.3f %1.3f %1.3f]', a0, a1, a2, a3, a4));

%%%%%%%%%% Perform 4:th order sigma-delta modulation %%%%%%%%%%%

y = zeros(size(x));

R1=0; R2=0; R3=0; R4=0;

%Integrator registers, R4 is closest to the quantizer

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

%Vectors R1v - R4v are only for inspecting the register contents

R1v = zeros(1, length(x));

R2v = zeros(1, length(x));

R3v = zeros(1, length(x));

R4v = zeros(1, length(x));

disp('Starting sigma-delta');

for k = 1:length(x)

% Calculate output as a function of current register content.

Y = 2 * (R4 >= 0) - 1;

% Y=1 if R4 >= 0, and -1 otherwise

(k) = Y;

% Calculate what next reg contents will be

R1next = R1 - a4 * Y + a0 * x(k);

R2next = R2 - a3 * Y + R1;

R3next = R3 - a2 * Y + R2;

R4next = R4 - a1 * Y + R3;

% Save it (for inspection later)

R1v(k) = R1;

R2v(k) = R2;

R3v(k) = R3;

R4v(k) = R4;

% Apply clock to registers..

R1 = R1next;

R2 = R2next;

R3 = R3next;

R4 = R4next;

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

end

%%%%%%%%%% Calculate output power spectrum %%%%%%%%%%

disp('Starting power spectrum calculation');

figure;

[Pxx F] = psd(y, nfft, Fs);

plot(F, 10*log10(Pxx)); grid on;

ylabel('Power spectrum (dB)');

title('Power spectrum: 4th order LP delta sigma');

xlabel('Frequency (Hz)');

%%%%%%%%%% Calculate Z-plane %%%%%%%%%%

disp('Starting calculating Z-plane');

figure;

zplane(B,A);

title('Z-plane pole zero plot');

3 M 2LP 4BP.M

% Example 2nd order LP delta-sigma transformed to 4th order bandpass delta-sigma%

%%%%%%%%%% Setup input Data %%%%%%%%%%

npoints = 100000;

%Number of data points in simulation

nfft = 8192;

% FFT Resolution

Fs = XXX;

% Sample rate

A = 0.01;

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

% test tone standard deviation

% Multitone test signal

x = zeros(1, npoints);

Hzbin = Fs / nfft;

for f = linspace(175eX,275eX, 10);

 f1 = round(f / Hzbin) * Hzbin;

 % Bin align

 x = x + sqrt(2) * A * sin(2*pi*f1/Fs*[0:npoints-1]);

end

%%%%%%%%%% SETUP NTF FILTER %%%%%%%%%%

[B, A] = butter(2, 0.20,'high');

B=B/B(1); A=A/A(1);

H = abs(freqz(B,A, [0 pi]));

out_of_band_gain = H(2)

%%%%%%%%%% CALCULATE a0 a1 a2 a3 a4 %%%%%%%%%%

a1=A(2)+2;

a2=A(3)-1+a1;

a0 = a2;

%%%%%%%%%% CALCULATE a for the BP transformation %%%%%%%%%%

a = -cos(225*pi/4500);

%%%%%%%%%% DISPLAY CALCULATED COEFICIENTS %%%%%%%%%%

disp(sprintf('[a0 a1 a2] = [%1.3f %1.3f %1.3f]', a0, a1, a2));

%%%%%%%%%% Perform 4:th order sigma-delta modulation %%%%%%%%%%

y = zeros(size(x));

R1A=0; R1B=0; R1C=0; R1D=0;

R2A=0; R2B=0; R2C=0; R2D=0;

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

% Vectors R1v - R4v are only for inspecting the register contents

R1Av = zeros(1, length(x));

R1Bv = zeros(1, length(x));

R1Cv = zeros(1, length(x));

R1Dv = zeros(1, length(x));

R2Av = zeros(1, length(x));

R2Bv = zeros(1, length(x));

R2Cv = zeros(1, length(x));

R2Dv = zeros(1, length(x));

disp('Starting sigma-delta');

for k = 1:length(x)

 % Calculate H filter outputs

 T1 = -sum([a 1 2*a 1] .* [R1A R1B R1C R1D]);

 T2 = -sum([a 1 2*a 1] .* [R2A R2B R2C R2D]);

 % Calculate Y output

 Y = 2 * (T2 >= 0) - 1;

 % Y=1 if T2 >= 0, and -1 otherwise

 y(k) = Y;

 % Calculate what next reg contents will be

 R1Anext = x(k) - a2 * Y;

 R1Bnext = R1A;

 R1Cnext = T1;

 R1Dnext = R1C;

 R2Anext = T1 - a1 * Y;

 R2Bnext = R2A;

 R2Cnext = T2;

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

R2Dnext = R2C;

% Save it (for inspection later)

R1Av(k) = R1A; R1Bv(k) = R1B; R1Cv(k) = R1C; R1Dv(k) = R1D;

R2Av(k) = R2A; R2Bv(k) = R2B; R2Cv(k) = R2C; R2Dv(k) = R2D;

% Apply clock to registers..

R1A = R1Anext; R1B = R1Bnext; R1C = R1Cnext; R1D = R1Dnext;

R2A = R2Anext; R2B = R2Bnext; R2C = R2Cnext; R2D = R2Dnext;

end

%%%%%%%%%% Calculate output power spectrum %%%%%%%%%%

disp('Starting power spectrum calculation');

figure;

[Pxx F] = psd(y, nfft, Fs);

plot(F, 10*log10(Pxx)); grid on;

ylabel('Power spectrum (dB)');

title('Power spectrum: 2nd order LP delta sigma transformed to 4th order BP delta-sigma');

xlabel('Frequency (Hz)');

4 **P 4LP 8BP.M**

% Calculate NTF and STF for a 4th order LP delta-sigma structure based on fig 5.8 which is
%transformed

% to a 8th order BP delta-sigma structure based on transformation (9.4) in the yellow book

%Unit in Mhz

Fs = XXX;

% Sample rate

Fm = XXX;

% Passband center

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

```
%Setup LP filter NTF
```

```
[B,A] = butter(4, 0.10, 'high'); B=B/B(1); A=A/A(1);
```

```
H=abs(freqz(B,A,[0 pi])); out_of_band_gain_LP = H(2)
```

```
% Calculate a1, a2, a3 and a4 coefficients (as in fig 5.8, but coefficients swapped)
```

```
a1 = +4 + A(2);
```

```
a2 = -6 + 3*a1 + A(3);
```

```
a3 = +4 - 3*a1 + 2*a2 + A(4);
```

```
a4 = -1 + a1 - a2 + a3 + A(5);
```

```
a0 = a4;
```

```
%Check LP
```

```
figure; zplane(B,A);
```

```
title('Poles and zeroes for LP-prototype');
```

```
figure; freqz(B,A, 1000, Fs);
```

```
title('NTF for LP prototype');
```

```
figure; freqz(1,A, 1000, Fs);
```

```
title('STF for LP prototype');
```

```
% Check NTF for BP transform:
```

```
k = 1;
```

```
a = -cos(XXX*pi/(Fs/2));
```

```
Bntf = [(1) (8*a) (4+24*a^2) (24*a+32*a^3) (6+48*a^2+16*a^4) (24*a+32*a^3) (4+24*a^2) (8*a) (1)]
```

```
Antf = [(1) (8*a-a*a1) (4+24*a^2-a1-6*a^2*a1+a^2*a2) (24*a+32*a^3-9*a*a1-12*a^3*a1+2*a*a2+4*a^3*a2-a^3*a3) (6+48*a^2+16*a^4-3*a1-24*a^2*a1-8*a^4*a1+a2+10*a^2*a2+4*a^4*a2-3*a^2*a3-2*a^4*a3+a^4*a4) (24*a+32*a^3-15*a*a1-20*a^3*a1+8*a*a2+12*a^3*a2-3*a*a3-7*a^3*a3+4*a^3*a4) (4+24*a^2-3*a1-18*a^2*a1+2*a2+13*a^2*a2-a3-9*a^2*a3+6*a^2*a4) (8*a-7*a*a1+6*a*a2-5*a*a3+4*a*a4) (1-a1+a2-a3+a4)]
```

```
Bstf = k*conv([(a^2) (2*a) 1], [(a^2) (2*a) 1]);
```

```
Astf = Antf
```

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

% Check BP

H=abs(freqz(Bntf,Antf, [0 pi])); out_of_band_gain_BP = H(2)

figure; zplane(Bntf,Antf);

title('Poles and zeroes after transform to BP');

figure; freqz(Bntf,Antf, 1000, Fs);

title('NTF after transform to BP');

figure; freqz(Bstf,Astf, 1000, Fs);

title('STF after transform to BP');

5 FCHIRP.M

%By Rune Olsson%

%Funktion chirp [yi,yq]=Fchirp(fo,B,T,fs);

%B=Bandbredd[Hz]

%T= pulslangd[s]

%fs=samplingstakt[Hz]

%fo=barvagsfrekvens[Hz]

function [yi,yq]=Fchirp(fo,B,T,fs);

%fmom=fo-B/2+B*t/T;

N=round(T*fs);

t=(1:N)/fs;

fas=fo*t-B/2*t+1/2*B/T*t.^2;

yi=cos(2*pi*fas);

yq=sin(2*pi*fas);

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

6 **R 4LP 8BP HARD.M**

```
% Example 4th order LP delta-sigma transformed to 8th order bandpassdelta-sigma %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Setup input Data %%%%%%%%%
npoints = 60000;    % Number of data points in simulation

nfft = 8192;        %,FFT Resolution

Fs = XXX;          % Sample rate

A = 0.001;         % test signal standard deviation

% Chirp a' la Rune

B = XXX;

fo = XXX;

T = 1e-6;

[x,yq]=Fchirp(fo,B,T,Fs);

x = A * x/std(x);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SETUP NTF FILTER %%%%%%%%%

[B, A] = butter(4, 0.10, 'high');

B=B/B(1); A=A/A(1);

H = abs(freqz(B,A, [0 pi]));

out_of_band_gain = H(2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% CALCULATE a0 a1 a2 a3 a4 %%%%%%%%%

a1 = +4 + A(2);

a2 = -6 + 3*a1 + A(3);

a3 = +4 - 3*a1 + 2*a2 + A(4);

a4 = -1 + a1 - a2 + a3 + A(5);

a0 = a4;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DISPLAY CALCULATED COEFICIENTS %%%%%%%%%

disp(sprintf('[a0 a1 a2 a3 a4] = [%1.3f %1.3f %1.3f %1.3f %1.3f]', a0, a1, a2, a3, a4));
```

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

%%%%%%%%%% CALCULATE a for the BP transformation %%%%%%%%%%

%aa1 = -cos(XXX*pi/XXXX);

%aa2 = -cos(XXX*pi/XXXX);

%aa3 = -cos(XXX*pi/XXXX);

%aa4 = -cos(XXX*pi/XXX);

%Approximation that makes multipliers simple

aa1 = -253/256;

aa2 = -253/256;

aa3 = -253/256;

aa4 = -253/256;

%%%%%%%%%% Perform 8:th order BP sigma-delta modulation %%%%%%%%%%

y = zeros(size(x));

R1A=0; R1B=0; R1C=0; R1D=0;

R2A=0; R2B=0; R2C=0; R2D=0;

R3A=0; R3B=0; R3C=0; R3D=0;

R4A=0; R4B=0; R4C=0; R4D=0;

disp('Starting sigma-delta');

for k = 1:length(x)

%Calculate H filter outputs

T1 = -sum([aa1 1 2*aa1 1] .* [R1A R1B R1C R1D]);

T2 = -sum([aa2 1 2*aa2 1] .* [R2A R2B R2C R2D]);

T3 = -sum([aa3 1 2*aa3 1] .* [R3A R3B R3C R3D]);

T4 = -sum([aa4 1 2*aa4 1] .* [R4A R4B R4C R4D]);

%Calculate Y output

Y = 2 * (T4 >= 0) - 1;

%Y=1 if T2 >= 0, and -1 otherwise

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

y(k) = Y;

%Calculate what next reg contents will be

R1Anext = x(k) - a4 * Y;

R1Bnext = R1A;

R1Cnext = T1;

R1Dnext = R1C;

R2Anext = T1 - a3 * Y;

R2Bnext = R2A;

R2Cnext = T2;

R2Dnext = R2C;

R3Anext = T2 - a2 * Y;

R3Bnext = R3A;

R3Cnext = T3;

R3Dnext = R3C;

R4Anext = T3 - a1 * Y;

R4Bnext = R4A;

R4Cnext = T4;

R4Dnext = R4C;

% Apply clock to registers..

R1A = R1Anext; R1B = R1Bnext; R1C = R1Cnext; R1D = R1Dnext;

R2A = R2Anext; R2B = R2Bnext; R2C = R2Cnext; R2D = R2Dnext;

R3A = R3Anext; R3B = R3Bnext; R3C = R3Cnext; R3D = R3Dnext;

R4A = R4Anext; R4B = R4Bnext; R4C = R4Cnext; R4D = R4Dnext;

end

%%%%%%%%%% Calculate output power spectrum %%%%%%%%%%

figure;

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

```
[Pxx F] = psd(y, nfft, Fs);

plot(F, 10*log10(Pxx)-5); grid on; zoom on;

ylabel('Power spectrum(dB)');

title('Power spectrum: 4th order LP delta sigma transformed to 8th order BP delta-sigma');

xlabel('Frequency (Hz)');

figure;

c = sign(x);

[Py G] = psd(c, nfft, Fs);

plot(G, 10*log10(Py)-20); grid on; zoom on;

ylabel('Power spectrum(dB)');

title('Power spectrum: Hard - Q');

xlabel('Frequency (Hz)');
```

7 **INT 8BP.M**

```
%% Example 4th order LP delta-sigma transformed to 8th order bandpass delta-sigma %
% This version is a integer implementaion %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Setup input Data %%%%%%%%%
num_bits = 16;

% Number of bits in input signal
nt3 = 6;

% Number of bits truncated after first H-block
nt2 = 3;

% Number of bits truncated after second H-block
nt1 = 1;

% Number of bits truncated after third H-block
```

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

```
npoints = 9000;

% Number of data points in simulation

nfft = 8192;

% FFT Resolution

Fs = XXXX;

% Sample rate

A = 0.2;

% test signal standard deviation (relative full scale)

STFgain = 0.5;

% STF gain (relative FS for input and output)

% XXX MHz wide test signal

% [Be, Ae] = ellip(5, 0.5, 60, [XXX XXX]/XXXX);

% x = filter(Be, Ae, randn(1, npoints+1000)); x = x(1001:end);

% Multitone test signal

x = zeros(1, npoints);

Hzbin = Fs / nfft;

%for f = linspace(XXX,XXX, 10);

% f1 = round(f / Hzbin) * Hzbin;

% Bin align

% x = x + sin(2*pi*f1/Fs*[0:npoints-1]);

%end

% Chirp a' la Rune

%B = XXX;

%fo = XXX;

%T = 1e-6;

%[x,yq]=Fchirp(fo,B,T,Fs);
```

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A

```

center=XXX;

x= sin(2*pi*center/Fs*[0:npoints-1]);

x = round(2^(num_bits-1) * x/std(x) * A);

%%x= round(2^(num_bits-1) * x);

p = dec2bin(x);

if min(x) < -2^(num_bits-1) | max(x) >= 2^(num_bits-1);

    warning('x does not fit in num_bits bits');

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SETUP NTF FILTER %%%%%%%%%%%%%%%

[B, A] = butter(4, 0.10, 'high');

B=B/B(1); A=A/A(1);

H = abs(freqz(B,A, [0 pi]));

out_of_band_gain = H(2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% CALCULATE a0 a1 a2 a3 a4 %%%%%%%%%%%%%%%

1 = +4 + A(2);

a2 = -6 + 3*a1 + A(3);

a3 = +4 - 3*a1 + 2*a2 + A(4);

a4 = -1 + a1 - a2 + a3 + A(5);

% Rescale to desired STF gain and taking into account the k's

scal = 1/(STFgain*a4) * 2^(num_bits-1);

a4 = round(a4 * scal);

a3 = round(a3 * scal / 2^nt3);

a2 = round(a2 * scal / 2^nt3 / 2^nt2);

a1 = round(a1 * scal / 2^nt3 / 2^nt2 / 2^nt1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DISPLAY CALCULATED COEFICIENTS %%%%%%%%%%%%%%%

disp(sprintf('[a1 a2 a3 a4] = [%1.3f %1.3f %1.3f %1.3f]', a1, a2, a3, a4));

```

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

%%%%%%%%%% CALCULATE a for the BP transformation %%%%%%%%%%

%aa1 = -cos(XXX*pi/XXXX);

%aa2 = -cos(XXX*pi/XXXX);

%aa3 = -cos(XXX*pi/XXXX);

%aa4 = -cos(XXX*pi/XXXX);

%Approximation that makes multipliers simple

aa1 = -253/256;

aa2 = -253/256;

aa3 = -253/256;

aa4 = -253/256;

%%%%%%%%%% Perform 8:th order BP sigma-delta modulation %%%%%%%%%%

y = zeros(size(x));

R1A=0; R1B=0; R1C=0; R1D=0;

R2A=0; R2B=0; R2C=0; R2D=0;

R3A=0; R3B=0; R3C=0; R3D=0;

R4A=0; R4B=0; R4C=0; R4D=0;

% Vectors R1v - R4v are only for inspecting the register contents

R1Av = zeros(1, length(x));R1Bv = zeros(1, length(x));R1Cv = zeros(1, length(x));R1Dv = zeros(1, length(x));

R2Av = zeros(1, length(x));R2Bv = zeros(1, length(x));R2Cv = zeros(1, length(x));R2Dv = zeros(1, length(x));

R3Av = zeros(1, length(x));R3Bv = zeros(1, length(x));R3Cv = zeros(1, length(x));R3Dv = zeros(1, length(x));

R4Av = zeros(1, length(x));R4Bv = zeros(1, length(x));R4Cv = zeros(1, length(x));R4Dv = zeros(1, length(x));

disp('Starting sigma-delta');

for k = 1:length(x)

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

% Calculate H filter outputs

% T1 = -sum([aa1 1 2*aa1 1] .* [R1A R1B R1C R1D]);

% T2 = -sum([aa2 1 2*aa2 1] .* [R2A R2B R2C R2D]);

% T3 = -sum([aa3 1 2*aa3 1] .* [R3A R3B R3C R3D]);

% T4 = -sum([aa4 1 2*aa4 1] .* [R4A R4B R4C R4D]);

% Perform all multiplications with (-a) and (-2a)

R1A_times_a = R1A - (floor(R1A/2^7) + floor(R1A/2^8));

R1C_times_2a = 2*R1C - (floor(R1C/2^6) + floor(R1C/2^7));

R2A_times_a = R2A - (floor(R2A/2^7) + floor(R2A/2^8));

R2C_times_2a = 2*R2C - (floor(R2C/2^6) + floor(R2C/2^7));

R3A_times_a = R3A - (floor(R3A/2^7) + floor(R3A/2^8));

R3C_times_2a = 2*R3C - (floor(R3C/2^6) + floor(R3C/2^7));

R4A_times_a = R4A - (floor(R4A/2^7) + floor(R4A/2^8));

R4C_times_2a = 2*R4C - (floor(R4C/2^6) + floor(R4C/2^7));

% Calculate output of each H-block

T1 = (R1A_times_a - R1B + R1C_times_2a - R1D);

T2 = (R2A_times_a - R2B + R2C_times_2a - R2D);

T3 = (R3A_times_a - R3B + R3C_times_2a - R3D);

T4 = (R4A_times_a - R4B + R4C_times_2a - R4D);

% Calculated H-block output after truncating nt3, nt2, nt1 bits

T1_times_k3 = floor(T1 / 2^nt3);

T2_times_k2 = floor(T2 / 2^nt2);

T3_times_k1 = floor(T3 / 2^nt1);

% Save the truncated data for inspection later%

T1v(k) = T1_times_k3;

T2v(k) = T2_times_k2;

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

T3v(k) = T3_times_k1;

% Calculate Y output

Y = 2 * (T4 >= 0) - 1;

% Y=1 if T2 >= 0, and -1 otherwise

y(k) = Y;

% Calculate what next reg contents will be

R1Anext = x(k) - a4 * Y;

R1Bnext = R1A;

R1Cnext = T1;

R1Dnext = R1C;

R2Anext = T1_times_k3 - a3 * Y;

R2Bnext = R2A;

R2Cnext = T2;

R2Dnext = R2C;

R3Anext = T2_times_k2 - a2 * Y;

R3Bnext = R3A;

R3Cnext = T3;

R3Dnext = R3C;

R4Anext = T3_times_k1 - a1 * Y;

R4Bnext = R4A;

R4Cnext = T4;

R4Dnext = R4C;

% Save it (for inspection later)

R1Av(k) = R1A; R1Bv(k) = R1B; R1Cv(k) = R1C; R1Dv(k) = R1D;

R2Av(k) = R2A; R2Bv(k) = R2B; R2Cv(k) = R2C; R2Dv(k) = R2D;

R3Av(k) = R3A; R3Bv(k) = R3B; R3Cv(k) = R3C; R3Dv(k) = R3D;

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

R4Av(k) = R4A; R4Bv(k) = R4B; R4Cv(k) = R4C; R4Dv(k) = R4D;

% Apply clock to registers..

R1A = R1Anext; R1B = R1Bnext; R1C = R1Cnext; R1D = R1Dnext;

R2A = R2Anext; R2B = R2Bnext; R2C = R2Cnext; R2D = R2Dnext;

R3A = R3Anext; R3B = R3Bnext; R3C = R3Cnext; R3D = R3Dnext;

R4A = R4Anext; R4B = R4Bnext; R4C = R4Cnext; R4D = R4Dnext;

end

%%%%%%%%%% Calculate output power spectrum %%%%%%%%%%

figure;

[Pxx F] = psd(y, nfft, Fs);

plot(F, 10*log10(Pxx)); grid on; zoom on;

ylabel('Power spectrum (dB)');

title('Power spectrum: 4th order LP delta sigma transformed to 8th order BP delta-sigma');

xlabel('Frequency (Hz)');

%figure;

%[Pxx F] = psd(x, nfft, Fs);

%plot(F, 10*log10(Pxx)); grid on; zoom on; ylabel('Power spectrum (dB)');

%title('Power spectrum: Input signal to 8th order BP delta-sigma');

%xlabel('Frequency (Hz)');

% Wordlength 1st H-block

fprintf(1, 'Reg R1A: %1.2f bits\n', log2(max(abs(R1Av)))+1);

fprintf(1, 'Reg R1B: %1.2f bits\n', log2(max(abs(R1Bv)))+1);

fprintf(1, 'Reg R1C: %1.2f bits\n', log2(max(abs(R1Cv)))+1);

fprintf(1, 'Reg R1D: %1.2f bits\n', log2(max(abs(R1Dv)))+1);

% Wordlength 2nd H-block

fprintf(1, 'Reg R2A: %1.2f bits\n', log2(max(abs(R2Av)))+1);

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

```
fprintf(1, 'Reg R2B: %1.2f bits\n', log2(max(abs(R2Bv)))+1);
fprintf(1, 'Reg R2C: %1.2f bits\n', log2(max(abs(R2Cv)))+1);
fprintf(1, 'Reg R2D: %1.2f bits\n', log2(max(abs(R2Dv)))+1);
% Wordlength 3rd H-block
fprintf(1, 'Reg R3A: %1.2f bits\n', log2(max(abs(R3Av)))+1);
fprintf(1, 'Reg R3B: %1.2f bits\n', log2(max(abs(R3Bv)))+1);
fprintf(1, 'Reg R3C: %1.2f bits\n', log2(max(abs(R3Cv)))+1);
fprintf(1, 'Reg R3D: %1.2f bits\n', log2(max(abs(R3Dv)))+1);
% Wordlength 4th H-block
fprintf(1, 'Reg R4A: %1.2f bits\n', log2(max(abs(R4Av)))+1);
fprintf(1, 'Reg R4B: %1.2f bits\n', log2(max(abs(R4Bv)))+1);
fprintf(1, 'Reg R4C: %1.2f bits\n', log2(max(abs(R4Cv)))+1);
fprintf(1, 'Reg R4D: %1.2f bits\n', log2(max(abs(R4Dv)))+1);
```

8 OUTPUT.M

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%VSIM%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ut_kod;          %file to take data from (from VSIM)
d = ut_kod;
e = find(d==0);
d(e) = -1;
kod_vsim = d';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FPGA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load logic_analyze; %file to take data from (from FPGA)
c = logic_analyze(:,2); % Column number
a=c(1:9000);
```


Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.	
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen	
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev
EMW FM/DC Håkan Enskog		1999-06-10	A
			File

```
b = find(a==0);
```

```
a(b) = -1;
```

```
kod_fpga = a';
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%MATLAB%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
load matlab_utsig;      %file to take gdata from (MATLAB)
```

```
kod_matlab = y;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
nfft = 8192;
```

```
% FFT Resolution
```

```
Fs = XXXX;
```

```
% Sample rate
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
figure;
```

```
[Pxx F] = psd(kod_fpga, nfft, Fs);
```

```
[Pyy G] = psd(kod_vsim, nfft, Fs);
```

```
[Pzz H] = psd(kod_matlab, nfft, Fs);
```

```
plot(F, 10*log10(Pxx),'k',G, 10*log10(Pyy),'r',H, 10*log10(Pzz),'g'); grid on; zoom on;
```

```
ylabel('Power spectrum (dB)');
```

```
title('9000 sampel from FPGA(black), VSIM(red) and MATLAB(green)');
```

```
xlabel('Frequency (Hz)');
```

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>	Nr - No.			
Magnus Nilsson & Michael Melin	5/036 31-1/FCK 115 07 Uen			
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

Appendix B: VHDL-code

On the following pages the VHDL-code is presented. The code contains:

- Testbench for the modulator
- Mainblock
- Logicblocks

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use std.textio.all;

entity tb_sigma_h is

end tb_sigma_h;

architecture textfiltest of tb_sigma_h is

component top_sigma port(
    clk,clk2,reset : in std_logic;
    sigma_in      : in std_logic_vector(15 downto 0);
    sigma_out     : out std_logic);
end component;

    signal tb_clk          : std_logic := '1';
    signal tb_clk2         : std_logic := '1';
    signal tb_reset       : std_logic := '1';
    signal tb_sigma_in    : std_logic_vector(15 downto 0);
    signal tb_sigma_out   : std_logic;

begin

    inst_top_sigma : top_sigma port map
        (clk => tb_clk, clk2 => tb_clk2, reset => tb_reset, sigma_in =>
tb_sigma_in, sigma_out => tb_sigma_out);

    tb_clk2 <= not (tb_clk2) after 10 ns;
    tb_clk <= not (tb_clk) after 20 ns;

    process

    file x_file: text is in "in1_9000.txt";
    file z_file: text is out"ut_kod.txt";

    procedure sim is

        variable l1 : line;
        variable log_val_1 : bit_vector(15 downto 0);
        variable good : boolean;

    begin
        if ((not (endfile(x_file)))) then

            good := false;

            while ((not good) and (not(endfile (x_file)))) loop

                readline(x_file, l1);
                read(l1, log_val_1, good);
            end loop;

            tb_sigma_in <= to_StdLogicVector(log_val_1); --omvandling
bit_vector till std_logic_vector

            end if;

        end sim;

    procedure utfil is

```

```
variable z1 : line;
variable good : boolean;
variable log_val_1 : bit;

begin

log_val_1 := to_bit(tb_sigma_out);

    write (z1, log_val_1);
    writeline(z_file, z1);

end utfil;

begin

    wait for 5 ns;
    tb_reset <= '0';

    for i in 1 to 9000 loop

        sim;
        utfil;

        wait for 40 ns;

    end loop;

    wait;

end process;

end textfiltest;
```

```

library ieee;
use ieee.std_logic_1164.all;

entity top_sigma is
    port (clk,clk2,reset: in std_logic;
          sigma_in      : in std_logic_vector(15 downto 0);
          sigma_out     : out std_logic);
end top_sigma;

architecture top_sigma_d of top_sigma is

component h_block_001 port(
    clk,clk2, reset, sign: in std_logic;
    sign_1      : in std_logic_vector(15 downto 0);
    h1_out     : out std_logic_vector(13 downto 0));

end component;

component h_block_002 port(
    clk, clk2,reset, sign: in std_logic;
    sign_2      : in std_logic_vector(13 downto 0);
    h2_out     : out std_logic_vector(13 downto 0));

end component;

component h_block_003 port(
    clk, clk2,reset, sign: in std_logic;
    sign_3      : in std_logic_vector(13 downto 0);
    h3_out     : out std_logic_vector(13 downto 0));

end component;

component h_block_004 port(
    clk, reset, sign: in std_logic;
    sign_4      : in std_logic_vector(13 downto 0);
    quant      : out std_logic);

end component;

----- internal signals in the sigma
block-----

        signal b1_b2      : std_logic_vector(13 downto 0); --
mellan b1 och b2
        signal b2_b3      : std_logic_vector(13 downto 0); --
mellan b2 och b3
        signal b3_b4      : std_logic_vector(13 downto 0); --
mellan b3 och b4
        signal sigma_out_d : std_logic; --
dummy signal mellan sign&sigma_out

-- port mappning

begin

    inst_h_block_001 : h_block_001
        port map (reset => reset, clk => clk, clk2 => clk2, sign_1 =>
sigma_in, h1_out => b1_b2, sign => sigma_out_d);

    inst_h_block_002 : h_block_002
        port map (reset => reset, clk => clk, clk2 => clk2, sign_2 =>

```

```
b1_b2, h2_out => b2_b3, sign => sigma_out_d);

    inst_h_block_003 : h_block_003
      port map (reset => reset, clk => clk, clk2 => clk2, signin_3 =>
b2_b3, h3_out => b3_b4, sign => sigma_out_d);

    inst_h_block_004 : h_block_004
      port map (reset => reset, clk => clk, signin_4 => b3_b4, quant
=> sigma_out_d, sign => sigma_out_d);

    sigma_out <= sigma_out_d;

end top_sigma_d;
```

```

--Kod för block 1, 18 in, 20 ut, truncat till 14 ut,
--riktig 16-bit insignal, sign in
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity h_block_001 is
    port (clk,clk2, reset, sign: in std_logic;
          sign_1      : in std_logic_vector(15 downto 0);
          h1_out      : out std_logic_vector(13 downto 0));
end h_block_001;

architecture rtl of h_block_001 is

    signal a      : std_logic_vector(17 downto 0);
    signal b_tc   : std_logic_vector(17 downto 0);
    signal g      : std_logic_vector(19 downto 0);
    signal h1_in  : std_logic_vector(17 downto 0);
    signal f      : std_logic_vector(19 downto 0);
    signal e_limit : std_logic_vector(19 downto 0);
    signal j      : std_logic_vector(13 downto 0);
    signal k      : std_logic_vector(13 downto 0);
    signal l      : std_logic_vector(14 downto 0);
    signal l_6se  : std_logic_vector(20 downto 0);
    signal m      : std_logic_vector(21 downto 0);
    signal c      : std_logic_vector(18 downto 0);
    signal d      : std_logic_vector(21 downto 0);
    signal h      : std_logic_vector(20 downto 0);
    signal e      : std_logic_vector(22 downto 0);
    signal c_2se  : std_logic_vector(22 downto 0);
    signal i      : std_logic_vector(18 downto 0);
    signal n      : std_logic_vector(11 downto 0);
    signal o      : std_logic_vector(17 downto 0);
    signal g_temp : std_logic_vector(19 downto 0);

begin

    -----
    logic-----

    logic: process(sign_1, c, i, b_tc, c_2se, g_temp, g, d, h, e,
e_limit)

        begin

            if sign = '0' then
                h1_in <= (("01000000000000000000") + ((sign_1(15) & sign_
1(15)) & sign_1));
            else
                h1_in <= (("11000000000000000000") + ((sign_1(15) & sign_
1(15)) & sign_1));
            end if;

            c <= ((i(17) & i(17 downto 0)) + (b_tc(17) & b_tc));

            if (c(18) = '0') then
                c_2se <= ("0000" & c);
            else
                c_2se <= ("1111" & c);
            end if;

            g_temp <= ((not g) +1);
            d <= (h(20) & h) + (g_temp(19) & g_temp(19) & g_temp);

```

```

e <= c_2se + (d(21) & d);

if (e(22) = '0' and e(21 downto 19) >= 1) then
    e_limit <= ('0', others => '1');
elsif (e(22) = '1' and e(21 downto 19) = 0) then
    e_limit <= ('1', others => '0');
else
    e_limit <= (e(22) & e(18 downto 0));
end if;

```

```
end process logic;
```

-----multiplication with a-----

```

mult_a : process (n, a, o)
begin
    n <= (('0' & a(17 downto 7)) + ('0' & (a(17) & a(17 downto
8)))));

    if n(11) = '1' then
        o(17 downto 12) <= "111111";
        o(11 downto 0) <= n;
    else
        o(17 downto 12) <= "000000";
        o(11 downto 0) <= n;
    end if;

    i <= (('0' & ((not o) + 1)) + ('0' & a));
end process mult_a;

```

-----multiplication with
2a-----

```

mult_2a : process (j, f, k, l, l_6se, m)
begin
    j(13 downto 0) <= f(19 downto 6);
    k(12 downto 0) <= f(19 downto 7);
    k(13) <= f(19);
    l <= (('0' & j) + ('0' & k));

    if l(14) = '1' then
        l_6se(20 downto 15) <= "111111";
        l_6se(14 downto 0) <= l;
    else
        l_6se(20 downto 15) <= "000000";
        l_6se(14 downto 0) <= l;
    end if;

    m <= (('0' & ((not l_6se) + 1)) + ('0' & (f & '0')));
    h <= m(20 downto 0);

end process mult_2a;

```



```

-----
utsignal-----
    ut: process(clk2,reset)
        begin
            if reset = '1' then
                h1_out    <=('0', others => '0');

            elsif clk2'event and clk2 = '1' then
                h1_out    <= e_limit(19 downto 6);

            end if;

        end process ut;

-----
registers-----

    registr: process(clk,reset)
        begin
            if reset = '1' then
                a          <=('0', others => '0');
                b_tc       <=('0', others => '0');
                g          <=('0', others => '0');
                f          <=('0', others => '0');

            elsif clk'event and clk = '1' then
                a          <= h1_in;
                b_tc       <= ((not a) + 1);
                f          <= e_limit;
                g          <= f;

            end if;

        end process registr;

end;

```

```

--Kod för block 2, 15 in, 17 ut, truncat till 14 ut,
--insignal från block 1, sign in
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity h_block_002 is
    port (clk, clk2, reset, sign: in std_logic;
          sign_2      : in std_logic_vector(13 downto 0);
          h2_out      : out std_logic_vector(13 downto 0));
end h_block_002;

architecture rtl of h_block_002 is

    signal a      : std_logic_vector(14 downto 0);
    signal b_tc   : std_logic_vector(14 downto 0);
    signal g      : std_logic_vector(16 downto 0);
    signal f      : std_logic_vector(16 downto 0);
    signal e_limit : std_logic_vector(16 downto 0);
    signal h2_in  : std_logic_vector(14 downto 0);
    signal j      : std_logic_vector(10 downto 0);
    signal k      : std_logic_vector(10 downto 0);
    signal l      : std_logic_vector(11 downto 0);
    signal l_6se  : std_logic_vector(17 downto 0);
    signal m      : std_logic_vector(18 downto 0);
    signal c      : std_logic_vector(15 downto 0);
    signal d      : std_logic_vector(18 downto 0);
    signal h      : std_logic_vector(17 downto 0);
    signal e      : std_logic_vector(19 downto 0);
    signal c_2se  : std_logic_vector(19 downto 0);
    signal i      : std_logic_vector(15 downto 0);
    signal n      : std_logic_vector(8 downto 0);
    signal o      : std_logic_vector(14 downto 0);
    signal g_temp : std_logic_vector(16 downto 0);

begin

    -----
    logic-----

    logic: process(sign_2, c, i, b_tc, c_2se, g_temp, g, d, h, e,
                  e_limit)

        begin

            if sign = '0' then
                h2_in <= (("010100011111111") + (sign_2(13) & sign_2));
            else
                h2_in <= (("101011100000001") + (sign_2(13) & sign_2));
            end if;

            c <= ((i(14) & i(14 downto 0)) + (b_tc(14) & b_tc));

            if (c(15) = '0') then
                c_2se <= ("0000" & c);
            else
                c_2se <= ("1111" & c);
            end if;

            g_temp <= ((not g) +1);
            d <= (h(17) & h) + (g_temp(16) & g_temp(16) & g_temp);
            e <= c_2se + (d(18) & d);

            if (e(19) = '0' and e(18 downto 16) >= 1) then

```

```

    e_limit    <= ('0', others => '1');
    elsif (e(19) = '1' and e(18 downto 16) = 0) then
        e_limit    <= ('1', others => '0');
    else
        e_limit    <= (e(19) & e(15 downto 0));
    end if;

```

```
end process logic;
```

-----multiplication with a-----

```

mult_a : process (n, a, o)
begin
    n <= (('0' & a(14 downto 7)) + ('0' & (a(14) & a(14 downto
8)))));

    if n(8) = '1' then
        o(14 downto 9) <= "111111";
        o(8 downto 0)  <= n;

    else
        o(14 downto 9) <= "000000";
        o(8 downto 0)  <= n;

    end if;

    i <= (('0' & ((not o) + 1)) + ('0' & a));

end process mult_a;

```

-----multiplication with
2a-----

```

mult_2a : process (j, f, k, l, l_6se, m)
begin
    j(10 downto 0) <= f(16 downto 6);
    k(9 downto 0)  <= f(16 downto 7);
    k(10)          <= f(16);
    l              <= (('0' & j) + ('0' & k));

    if l(11) = '1' then
        l_6se(17 downto 12) <= "111111";
        l_6se(11 downto 0)  <= l;

    else
        l_6se(17 downto 12) <= "000000";
        l_6se(11 downto 0)  <= l;

    end if;

    m <= (('0' & ((not l_6se) + 1)) + ('0' & (f & '0')));
    h <= m(17 downto 0);

end process mult_2a;

```

utsignal-----

```

utsignal: process(clk2,reset)
    begin
        if reset = '1' then
            h2_out      <=('0', others => '0');

        elsif clk2'event and clk2 = '1' then
            h2_out  <= e_limit(16 downto 3);

        end if;

    end process;

-----
registers-----

registr: process(clk,reset)
    begin
        if reset = '1' then
            a      <=('0', others => '0');
            b_tc   <=('0', others => '0');
            g      <=('0', others => '0');
            f      <=('0', others => '0');

        elsif clk'event and clk = '1' then
            a <= h2_in;
            b_tc <= ((not a) + 1);
            f <= e_limit;
            g <= f;

        end if;

    end process registr;

end;

```

```
--Kod för block 3, till h-block 3: 15 in, 15 ut, truncat till 14 ut,
--insignal från block 2, sign in
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
```

```
entity h_block_003 is
    port (clk, clk2, reset, sign: in std_logic;
          sign_3      : in std_logic_vector(13 downto 0);
          h3_out      : out std_logic_vector(13 downto 0));
end h_block_003;
```

```
architecture rtl of h_block_003 is
```

```
signal a      : std_logic_vector(14 downto 0);
signal b_tc   : std_logic_vector(14 downto 0);
signal g      : std_logic_vector(14 downto 0);
signal e_limit : std_logic_vector(14 downto 0);
signal f      : std_logic_vector(14 downto 0);
signal h3_in  : std_logic_vector(14 downto 0);
signal j      : std_logic_vector(8  downto 0);
signal k      : std_logic_vector(8  downto 0);
signal l      : std_logic_vector(9  downto 0);
signal l_6se  : std_logic_vector(15 downto 0);
signal m      : std_logic_vector(16 downto 0);
signal c      : std_logic_vector(15 downto 0);
signal d      : std_logic_vector(16 downto 0);
signal h      : std_logic_vector(15 downto 0);
signal e      : std_logic_vector(17 downto 0);
signal i      : std_logic_vector(15 downto 0);
signal n      : std_logic_vector(8  downto 0);
signal o      : std_logic_vector(14 downto 0);
signal g_temp : std_logic_vector(14 downto 0);
```

```
begin
```

```
-----
logi-----
```

```
    logic: process(sign_3, c, i, b_tc, g_temp, g, d, h, e, e_limit)
```

```
        begin
```

```
            if sign = '0' then
                h3_in <= ("0010111111110011") + (sign_3(13) & sign_3);
            else
                h3_in <= ("110100000001101") + (sign_3(13) & sign_3);
            end if;
```

```
            h <= m(15 downto 0);
            c <= ((i(14) & i(14 downto 0)) + (b_tc(14) & b_tc));
            g_temp <= ((not g) +1);
            d <= (h(15) & h) + (g_temp(14) & g_temp(14) & g_temp);
            e <= (c(15) & c(15) & c) + (d(16) & d);
```

```
            if (e(17) = '0' and e(16 downto 14) >= 1) then
                e_limit <= ('0', others => '1');
            elsif (e(17) = '1' and e(16 downto 14) = 0) then
                e_limit <= ('1', others => '0');
            else
                e_limit <= (e(17) & e(13 downto 0));
            end if;
```

```

end process logic;

-----multiplication with
-----

mult_a : process (n, a, o)
begin
    n <= (('0' & a(14 downto 7)) + ('0' & (a(14) & a(14 downto
8)))));

    if n(8) = '1' then
        o(14 downto 9) <= "111111";
        o(8 downto 0) <= n;

    else
        o(14 downto 9) <= "000000";
        o(8 downto 0) <= n;

    end if;

    i <= (('0' & ((not o) + 1)) + ('0' & a));

end process mult_a;

```

```

-----multiplication with
2a-----

mult_2a : process (j, f, k, l, l_6se, m)
begin
    j(8 downto 0) <= f(14 downto 6);
    k(7 downto 0) <= f(14 downto 7);
    k(8) <= f(14);
    l <= (('0' & j) + ('0' & k));

    if l(9) = '1' then
        l_6se(15 downto 10) <= "111111";
        l_6se(9 downto 0) <= l;

    else
        l_6se(15 downto 10) <= "000000";
        l_6se(9 downto 0) <= l;

    end if;

    m <= (('0' & ((not l_6se) + 1)) + ('0' & (f & '0')));

end process mult_2a;

```

```

-----
utsignal-----

utsignal: process(clk2,reset)
begin
    if reset = '1' then
        h3_out <=('0', others => '0');
    end if;
end process;

```

```

        elsif clk2'event and clk2 = '1' then
            h3_out <= e_limit(14 downto 1);

        end if;

    end process;

-----
registers-----

    registr: process(clk,reset)

        begin

            if reset = '1' then
                a <=('0', others => '0');
                b_tc <=('0', others => '0');
                g <=('0', others => '0');
                f <=('0', others => '0');

                elsif clk'event and clk = '1' then
                    a <= h3_in;
                    b_tc <= ((not a) + 1);
                    f <= e_limit;
                    g <= f;

                end if;

            end process registr;

        end;

```

```

--Kod för block 4, till h-block 4: 15 in, 15 ut, qvantat till 1,
--insignal från block 3, sign in
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity h_block_004 is
    port (clk,reset, sign: in std_logic;
          sign_4      : in std_logic_vector(13 downto 0);
          quant       : out std_logic);
end h_block_004;

architecture rtl of h_block_004 is

    signal a      : std_logic_vector(14 downto 0);
    signal b_tc   : std_logic_vector(14 downto 0);
    signal g      : std_logic_vector(14 downto 0);
    signal f      : std_logic_vector(14 downto 0);
    signal e_limit : std_logic_vector(14 downto 0);
    signal h4_in  : std_logic_vector(14 downto 0);
    signal j      : std_logic_vector(8  downto 0);
    signal k      : std_logic_vector(8  downto 0);
    signal l      : std_logic_vector(9  downto 0);
    signal l_6se  : std_logic_vector(15 downto 0);
    signal m      : std_logic_vector(16 downto 0);
    signal c      : std_logic_vector(15 downto 0);
    signal d      : std_logic_vector(16 downto 0);
    signal h      : std_logic_vector(15 downto 0);
    signal e      : std_logic_vector(17 downto 0);
    signal i      : std_logic_vector(15 downto 0);
    signal n      : std_logic_vector(8  downto 0);
    signal o      : std_logic_vector(14 downto 0);
    signal g_temp : std_logic_vector(14 downto 0);

begin
    -----
    logi-----

    logic: process(sign_4, c, i, b_tc, g_temp, g, d, h, e, e_limit)

        begin

            if sign = '0' then
                h4_in <= ("001111010111011") + (sign_4(13) & sign_4);
            else
                h4_in <= ("110000101000101") + (sign_4(13) & sign_4);
            end if;

            h <= m(15 downto 0);
            c <= ((i(14) & i(14 downto 0)) + (b_tc(14) & b_tc));
            g_temp <= ((not g) +1);
            d <= (h(15) & h) + (g_temp(14) & g_temp(14) & g_temp);
            e <= (c(15) & c(15) & c) + (d(16) & d);

            if (e(17) = '0' and e(16 downto 14) >= 1) then
                e_limit <= ('0', others => '1');
            elsif (e(17) = '1' and e(16 downto 14) = 0) then
                e_limit <= ('1', others => '0');
            else
                e_limit <= (e(17) & e(13 downto 0));
            end if;

        end process;

    end architecture;

```



```

        quant    <= not(e_limit(14));
    end process logic;
-----multiplication with
-----

    mult_a : process (n, a, o)
    begin
        n <= (('0' & a(14 downto 7)) + ('0' & (a(14) & a(14 downto
8)))));

        if n(8) = '1' then
            o(14 downto 9) <= "111111";
            o(8 downto 0)    <= n;

        else
            o(14 downto 9) <= "000000";
            o(8 downto 0) <= n;

        end if;

        i <= (('0' & ((not o) + 1)) + ('0' & a));
    end process mult_a;
-----multiplication with
2a-----

    mult_2a : process (j, f, k, l, l_6se, m)
    begin

        j(8 downto 0) <= f(14 downto 6);
        k(7 downto 0) <= f(14 downto 7);
        k(8)          <= f(14);
        l              <= (('0' & j) + ('0' & k));

        if l(9) = '1' then
            l_6se(15 downto 10) <= "111111";
            l_6se(9 downto 0)   <= l;

        else
            l_6se(15 downto 10) <= "000000";
            l_6se(9 downto 0)   <= l;

        end if;

        m <= (('0' & ((not l_6se) + 1)) + ('0' & (f & '0')));

    end process mult_2a;
-----
registers-----

    registr: process(clk,reset)
    begin

        if reset = '1' then
            a    <=('0', others => '0');
            b_tc <=('0', others => '0');

```

```
        g      <=('0', others => '0');
        f      <=('0', others => '0');

    elsif clk'event and clk = '1' then
        a      <= h4_in;
        b_tc   <= ((not a) + 1);
        f      <= e_limit;
        g      <= f;

    end if;

end process registr;
end;
```

Uppgjord (även faktaansvarig om annan) - Prepared (also subject responsible if other)		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - Doc respons/Approved	Kontr - Checked	Datum - Date	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

Appendix C: Circuit board

For the implementation of our design we used an existing construction (ROA 117 4240/1) that was modified to our needs.

The card was mounted according to the schematics, except for the DA and HOTLINK which were left unmounted.

The component V11 was not available, therefor an alternative design was mounted instead. The connection points on the card are in parenthesis. A shortcut is needed between A07 and B07 on P43, the resistors R113 and R114 also needs to be removed.

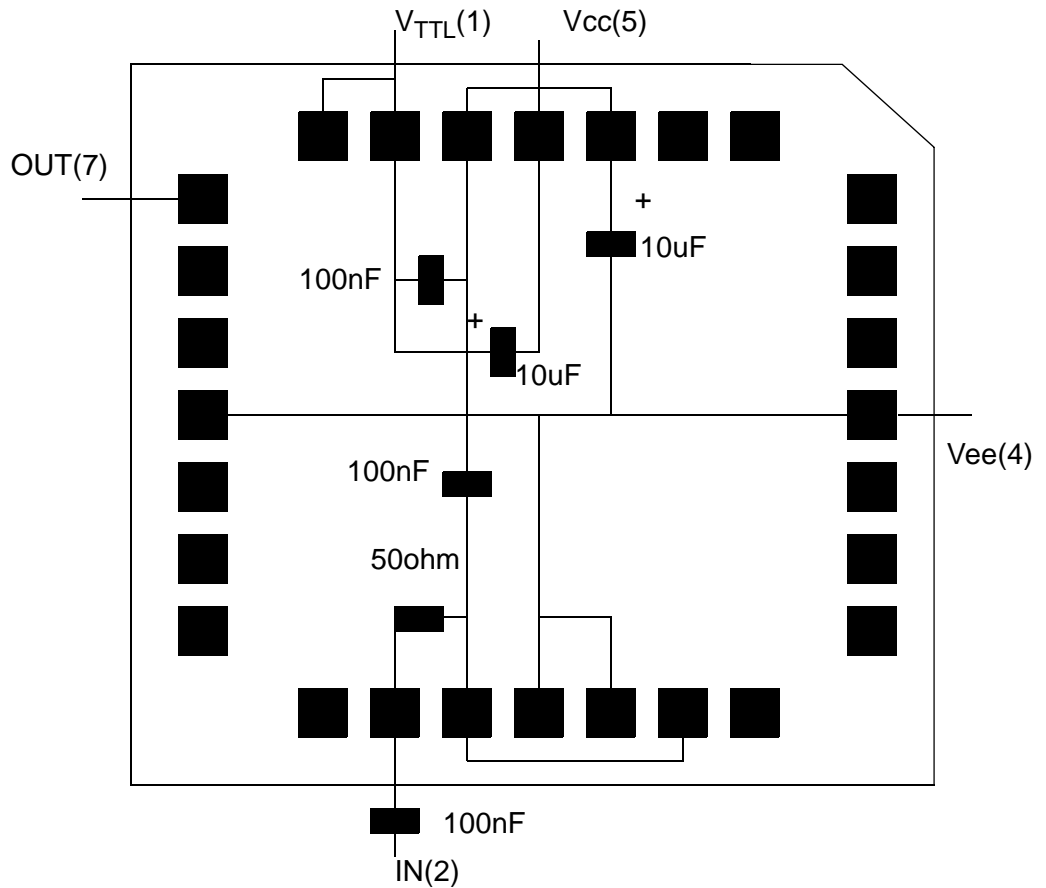
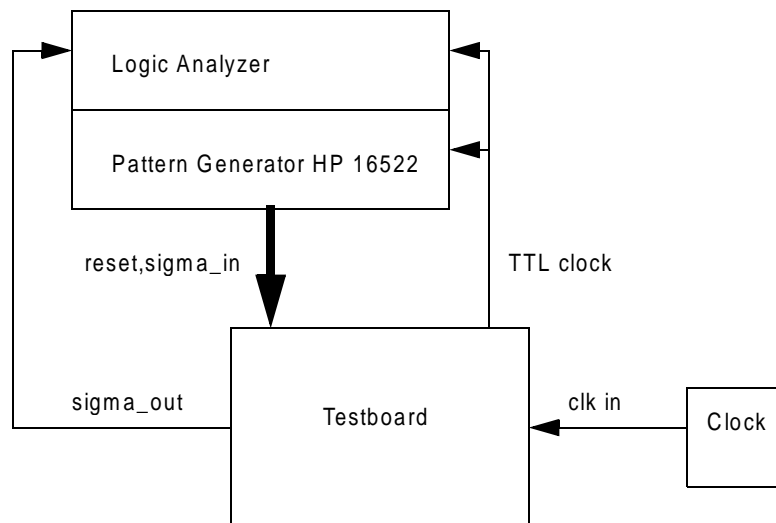


Fig.1. RYT 408 235/C Bottom view

Uppgjord (även faktaansvarig om annan) - <i>Prepared (also subject responsible if other)</i>		Nr - No.		
Magnus Nilsson & Michael Melin		5/036 31-1/FCK 115 07 Uen		
Dokansv/Godkänd - <i>Doc respons/Approved</i>	Kontr - <i>Checked</i>	Datum - <i>Date</i>	Rev	File
EMW FM/DC Håkan Enskog		1999-06-10	A	

Appendix D: Realization

At the realization and testing of the design, the input signal was taken from the pattern generator in HP 16522 and the output was recorded with the logic analyzer in the same instrument. Both the patterngenerator and the logic analyzer were clocked with the TTL clock on the testboard, connected at P23. As testsignal we used a sinus signal (loop of 40 samples). 16 ksamples were recorded and analyzed in Matlab.



The pinconnections on the FPGA and the testboard are located on the following page.