UNIVERSITY OF OSLO
Department of Informatics

Power efficient $\Delta\Sigma$ bitstream encoder/decoder

Master thesis

Daniel Mo

August 3, 2009

Abstract

The thesis presents design and implementation of components for recoding of signals between binary PCM and a bitstream representation. The bitstream representation enables real time signal processing using a power efficient algorithm for time-domain cross-correlation. The greatest savings are made when the system is restricted to a low Oversampling Ratio (OSR).

A fully digital Delta Sigma encoder is presented. The low OSR is the largest limiting factor in the system. Both simulations and measured results of the implemented modulator shows an achievable Signal to Noise Ratio (SNR) in the range $30 - 35$dB.

Digital filters are designed for sampling rate conversion between the Nyquist rate and the oversampled rate. Considerations and trade-offs specific to filter design in a Delta Sigma context are given. Several measures are taken to improve power efficiency of the circuits, while evaluating the impact on signal quality after conversion.

Both filters and the Delta Sigma encoder are implemented in 90nm CMOS on a single 1mm^2 chip, together with the cross-correlator. Circuit performance is evaluated for all circuits by theoretical considerations, simulations and chip measurements.

Contents

Acknowledgments

I would like to thank my supervisor Tor Sverre Lande for accepting me as his student and for motivation and guidance throughout these two years. Helpful discussions and valuable feedback have improved the results of the project.

Thanks to my co-supervisor Håkon A. Hjortland. His attention to LaTeXdetails have helped in making this text look even more beautiful than it otherwise would.

Thanks also to Olav, Kristian, Bård, Øyvind, Dag, Håvard and the others who spent time in the lab these years. Interesting technical discussions as well as lunch-time conversations have kept spirits up.

One final "thank you" to `#angry_angakoks` and `svada-core` for allowing me to maintain a refreshing social life during periods of making my home in the lab.

Acronyms

ADC Analog-to-Digital Converter

AD Analog/Digital

API Application Programming Interface

CIC Cascaded Integrator Comb

CMOS Complementary Metal Oxide Semiconductor

CSD Canonical Signed Digit

DAC Digital-to-Analog Converter

DAQ Data Acquisition

DA Digital/Analog

DSD Direct Stream Digital

DSM Delta Sigma Modulator

DSP Digital Signal Processing

ENOB Effective Number of Bits

FFT Fast Fourier Transform

FIR Finite Impulse Response

FOM Figure of Merit

GTL Gunning Transceiver Logic

IIR Infinite Impulse Response

ITRS International Technology Roadmap for Semiconductors

MASH Multi-stAge noise SHaping

MCU Micro-controller Unit

MUX Multiplexer

NTF Noise Transfer Function

OSR Oversampling Ratio

PCB Printed Circuit Board

PCM Pulse Code Modulation

PSD Power Density Spectrum

RF Radio Frequency

SACD Super Audio CD

SNR Signal to Noise Ratio

SPI Serial Peripheral Interface Bus

SQNR Signal to Quantization Noise Ratio

STF Signal Transfer Function

TQFP Thin Quad Flat Pack

TTL Transistor-Transistor Logic

USART Universal Synchronous/Asynchronous Receiver/Transmitter

USB Universal Serial Bus

WSN Wireless Sensor Network

1 Introduction

1.1 Ubiquitous computing

One of the major trends in modern microelectronics is the increase in development of devices specialized for its application. The evolution in electronics has moved from large and high-performance mainframe machines with multiple users, through personal computers where most people have access to at least one multi-purpose machine. Computing power continues to increase while silicon feature sizes continues to decrease. This enables smaller computers to be integrated into our surroundings, performing simple tasks that simplify our lives. The ideal for such autonomous devices is to blend in with the environment and work without requiring any awareness from a user. This idea has been characterized as a third wave of computing, or ubiquitous computing [35].

These devices have other demands than the traditional computing systems. Traditional applications requires a high performance while ubiquitous applications emphasizes very low power and size. If small autonomous devices are to exist in large numbers, each can not require frequent maintenance such as change of batteries. Size is also an important matter. If devices are to be a natural part of the users environment, its physical size can not be bulky or heavy. The cell phone is one example. It would not likely be part of our everyday life if it never shrunk beneath the size of a small suitcase. Even smaller devices are also becoming available. Computer chips worn on or inside the body would not be possible if not for a small size and very low power requirements. Such applications may not include a large battery or consume power in the same range as a personal computer. This leads to a change of emphasis in the electronic solution. New application fields are opened up and new difficulties have to be met.

Wireless Sensor Network (WSN)

One such field where there has been a significant research activity lately is in Wireless Sensor Networks (WSNs). A WSN consists of a number of small devices or nodes performing some sensor task. These are intended to work independent of each other and without outside management. Communication of data is done by radio transmission, but instead of communicating with a common centralized point, the nodes work as relays. This reduces the required transmitter range to the distance between nodes in the network. Power is saved by limiting the range of the transmitters, which allows smaller batteries or longer battery life of the devices. The application areas for small, radio operated devices are

numerous: From surveillance in industry to energy savings in buildings. A WSN has even been used to optimize ice skating conditions by embedding a network of temperature sensor nodes within the ice [2].

Sensing devices often perform very limited tasks. Real world processes are commonly slowly changing and does not always require high speed operation. Radio transmission on the other hand requires high frequencies, and is left as the most power hungry part of the unit. In other words, the largest power savings come from reducing the range of the transmitter, but also from reducing the amount of data being transmitted. One way of doing this is by performing basic data processing tasks before transmission, instead of on a central device.

Common to many such devices is that they require use of mixed-mode signals in interfacing with the surroundings. Sensing of an environmental process requires some form of sensor and an Analog-to-Digital Converter (ADC), while control of a process requires a Digital-to-Analog Converter (DAC). A highly successful and power efficient method for both AD and DA conversion is Delta Sigma modulation. Converters using this technique are widespread and dominant among high resolution converters for frequencies below a few MHz.

1.2 Power consumption

The International Technology Roadmap for Semiconductors (ITRS) identifies power consumption as one of the major challenges of the industry in both the short term and the long term. One of the reasons for this is that improvements of effective heat removal seems to level, while the density of transistors per chip area continues to increase in the foreseeable future [14]. In addition comes an ever increasing demand for portable and low-power devices operated by battery. Battery life puts a limit on the practical power consumption of portable consumer electronics. Reducing power consumption is important both for high performance systems and for lower performance portable applications.

Dynamic power

Two main factors make up the total power dissipation of an integrated circuit; supply voltage and total current through transistors. Total current is a function of the supply voltage V, switched capacitance and frequency of operation f. The switched capacitance is given as the fraction of active gates A times the total capacitance C in the system. This leads to an equation for dynamic power consumption in a digital circuit [16].

$$P_{dynamic} = ACfV^2$$

For a digital Complementary Metal Oxide Semiconductor (CMOS) circuit, supply voltage is usually limited by the processing technology and is typically the same in a given system.

Total capacitance in a digital circuit may be assumed proportional to the number of transistors N, as long as the circuit is built of mostly minimum sized transistors. This leads to a simplification of the above power relationship, suited for comparison of circuits within the same processing technology and supply voltage.

$$P_{dynamic} \propto N \times f_{clk}$$

Activity level of the transistors requires detailed knowledge of the circuit and may be hard to estimate in a complex system. Fortunately, this may be further simplified by assuming the activity level to be approximately proportional to the clock rate of a system. This holds for synchronous circuits where levels are changing only as a result of the clock signal.

Activity level is often signal-dependent. Consider for example the addition of two digital signals. It is clear that the addition of sequences of zeros requires a very low activity level in the adders, regardless of the clock rate, whereas addition of highly active signals requires a larger number of state changes in the adders. From this, it can also be argued that the approximation is well suited for noise-shaping systems, in that the activity level of the signal is always high, even for a static input. The activity thus has a closer dependence on the clock rate than on the input signal.

From the above equations it is clear that the high clock rate of an oversampled system contributes to the power dissipation, and should be kept as low as possible. A doubling of the clock rate will only allow for half as many transistors without increasing dynamic power dissipation.

Static power

As processes move towards a finer pitch, an increasing portion of the total currents are due to static leakage currents in the transistors. These currents come from two sources: The first is from sub-threshold currents through channels when transistors are in the off-state, or weak inversion. Threshold currents increase with a reduction in threshold voltage. The second source is gate oxide leakage; currents leak through the very thin gate insulation. This current is related to the thickness of the gate oxide layer. The layer is reduced proportionally with transistor minimum sizes to avoid short channel effects [16]. Both current sources are technology dependent, and proportional to the number of transistors and supply voltage. They are also independent of activity in the circuit. A modification of the above power equation must be made to take this into account.

$$P = P_{dynamic} + P_{static}$$

where

$$P_{dynamic} \propto N \times f_{clk}$$
$$P_{static} \propto N$$

For larger feature processes, static power dissipation is usually negligible in comparison with the dynamic power. With advances in manufacturing technology, static power is no longer negligible and makes up an ever increasing portion of the total power dissipated in a circuit. This is especially true where dynamic power is low due to a low clock rate or activity level.

This increases the effect of circuit complexity on total power dissipation. For modern processes, one may no longer add complexity without adding to the power dissipation. Traditional solutions that reduce power dissipation while increasing the number of transistors becomes less effective. This motivates a search for solutions on other levels, to lower the total power consumption of a system.

Power reduction

Several approaches to power reduction have been successfully explored. It is practical to consider such measures on several levels and it is possible to divide these into at least four levels [7]:

- Technology
- Circuit
- Architecture
- Algorithm

The Technology level includes reductions in power due to overall supply voltage reduction and transistor minimum sizes. The Circuit level includes low-level measures such as choice of topology, choice of logic family and gate sizing. Reductions on this level are closely connected to the transistors' analog properties. Architectural improvements include pipe-lining and parallelism to increase throughput and allow for a lower-voltage design. Implementation of a function using an effective algorithm is a good way of decreasing total power dissipation by decreasing the total number of required operations. A good design for low power consumption may include measures on several or all of these levels.

Many applications require real time computations, especially in the fields typical for ubiquitous computing. Once a circuit is able to achieve a real time computation, there is nothing further to be gained from increasing the speed anywhere in the circuit. This is a degree of freedom that instead may be used to reduce power consumption. A good example of this can be seen by comparing the carry select adder topology with the ripple carry topology. The carry select is a faster topology, but requires a larger amount of internal state changes than the simpler ripple carry topology. This results in a higher power dissipation [7]. For applications where optimizing for speed is not important, the ripple carry adder is the better one because of its lower activity and thus lower power consumption per operation.

2 System overview

The system presented in this thesis is a collaborative project between Olav E. Liseth and the author. A single chip is produced in a 90nm CMOS process implementing power efficient cross-correlation by recoding a digital signal as a bitstream. The cross-correlator is the work of Liseth and is presented in [20], while signal recoding and interfacing is done as part of this thesis.

2.1 Signal recoding

Signal recoding is necessary in two steps: An incoming digital binary coded signal is recoded into its bitstream representation before processing by the cross correlator. The output of the correlator needs conversion back to the original form, either for simple readout using a computer or further processing steps. Figure 2.1 shows how the encoding and decoding is done.

The system is divided into four main parts, and these can be considered somewhat independent of each other.

Digital-to-digital DSM Converts oversampled Pulse Code Modulation (PCM) to a one-bit Delta Sigma bitstream. Chapter 3.

Interpolation filter Increases the sampling rate of the incoming Nyquist-rate signal. Chapter 4.

Decimation filter Converts bitstream back to Nyquist rate PCM. Chapter 5.

Signal processing Performs an efficient calculation on the bitstream. Discussed briefly in this chapter.

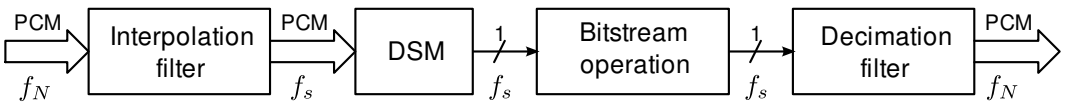


Figure 2.1: Intermediate bitstream representation allows for efficient signal processing.

Recoding enables use of signal processing operations on the bitstream representation. Some operations may be implemented more efficiently for bitstream code than binary code. Although the recoding of a digital input requires additional hardware and possibly losses to signal quality, this may be justified if it enables larger savings in the signal processing step. The PCM representation is both easier to interpret for humans and better suited for presentation or further processing on a computer than the raw output from the bitstream signal processor.

As Delta Sigma modulation is a technique originating from Analog/Digital (AD) and Digital/Analog (DA) conversion, high quality converters are available for a variety of uses. Hence the idea of signal processing directly on the bitstream representation is even more attractive for real world applications involving AD/DA conversion. Bitstream signal processing directly on the output of a single bit ADC removes the need for filtering prior to the processing step and reduces the overall hardware needs. Similarly for DA conversion; converting the single-bit (or multi-bit) processed signal directly to analog removes the need for the interpolating filter used as part of Nyquist rate to analog Delta Sigma Modulator (DSM) DACs today.

The cross-correlator is one example of a system utilizing signal processing directly in the oversampled domain. Another example of using Delta Sigma encoding as a means of enhancing a system is found in ultrasound research. A beam-former used for ultrasound imaging depends on adjustable delays with a good time-resolution. Such delays are simple to implement in an oversampled system where time steps are inherently small [26].

2.2 Bitstream cross-correlation

The implemented signal processing block is based on power efficient cross correlation as presented by Lande *et al.* [18]. Cross correlation is a useful operation in signal processing, but computationally very expensive. The function for cross correlation over a window of length N is

$$y[n] = f[n] \star g[n] = \sum_{k=0}^{N} f[k]g[k+n]$$

The five-pointed star $\star$ is the discrete cross-correlation operator. Each output sample of the cross-correlation consists of N multiplications between input samples accumulated as a sum. This requires either a large number of hardware-expensive multipliers, or few multipliers operating at an increased rate.

Savings in power are done by operating on bitstreams rather than on binary coded PCM signals. By recoding both operands to their bitstream representation, computations are simplified. Multiplication of single bit signals is done using simple XOR logic gates instead of multiplier circuits. The lower area cost enables operation in parallel. Summation of the multiplied products is done asynchronously in a sorting register using the bubble sort

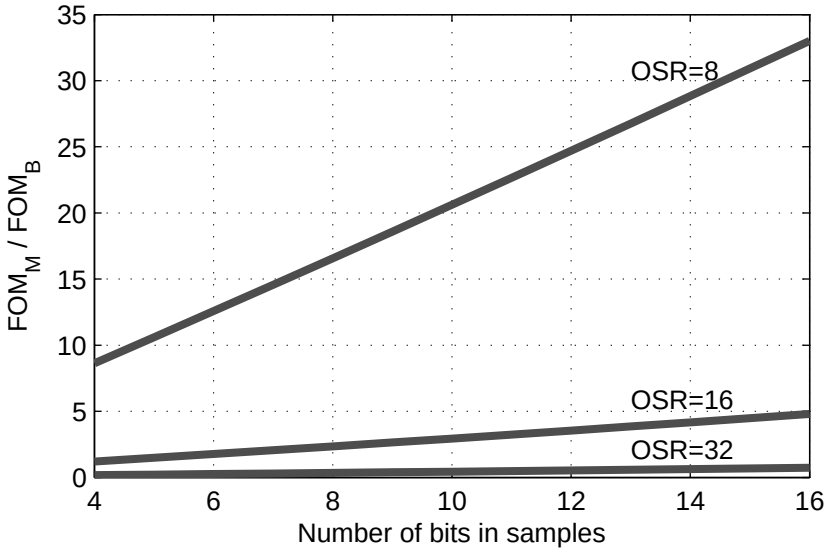


Figure 2.2: Estimated relative improvements from bitstream cross-correlation [18].

algorithm. Power savings of this method are estimated based on a comparison between the bitstream cross-correlator and a traditional multiplexed multiply-and-accumulate implementation. A Figure of Merit (FOM) is calculated for each approach using a power estimate similar to the one presented in section 1.2.

$$\text{FOM} = \text{Transistor count} \times \text{Clock rate}$$

For the multiplier based architecture, the FOM_M is a function of correlation length and input word size. For the bitstream implementation, OSR and correlation length is the most important parameters in determining the FOM_B. An estimate of the power savings based on transistor counts from recent articles is given in [18]. Results are reproduced in figure 2.2 and modified to for the maximum practical cross-correlation length of 1024 bits as found and implemented on the chip by Liseth [20]. The effective correlation window length is $N = 1024/OSR$ and estimated power savings may be shown to grow with an increase in this correlation window length.

The above results are crucial for the choice of sampling rates used in the system. Signal quality of Delta Sigma signal modulation is known to be highly dependent on the OSR of the input signal. This is in conflict with the goal of reducing overall power dissipation by the cross-correlation operation. To allow for the largest potential savings, OSR is kept at the lowest practical value without sacrificing too much dynamic range in the modulated signal. This is shown in chapter 3 to be $OSR = 8$.

2.2.1 Module interface

The bitstream cross correlator takes two Delta Sigma modulated single bit signals as input. One of the signals is loaded into a template register at start-up and does not change during normal operation. The other signal is input by shifting through a register synchronously with the system clock. Output from the unit is a multi-bit sum of all equal valued bits in the input signals.

To ensure sufficient signal resolution in the system, the internal data bus width is decided as a result of the maximum possible window size of 1024 bits for the cross-correlation unit. This results in a range of $1024 = 2^{10}$ possible output values after summation of multiplied bits, hence all PCM word sizes in the system is set to 10 bits. This is also sufficient resolution to properly evaluate the filtering modules.

The exact characteristics of the output signal was not investigated until after chip production. For this reason, the decimation filter following the cross-correlation module is designed for the general case as shown in figure 2.1, where the output after processing is assumed to be a single bit stream with a frequency domain characteristic similar to that of the input signal.

2.2.2 Convolution

The operation of convolution is closely related to cross-correlation. For discrete time real signals, the operations are interchangeable by time-reversing one of the signals. Hence the cross-correlation circuit may be used for convolution by reversing the stored template used for calculation. This implies that the circuit may also be used for effective implementation of programmable FIR filters by loading the template register with the bitstream representation of the filter impulse response.

However, typical FIR filter impulse responses have a composition not well suited for Delta Sigma modulation. High order filter functions commonly requires a large dynamic range to represent the impulse response. This may be seen by considering the low-pass filter design approach described in section 4.1. Applying a window function to a sinc shaped function necessarily results in a large center value and very small variations away from the center. Modulation of such a signal would require a very high OSR to preserve the dynamic range. Although this is difficult to achieve for high speed circuits, it may have value in real-world applications where signal frequencies are very low.

2.3 System evaluation

To evaluate the system and effects of trade-offs made, two criteria are put down for the system. The signal processing block is the overall largest part, and has its own criteria, namely performing precise cross-correlation while using as little power as possible. The

data conversion part of the system is a smaller part of the total, and the overall power criteria is reflected in the restriction to an OSR of 8 as discussed in 2.2. The conversion blocks are to provide a recoding of the signal while maintaining signal quality and without adding too much to the overall power dissipation.

Bandwidth, or maximum speed of operation is considered unimportant for this system as intended usage is for low and intermediate frequencies. The target operating range is frequencies around common digital audio rates and lower.

Power estimate

Power is abstracted by the estimation presented in chapter 1.2, using frequency times transistor count. A high-level approach is necessary when designing filters and Delta Sigma systems because of the complexity of resulting circuits. Hence it is useful to use a simple estimate because more precise calculations or simulations on transistor level quickly becomes tedious for a large system.

SNR

Signal quality is quantitatively evaluated by its dynamic range, or SNR, of the signal after filtering and recoding. The primary goal of the filters and DSM is to maintain a high SNR. SNR is used throughout the analysis of the various filter blocks and the DSM, and is measured as the ratio of signal power to the highest in-band noise power. This will be further discussed in later chapters.

Figure of Merit (FOM)

In combining these measures, we arrive at a FOM for the system:

$$\text{FOM} = \frac{\text{SNR}}{\text{Transistor count} \times f_{clk}} \tag{2.1}$$

This may be used in comparison between blocks performing the same function. For instance, a comparison may be done between multi-rate filters only if their purpose is the same. Using the above FOM for the DSM is not comparable with that of the decimation filter, as their functions are different. Still, it is useful in summarizing the effect of trade-offs done. It is left as a goal for design, rather than as an effective means of evaluation.

3 Delta-Sigma modulation

With advances in manufacturing technologies comes a change in the way circuits are effectively designed. Smaller feature sizes allows more transistors on a single chip, but requires a lowering of the supply voltages to keep power consumption down. Supply voltages of 1.0V and below greatly reduces the range where a transistor is in its strong inversion region. For this reason, traditional methods for analog design conflicts with the benefits of using a small feature process. Digital design methods on the other hand are able to fully profit from smaller technologies.

Delta sigma converters have gained in popularity as they provide a solution to the problems associated with analog design on state-of-the-art processes. They are realizable in low-cost CMOS processes using a minimum of sensitive analog circuitry. Converters using this technique typically produce a high resolution over a moderate signal bandwidth.

Delta Sigma modulation has been around since the 1960s and has grown to a large field of research. Several good books are available on the subject of using DSMs for AD and DA conversion [31, 27].

An overview of the principles is given in the first two sections of this chapter. Emphasis is on modulation in a fully digital system using a low OSR. Section 3.3 discusses design and trade-offs for the digital-to-digital delta-sigma encoder that is implemented on chip. Simulations of the DSM are presented in 3.4, showing about 5 bits of precision. The chapter finishes with a brief discussion of arithmetic operations on bitstreams in section 3.5.

3.1 Digital Encoding

Common to all digital coding is the discretization of a signal, both in time and value. Both sampling in time and quantization in value is done in A/D converters. Although A/D conversion is outside the scope of this thesis, quantization of signal amplitude will be looked into, as it is important in understanding differences between several coding forms.

A sampled and quantized signal may be represented digitally in several ways, the most straight-forward being PCM. This refers to the common way of representing a digital signal as a sequence of binary coded amplitude values.

Delta sigma modulation refers to a quite different approach to the coding of a signal.

3.1.1 Coding forms

Some coding forms are especially well suited for certain applications. As an example, consider the the μ-law and A-law companding schemes used in most telephone systems today. Logarithmic coding is effective for representing a speech signal. The human ear is more sensitive to changes when the sound signal is small than in a signal with a large amplitude. Hence, the resolution of such signals should be highest where the amplitude is low. The A/μ-law companding uses quantization steps which are not equally spaced, but proportional to the signal amplitude. This reduces the necessary bit-width and storage requirements without largely affecting the perceived quality of the signal. In other words, the coding utilizes knowledge about the signal's environment to represent it in an efficient manner. The same coding scheme is not suitable for any generic signal, but the recoding makes it more suitable for specific cases.

Delta Sigma coding is an other way to represent digital signals. Although mainly used as an intermediate code in data conversion, delta sigma has seen some use as the main coding scheme in a commercial system. Super Audio CD (SACD) uses a specification called Direct Stream Digital (DSD). This standard utilizes Delta Sigma modulation with a high oversampling ratio to produce a bitstream. The bitstream is then compressed and stored without conversion to PCM, as in done in ordinary music Cd's[15]. The motivation for using this coding seems purely to be an increase in sound quality by avoiding conversion errors.

3.1.2 Quantization

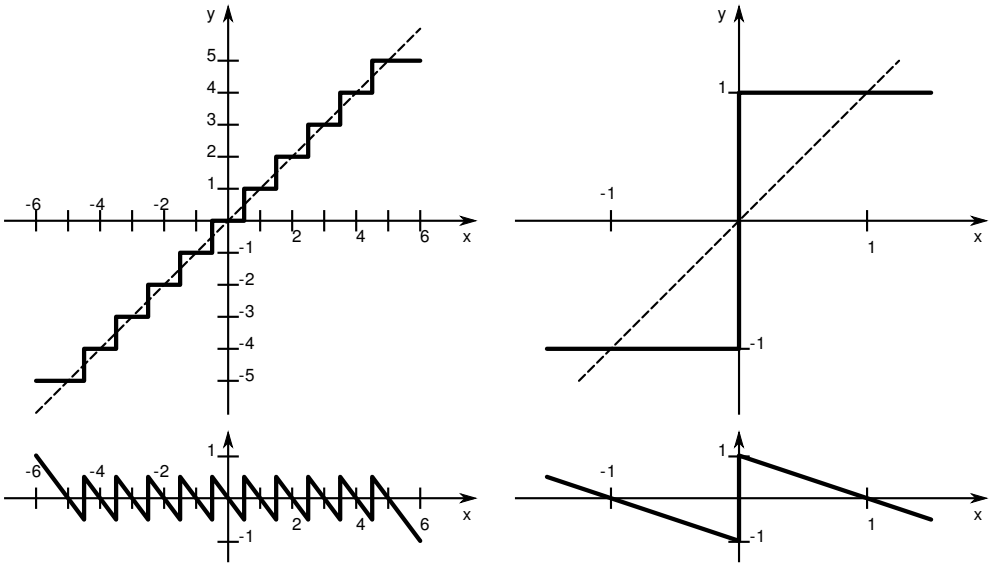
Because of the close relation to A/D conversion, it is useful to consider quantization in terms of discretizing a continuous input signal. The same considerations applies when recoding a digital signal into fewer or differently spaced levels.

Quantization is simply the process of approximating a continuous amplitude using a sequence of two or more ordered steps represented by binary values. Refer to figure 3.1 for two different examples of how a continuous signal may be quantized. The quantizer in figure 3.1b is very simple to implement, either as an analog comparator or digitally as an extraction of the most significant bit or the sign bit.

The quantization error e_{rms} of any quantizer, assuming an uncorrelated white noise model of the error, is found to be

$$e_{rms}^2 = \frac{\Delta^2}{12}$$

where Δ is the step size of the quantizer.



(a) 11 step mid-tread quantizer with $\Delta = 1$ (b) Two step mid-rise quantizer with $\Delta = 2$

Figure 3.1: Input-output characteristics of two example quantizers. Quantization error is shown below.

3.1.3 Oversampling

The Nyquist-Shannon sampling theorem states that a signal which is band-limited by f_0 may be fully represented by a sequence of equally spaced values, no further apart than $1/2f_0$. For a good representation, the data points must be very precise in terms of magnitude quantization. This is difficult to achieve in a converter, especially for low supply voltages, due to very high demands of the analog components.

Oversampling refers to increasing the number of samples beyond the requirements of the Nyquist theorem. The OSR is the ratio between the required Nyquist rate and the oversampling frequency. This may be summed up as:

$$f_s = OSR \times f_N = OSR \times 2f_0$$

The effect of oversampling on the quantization error is best viewed from the frequency domain. The traditional assumption is that quantization error is truly uncorrelated with the input signal and has a uniform random distribution. This is a simplification, but is valid for most cases where the input is highly active. This leads to a well-known formula for the amount of quantization error in a limited signal band. Given the spectral density of sampled white noise

$$E(f) = e_{rms}\sqrt{\frac{2}{f_s}}$$

Integrating for total quantization noise power within band of interest

$$q_{rms}^2 = \int_0^{f_0} E^2(f)\,\mathrm{d}f = e_{rms}^2\frac{2}{f_s}\int_0^{f_0} 1\,\mathrm{d}f = e_{rms}^2\frac{2f_0}{f_s}$$

Inserting for OSR gives the relation between in-band noise power q_{rms}^2 and OSR:

$$q_{rms}^2 = \frac{e_{rms}^2}{OSR}$$

This result shows that each doubling of the Oversampling Ratio (OSR) increases the resolution by a factor of $\sqrt{2}$, or 3dB.

Oversampling may be used as a simple method for increasing the resolution of any ADC. Several common micro-controllers contain a moderate resolution ADC. This provides a cheap method of increasing the resolution in applications where speed requirements are low. It is not very effective; an OSR of 4096 is required to increase ADC resolution by 6 bits. [4]

3.1.4 Noise shaping

Although not very effective in itself, oversampling is necessary for the concept known as noise shaping. Put simply, this is the filtering of quantization noise from a flat spectrum into

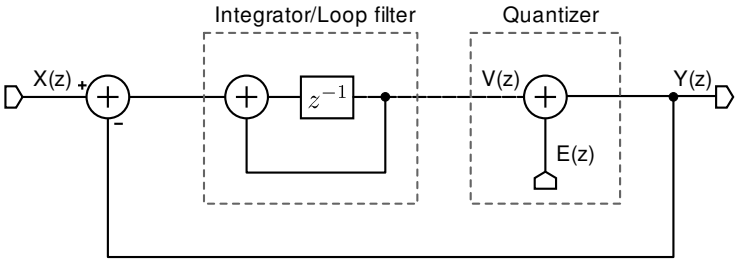
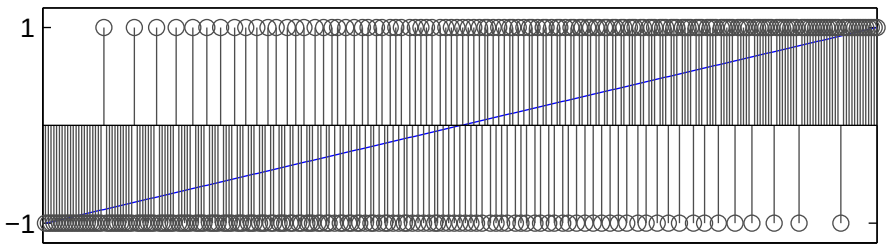
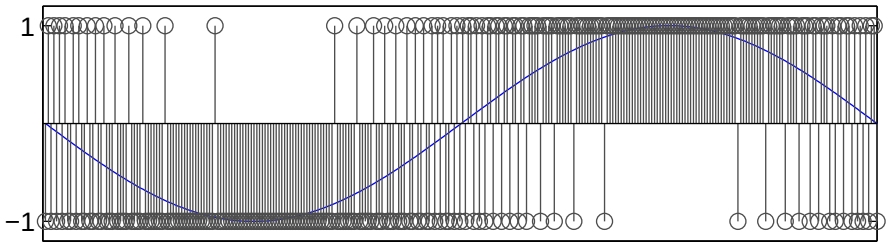


Figure 3.2: First order Delta Sigma modulator model. Quantizer replaced by injected error signal.



(a) Ramp signal



(b) Sinusoid signal

Figure 3.3: Operation of a 1. order DSM in the time domain. $OSR = 300$.

a spectrum with more high-frequency content and less low-frequency content. Noise power is moved outside the signal band, while leaving the signal power unaffected, thus increasing the SNR. This is done by applying different filtering functions to the input signal and the quantization error signal. Figure 3.2 shows the linear equivalent of a first-order DSM. The quantizer is replaced by the addition of an error signal, assumed to be independent of the input. This model may be used to determine the two transfer functions for the signal and the noise. From the figure

$$\begin{aligned} V(z) &= z^{-1}\big(V(z) + X(z) - Y(z)\big) \\ &= \frac{z^{-1}}{1 - z^{-1}}\big(X(z) - Y(z)\big) \end{aligned}$$

The output is given by

$$\begin{aligned} Y(z) &= E(z) + V(z) \\ &= E(z) + \frac{z^{-1}}{1 - z^{-1}}\big(X(z) - Y(z)\big) \\ (1 - z^{-1})Y(z) &= (1 - z^{-1})E(z) + z^{-1}X(z) - z^{-1}Y(z) \\ Y(z) &= (1 - z^{-1})E(z) + z^{-1}X(z) \end{aligned}$$

This gives a Noise Transfer Function (NTF) with a simple filter shape and a Signal Transfer Function (STF) which is only a single delay on the input. DSMs may also be made delay-free by moving the integrator delay step and adding an additional delay in the feedback loop.

$$\begin{aligned} NTF(z) &= 1 - z^{-1} \\ STF(z) &= z^{-1} \end{aligned}$$

This noise transfer function response is shown in figure 3.4 together with that of a second order modulator. The time domain output from a two-step first order DSM is shown in figure 3.3. Using a high OSR for illustration, it is possible to see how the average value of the 1-bit output approximates that of the input. However, a frequency domain analysis is necessary to get a good measure of the quality of the modulated signal. This is further detailed in 3.2

3.1.5 Higher order modulators

The noise shaping of the first order modulator is not effective enough for high quality signal encoding. The filter functions NTF and STF of the modulator may be extended for more efficient noise shaping. By simply adding more integrator steps, the order and

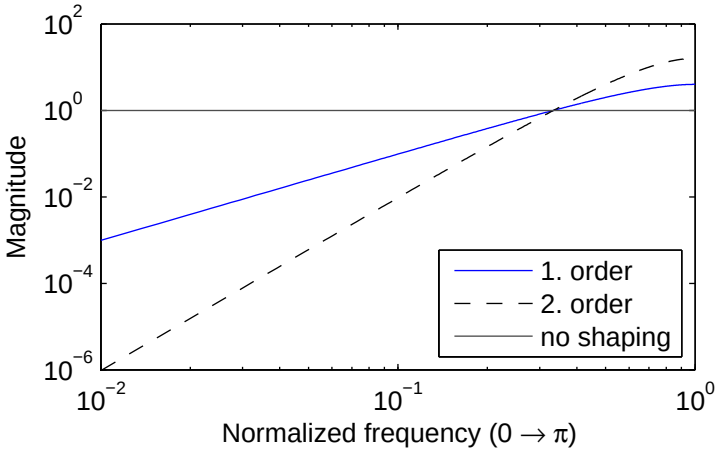
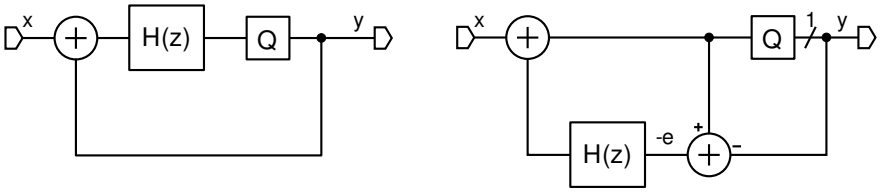


Figure 3.4: Noise transfer function response for 1st and 2nd order modulators. $NTF = 1$ is also shown for comparison.



(a) Single feedback structure

(b) Error feedback structure

Figure 3.5: General modulator structures showing loop filter placement.

noise-shaping effect of the modulator is increased. This introduces instability problems, and requires careful design methods for the modulator to be usable. When designing modulators of a higher order, it is useful to consider the general model using a loop filter as shown in figure 3.5a. The filter may then be analyzed and implemented as any ordinary discrete-time filter.

Most research and and books in the field of delta sigma modulation are directed towards higher order modulators, as these are effective in high quality data conversion [27, 31]. These publications tend to treat DSMs in the light of conversion between analog and digital. In the field of Radio Frequency (RF) systems, fully digital DSMs are used in the conversion between PCM coded signals and frequency or phase domain signals [6].

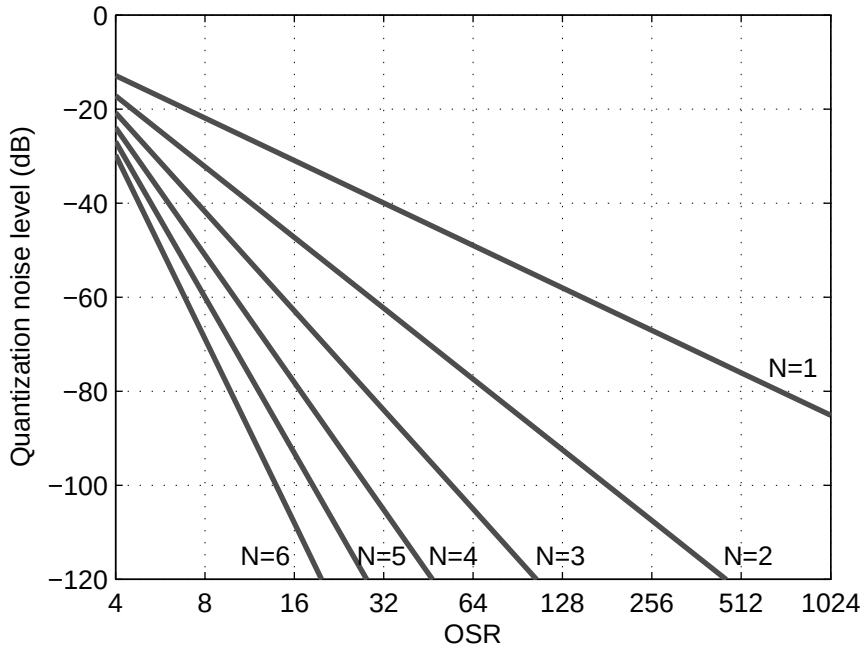


Figure 3.6: Theoretical in-band noise power for modulators of orders 1 through 6

3.2 Coding Efficiency

For a code to be useful it must provide benefits to outweigh its disadvantages, at least for specific applications. The main trade-off in using a Delta Sigma coding is between signal quality and the increased clock rate. This may lead to an unacceptable power consumption, if the oversampling ratio is high and if there are no significant benefits in using the oversampled coding. The increased clock rate may also impose limits on the maximum signal bandwidth, especially when the oversampling ratio is large.

For this reason, delta-sigma coding is normally only used as an intermediate code in the converter steps between analog and digital, while PCM is commonly being used for digital data processing. Depending on the application, it may be possible to exploit the oversampled coding in other ways, such as improvements of an algorithm or reduction of circuit complexity.

Ideal Noise Transfer Functions (NTFs)

From the low order modulators described in section 3.1, it is clear that the noise shaping ability of a modulator decides the quality of the resulting signal. The dependency of signal quality on modulator structure is interesting and thoroughly covered in several texts [27,

31]. In theory, the Signal to Quantization Noise Ratio (SQNR) may be generalized for modulators consisting of simple feedback loops with all coefficients equal to unity. These modulators have simple noise transfer functions (NTFs) of the form:

$$NTF = (1 - z^{-1})^N$$

where N is the number of loops and determines the order of the modulator. The quantization noise power within the signal band may be shown to approximately follow

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

and a summary of the most interesting range of values is shown in figure 3.6 on the preceding page. The figure shows how each doubling of the OSR increases the number of bits of resolution by $N + 0.5$ [31].

Practical Delta Sigma Modulators (DSMs)

It is important to note that these values unfortunately seem impossible to achieve in practice, mainly due to the instability problems of any single loop modulator of order higher than one. For this reason, it is necessary to modify the NTF of a high order modulators, reducing its noise-shaping ability. This will inevitably cause achievable signal quality to be much lower, especially for low oversampling ratios combined with higher order modulators. Several techniques have been developed to counteract this decrease of quality. Most important are multi-bit quantizers and cascaded modulator architectures, both of which produce multi-bit word streams. They will thereby lose several of the benefits of true single bit coding. Another successful technique is improving the NTF of the modulator by spreading its zeros across the signal band, thus further reducing quantization noise levels near the upper frequency of the signal band.

An empirical study shows that even with optimally placed zeros, the practical SNR for modulators of orders 5 through 8 lie below 40 dB for an OSR of 8, and below 70 dB for an OSR of 16. The results are taken from [31, p.112] and summarized in figure 3.7.

3.2.1 Signal quality

A direct comparison of PCM and Delta Sigma coding efficiency is somewhat difficult, as a single PCM sample does not correspond directly to a number of Delta Sigma bits. DSMs have some internal memory in the loop filter and the feedback makes each output value depend on several of the previous values. Additionally modulators do not have a finite impulse response, meaning that any input value will continue to affect the output for an infinite duration. Even a zero signal input will produce a highly active output signal after modulation. However, for long signals, consisting of several PCM samples, it is possible

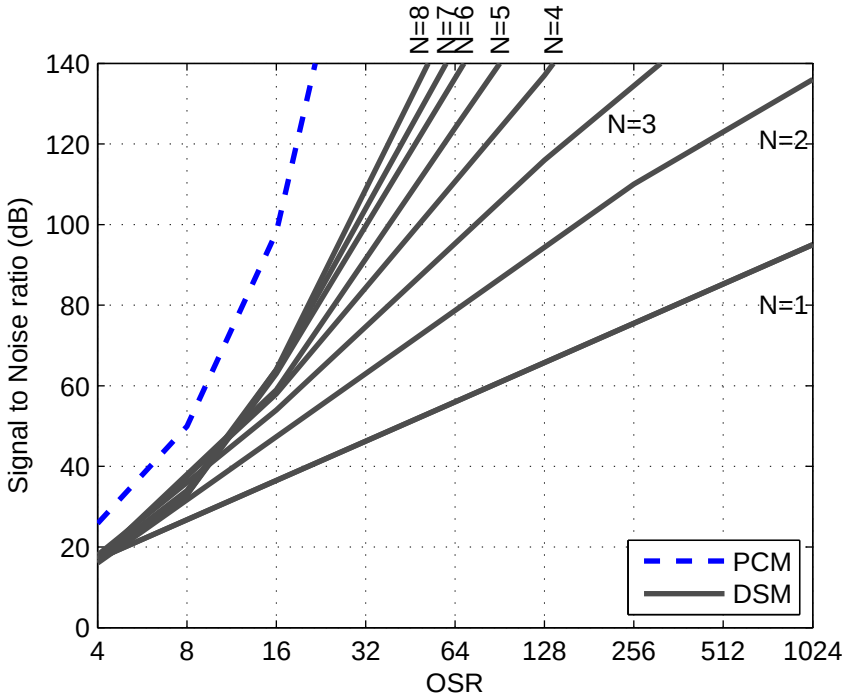


Figure 3.7: Empirical SNR for single-bit modulators of orders 1 through 8. NTFs with optimally placed zeros. Also shows SQNR for PCM code using same number of bits $N = OSR$. *Empirical data from [31, p.112]*

to compare the two by considering their SNRs. The SNR of a PCM coded signal may be calculated as the signal to quantization noise ratio:

$$\frac{S_{max}}{N_Q} = 20\log_{10} 2^N + 20\log_{10}\sqrt{\frac{3}{2}}$$

In the case of Delta Sigma, the efficiency of the coding is largely dependent on the modulator used, and it is not possible to calculate an SNR without exact knowledge of the modulator's transfer function. As mentioned earlier, the signal quality is dependent on the noise-shaping ability of the modulator. An estimate may be found from the in-band quantization noise level (see Fig.3.6), but the white noise model used does not account for nonlinear effects such as idle tones and harmonics in low order modulators. Although there are methods for theoretically calculating modulation quality, these are quite complicated[31, 3].

A more common method is to measure the SNR from the frequency spectrum of a modulated signal. This is done using a maximum amplitude sinusoid test signal. The SNR

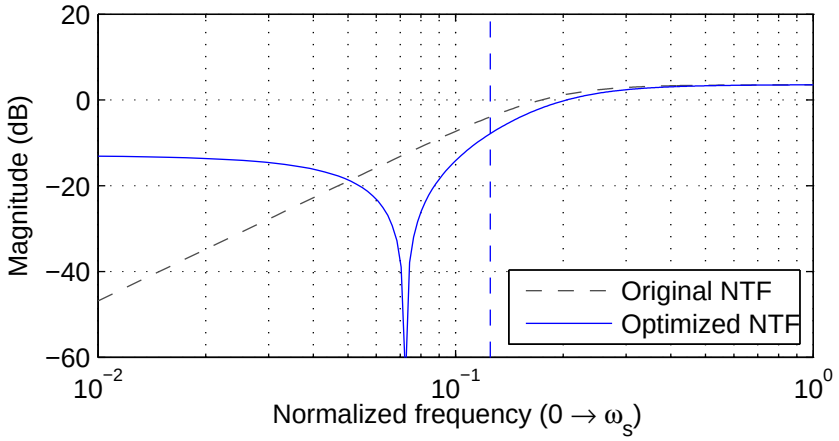


Figure 3.8: Effect of optimized zeros in a second order NTF, assuming $OSR = 8$. Nyquist frequency is indicated by dashed line.

is measured from the peak at the signal frequency to the highest non-signal component within the signal band. The result is the highest achievable SNR for the given test signal frequency. Other frequencies may cause tonal behavior within the passband, hence reducing the SNR. Obviously it is not possible to measure the SNR using test signals of every frequency and the measurement process turns into a somewhat inexact science. On the other hand, real-life signals tend to be more complex than simple sinusoids, and for these less regular signals the method has proven empirically to be a sufficiently good estimate of the SNR.

For one-bit modulators, Delta Sigma coding would be more efficient in terms of storage than PCM coding, if a given SNR may be achieved with a lower value for OSR than the PCM word-size. A comparison between the SQNR of PCM and simulated single bit modulators of a wide range of orders and OSRs is given in figure 3.7 on the facing page. From the figure, it is clear that PCM coding provides a higher information content per bit than any of the studied modulators. Even if delta-sigma coding does require more bits to represent a given signal than PCM does, it has other properties to make it attractive as an coding form.

3.2.2 Optimized NTF

The transfer function for a simple DSM does not give the optimal noise shaping effect for a given OSR. For a given OSR, it is possible to find a modified transfer function that has a somewhat lower NTF response at the corner frequency. From a z-domain point of view, this is done by spreading the zeros of the NTF away from the real axis. In the frequency

domain, this translates to moving the lowest frequency response value away from zero and towards the Nyquist frequency $f_N/2$. The effect is shown in figure 3.8.

Noise power is higher for lower frequencies, but is spread out over the signal band, resulting in slight improvements in overall SNR. Expected SNR improvement may be shown theoretically to be 3.5dB [31].

3.2.3 True serial representation

One of the main advantages of Delta Sigma code is simplicity, as a true one-bit serial representation of a signal. This is in contradiction to a multi-bit PCM signal. In its simplest implementation, the bits of a PCM signal are stored in parallel and processed in parallel. The signal may also be transmitted along a single electrical wire, by rearranging bits in time such as is done in any parallel to serial interface. Although only a single bit is processed at a time, the organization of bits into words must still be kept track of. PCM bits may always be sorted by significance, from most significant bit to least significant bit.

3.2.4 Robust coding

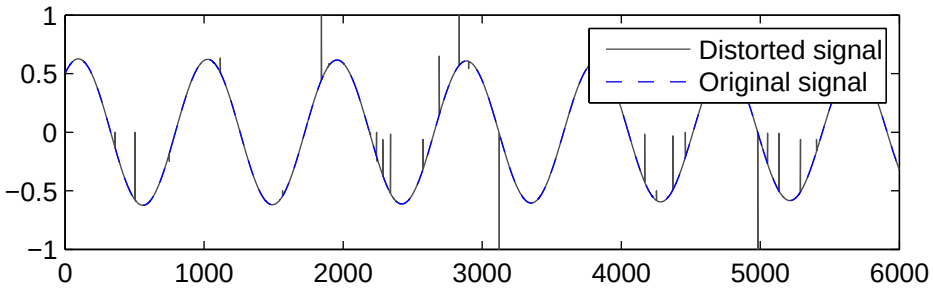
A Delta Sigma signal has no inherent ordering of the bits. The signal value is represented using several bits serialized in time, where each bit has the same weight on the signal value. This property is one that makes Delta Sigma a fault tolerant coding, since a few bit-errors will have less impact on the signal value than in PCM coding. Figure 3.9 shows the results of a simple test of how a small amount of bit-errors impact on signals in PCM and Delta Sigma coding. The Delta Sigma signal is oversampled by 64, modulated with a second order modulator, and reconstructed using a high order FIR-filter. Bit errors are introduced by flipping bits at random, with the same probability per bit for both the PCM and Delta Sigma signal. Delta Sigma coding has been shown to be a good way of implementing fault tolerant arithmetic operations and filters[32].

3.3 Delta Sigma Encoder

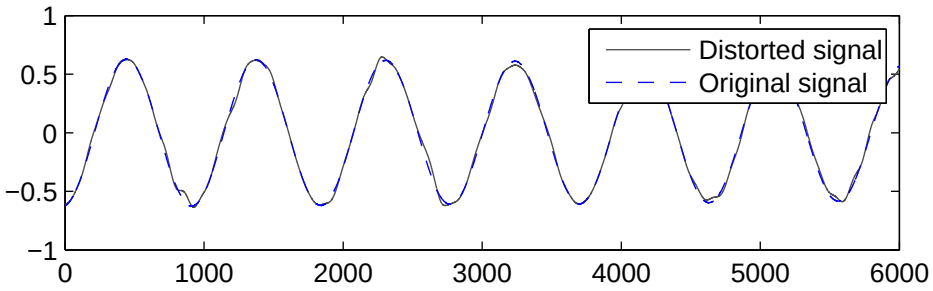
The purpose of a Delta Sigma Encoder is to convert a digital PCM signal into a bitstream representation while losing as little information as possible. While it is possible to use multi-bit quantization, many of the benefits of the code is only available when quantized to a single bit.

3.3.1 Digital-to-digital

The design of a fully digital encoder is somewhat different from that of an analog to digital converter. Obviously an encoder does not require any analog components. All



(a) 16 bit PCM coded signal



(b) Reconstructed Delta Sigma coded signal. $OSR = 64$

Figure 3.9: PCM and Delta-Sigma coded signals with 0.02% of bits randomly inverted. Signals are shown both with and without errors.

arithmetic operations are performed digitally. For a given input signal and initial state, the resulting bit sequence turns out the same in any number of attempts. Thus the encoder is fully deterministic, without the semi-random states associated with analog noise. Without analog components, process variations does not cause any discrepancies between a modelled system and its implementation in silicon. This makes it possible to use computer simulations on a higher level in the design process. Properties of the modulator may be precisely determined from simulations.

The fully digital system also causes some problems to be more apparent due to the lack of random variation of states. The deterministic behaviour increases periodicity in the modulator, causing pattern noise and limit cycles. A thorough study of these effects in a digital encoder, as well as a proposed solution is found in [6].

3.3.2 Requirements

Cross-correlation requires bitstream

The cross-correlation algorithm needs a true one-bit delta sigma signal representation to be valid. This drastically limits the amount of available choices for modulator structures. It rules out any modulator structure based around addition or subtraction at a stage later than the quantization. This includes the popular Multi-stAge noise SHaping (MASH) type of DSM. Several alternative structures are designed especially for low OSR operation, but are unsuited for the same reason. They depend on modulation in parallel before combining signals into a multi-bit delta-sigma *word stream* [22, 17].

Efficient cross-correlation requires low OSR

As discussed under section 2.2, the predicted savings in power consumption in the cross-correlator requires a low OSR. The system as a whole aims to demonstrate this principle, hence it is important to use as low OSR as possible. As discussed under section 3.2, the modulated signal quality quickly deteriorates as the OSR is lowered. An OSR of 8 is chosen as it is considered the best trade-off between signal quality and potential power savings of the bitstream cross-correlation. A lower OSR than 8 is hardly possible. As predicted by the results in figure 3.7, give an achievable SNR around 20dB, making it very hard to evaluate the quality of the rest of the system.

The restriction of OSR also affects the choice of modulator structure. Using several modulators in a series cascade may possibly result in single-bit output in certain cases. This subject is barely touched upon, as it requires a high oversampling ratio to allow a change in sampling rate at each step. This is clearly not an option for an OSR of 8.

3.3.3 Design choices

Of all the main blocks in the system, the modulator requires the smallest number of components, yet it is the single bottleneck limiting the signal quality of the whole system. For this reason, design choices for the encoder are directed only towards increasing the SNR, as this is reflected directly on the overall signal quality. Hardware complexity is only briefly considered, as it will have only a minor impact on the overall system power consumption.

Second order modulator

Initially a third order modulator was desired, but the choice of a second order seems natural after the OSR is decided. With an OSR as low as 8, the SNR gain of using a higher order modulator is already shown to be very small, if any. Figure 3.7 shows how the greatest advantage in quality is gained by going from a first to second order DSM at the given OSR. A third order modulator is known to exhibit less tonal behaviour than the second order. On the other hand, a second order modulator requires less design and simulation time to ensure stability.

In terms of hardware complexity, the choice of a second order modulator is sensible. The first order modulator requires one delay element and two adders. A modulator generally requires one delay element, one multiplier and two additional adders per order beyond the first. Increasing from first to second order roughly doubles the number of hardware elements, while increasing the predicted SNR by about 5dB. Similarly, an increase from second to third order increases complexity by about 1.5 times, highly dependent on the multiplier structure used, without a similar increase in expected SNR.

Error feedback structure

The fully digital arithmetic operations internal to the encoder allows for the choice of a structure based on error signal feedback instead of a single bit feedback loop. These structures are shown in figure 3.5a and 3.5b on page 27.

Remembering the error injection model of the quantizer, the transfer functions for this structure is quite simple to read from the figure.

$$Y(z) = X(z) + E(z) - H(z)E(z) = X(z) + \big[1 - H(z)\big]E(z)$$

Giving an STF of 1 and an NTF of $1 - H(z)$. The error-feedback structure is not suitable for analog filters, as it is very susceptible to component mismatch in analog realization of addition and filter coefficients [31].

For digital implementations, arithmetic precision is not an issue and the structure may prove to be somewhat more hardware-efficient. The error feedback signal is shown as a subtraction in the figure, but this is in fact realizable without any hardware. The error signal is simply the remaining LSBs not used in the output, and may be fed directly to the

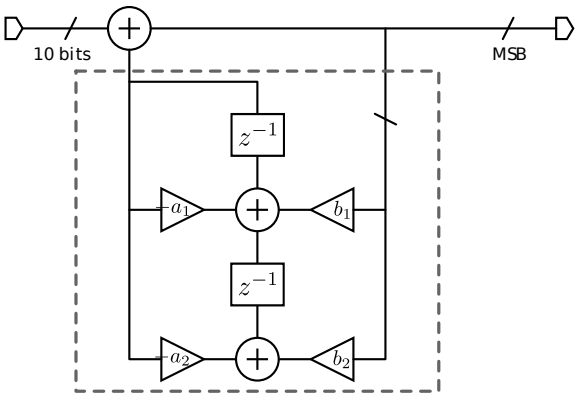


Figure 3.10: Second order error feedback structure with IIR filter.

loop filter. NTFs consisting of zeros only may be realized by a FIR loop filter, which further simplifies the modulator somewhat in terms of analysis and implementation.

The basic second order modulator may be realized using only two adders and two registers as opposed to four adders and two registers using the ordinary single bit feedback.

Bit width

The necessary bit width for the DSM input is related to the achievable SNR. Modulator SNR is expected to be quite low. To enable proper measurements of the DSM the incoming signal should have a somewhat higher resolution than the output. This ensures that resolution is not becoming a limiting factor. The standard way of expressing resolution is to calculate the Effective Number of Bits (ENOB) corresponding to the SNR level. This is defined as the number of bits necessary to recreate a PCM coded sinusoid with a given SNR.

$$ENOB = \frac{SNR - 1.76}{6.02}$$

For an expected SNR of $35-40$dB the upper limit is $ENOB = 6.4$. In other words, at least 7 bits must be used to ensure that the modulator is not limited by a low input resolution.

Loop filter with optimized zeros

The goal of producing as high quality output as possible, justifies use of an NTF with optimized zeros as described under section 3.2.2. This unfortunately removes some of the benefits of the error-feedback structure, as the filter needs to be an IIR type, having both feedback and feed-forward coefficients. Hence, the differences in hardware complexity between the error feedback and a single bit feedback becomes smaller for IIR loop filters. As seen from figure 3.10, the exchange of a FIR-type filter for an IIR filter introduces

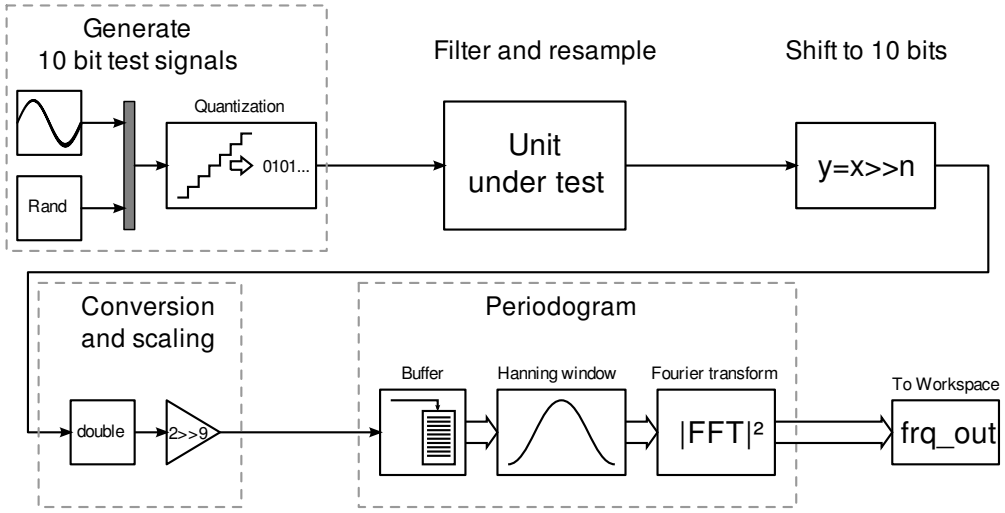


Figure 3.11: Simulink setup for calculating spectrograms.

two multiplication steps and two adders into the loop. Expected increase to SNR is only 3.5dB [31]. Hence it is an expensive method in terms of SNR vs. power for the DSM alone, and is only justifiable in the given context.

The noise transfer function is found using the Schreier's Delta Sigma toolbox for MATLAB. The software package includes functions for determining optimal spread of zeros in the NTF for a given modulator order and OSR. A good description of use and limitations is given in [31]

$$NTF = \frac{z^{-2} - 1.949z^{-1} + 1}{z^{-2} - 1.203z^{-1} + 0.4299}$$

H(z) for the error feedback structure may be found by substituting the b/a form for the NTF.

$$H(z) = 1 - NTF = 1 - \frac{b}{a} = \frac{a-b}{a}$$

$$H(z) = \frac{-0.7462z^{-1} + 0.5701}{z^{-2} - 1.203z^{-1} + 0.4299}$$

The filter function H(z) may be realized in the error feedback structure shown in figure 3.10.

3.4 Simulations

The DSM and filter blocks are all simulated using MATLAB and Simulink. The setup used is shown in figure 3.11. Some variations are used in scaling, depending on which filter is tested. This is to normalize all outputs to the same range for comparability.

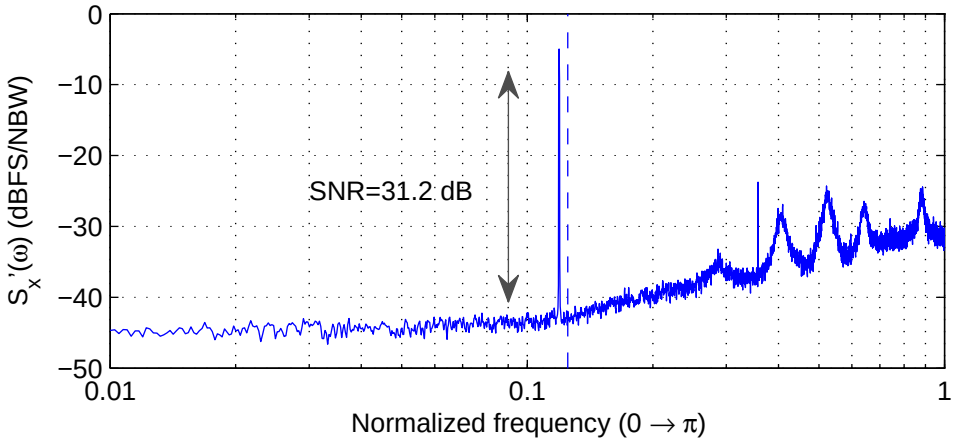


Figure 3.12: PSD of simulated modulator. Nyquist rate bandwidth indicated by dashed line.

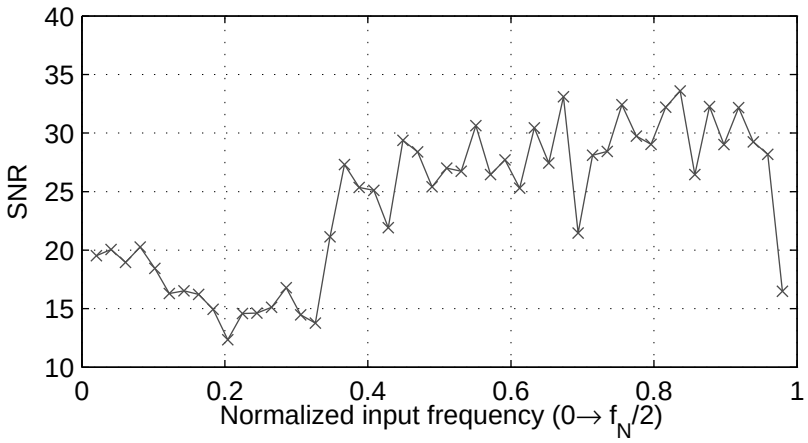


Figure 3.13: Calculated SNR for a range of input sinusoid frequencies. Simulations are run once for each input. Harmonics lowers SNR for $f_{in} < f_N/6$.

The model of the DSM is made to resemble the implemented circuit as closely as possible and uses bit shift operations to implement all coefficients in the loop filter as described in detail under section 6.1.5. Figure 3.12 shows the power spectrum of the modulator output when using an input signal of $0.7\sin(0.95\omega_N)$ sampled at 8 times the Nyquist rate. The signal to in-band quantization noise ratio is about 32dB in this simulation, which is within the expected values of $30-40$dB for a one-bit modulator at an OSR of 8. [31]

A more extensive study of the performance shows that the modulator is subject to in-band pattern noise. A sweep is done over a range of input frequencies and is shown in figure 3.13. The simulation is performed on a cascade of the final interpolation filter and the DSM. SNR is simulated and measured by script for each input sinusoid. Results in the figure show clearly how the signal quality is significantly poorer for inputs below frequencies of about $\frac{f_N/2}{3}$. The reason is found by looking closer at the Power Density Spectrum (PSD) of such a simulation. Harmonics occur at three times the input frequency and is the largest single source of unwanted noise on the output. Introduction of dithering in the modulator may have improved its modulation qualities.

The simulated SNR closest to the Nyquist frequency is in the transition band of the interpolation filter, and will not have its images sufficiently attenuated, resulting in a poor overall SNR.

3.4.1 Periodogram

The spectra used to characterize filter responses are calculated using Welch's method [30]. Time domain data is divided into overlapping sequences of length 8192 samples, with an overlap length of 1024 samples. All sequences are windowed using a length 8192 Hann window before applying a magnitude-squared Fast Fourier Transform (FFT). Normalization of the spectrum is done such that a full scale sinus signal is shown as 0dB [31].

Simulation of these models in MATLAB/Simulink runs in only a few seconds, using relatively large amounts of data. Some experimenting is done with different lengths of data and different numbers of averages for calculating the spectrograms. The final number of 8192 sequences is chosen as a compromise between simulation time and required resolution. Increasing data length beyond this does not increase resolution significantly. Shorter data lengths are sufficient, but the features of the spectra are clearer when large sequences are used.

The number of averages used in the spectra lie between 5 and 50, due to differences in sampling rates between filter types and between input and output rates. The difference between spectra is not problematic, as averaging mainly serves to smooth the graphs.

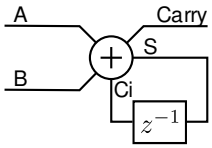


Figure 3.14: Structure of bitstream adder

3.5 Bitstream signal processing

The idea of signal processing directly on a DSM bitstream is motivated by several advantages when comparing to the traditional Nyquist rate multi bit alternative. Signal processing applications are often based around some form of sensor readout and requires an AD conversion in its initial step. Delta sigma converters are already in widespread use for this purpose, but requires digital-to-digital conversion before signal processing. Using bitstream processing methods eliminates the need for conversion steps between a Digital Signal Processing (DSP) and ADC/DAC, thus saving hardware.

Secondly, signal routing in a serial system is significantly simplified compared to parallel multi-bit systems. This also saves hardware area and simplifies the implementation.

Some arithmetic operations have proven to be quite effective when implemented in a bitstream environment. The cross-correlator is an example of this. There are also some drawbacks to the use of bitstream operations. Operations are less intuitive and harder to understand than their PCM counterparts. This results in more design and simulation time. More importantly, some operations may not be possible to realize, or realizations may not reasonable.

3.5.1 Addition

The most basic arithmetic operation is addition of two signals. Different implementations of bitstream addition are possible. A structure using a very simple form of noise shaping in the adder topology is presented in [29] and shown in figure 3.14. Its main advantage is simplicity. Using only an ordinary full adder circuit and a single delay element, enables addition of two single bit DSM sequences. The complexity is the same as for the serial adder discussed in section 6.1.6. Unfortunately, the addition introduces some amount of noise, and the resulting signal has a slightly poorer SNR than each of the operand signals. This is seen from figure 3.15 on the next page.

The addition may also be solved by the time-interleaving of two bitstreams. This requires a doubling of sampling rate to avoid loss of quality. Yet another approach is adding by averaging the two bitstreams [11]. An improved adder implementation is presented in [19] and is able to work at around 1GHz.

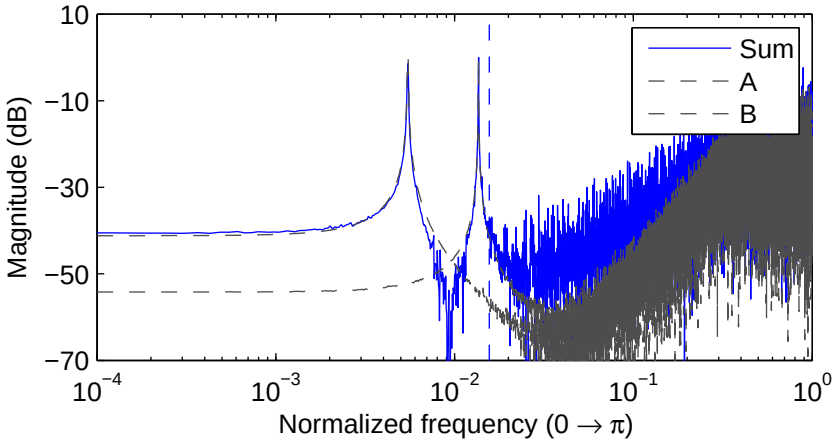


Figure 3.15: Signal quality after bitstream addition. $OSR = 64$

3.5.2 Multiplication

A simple multiplier using either the logical AND or XNOR operation is proposed, but results of this operation is not immediately satisfactory [29]. A large amount of samples are required for the value to settle requiring a very large oversampling rate. Even when settled, the signal quality is lowered significantly below that of the inputs. The value of such a circuit lies in its simplicity of implementation when compared to binary multipliers. Only a single gate is needed to achieve a crude multiplication of slowly varying signals. A quick simulation of the multiplication shows that it is hard to achieve a resolution better than about $20dB$, even when very high oversampling ratios and high order modulators are used.

4 Interpolation

A fully digital DSM encoder is used to perform the conversion from a binary encoded PCM signal to its oversampled bitstream representation. The encoder requires an oversampled input signal. For an analog DSM, oversampling is performed directly in the analog parts of the modulator by clocking these at the oversampling frequency. For a digital DSM, the oversampling of the incoming PCM signal is done by calculation in a digital interpolation filter as shown in figure 4.1. The filter increases the sample rate while maintaining the word size of the samples.

The theoretical background behind interpolation is given in section 4.1, aimed towards band-limited interpolation using digital filters. Requirements of the filter in a Delta Sigma system are identified in section 4.2. A two-step digital interpolation filter is designed for keeping hardware complexity low. An efficient FIR filter is presented in section 4.3. The CIC filter class is briefly discussed while stating reasons for parameter design choices in section 4.4. Because of the similar design of CIC filters for interpolation and decimation, both are discussed in this chapter, while details specific to the decimator implementation are given in chapter 5. Finally, models and simulation results for the interpolation filter are presented under 4.5.

4.1 Interpolation filter

The purpose of an interpolation filter is to up sample a signal, without either removing or adding information content. A signal sampled at the Nyquist rate is re-constructable according to the Nyquist theorem. An ideal Nyquist rate DAC may perfectly reconstruct a signal in the analog domain with frequency components up to $f_N/2$. In other words, the information content in a Nyquist rate signal is band-limited by $f_N/2$, where f_N is the

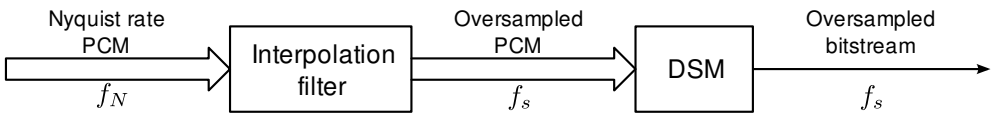


Figure 4.1: Upsampling from Nyquist rate to oversampling rate prior to Delta Sigma modulation.

Nyquist sampling rate.

There are a few different approaches to the interpolation problem. A simple, but impractical approach is to reconstruct the analog signal, and resample it again at a higher rate. This requires both an A/D and and a D/A conversion, which in turn is inefficient and may introduce both non-linearities, analog noise and further quantization errors in the resulting signal. Advantages of this method is that the rate change factor may be arbitrarily chosen, and it does not result in imaging due to upsampling.

The more common approach is to interpolate directly in the digital domain by recreating the desired number of samples in between the known samples. The simplest way of filling in the blanks is to copy the previous known value. The hardware implementation is very simple. A register sampling at the increased clock rate is all that is needed to perform what is known as a zero-order interpolation.

It is useful to consider this operation on a higher abstraction level. From a signal processing point of view, this equals a two step process. First the signal is upsampled, or zero-padded, with the necessary number of samples to increase the rate by the interpolation factor. Then a filter is applied to the signal, calculating the missing values. The FIR filter performing a zero order hold interpolation is

$$h_0[n] = \sum_{k=0}^{I-1} h_0[n-k] \quad \overset{Z}{\leftrightarrow} \quad H_0(z) = \sum_{k=0}^{I-1} z^{-k}$$

Its effect on a signal zero padded with $I-1$ samples is illustrated in figure 4.2a.

This is obviously no good approximation of the original signal, and may be improved using first order or linear interpolation. Unknown samples are instead calculated along the straight line between two neighboring samples. The filter interpretation of this operation is easiest to find using the continuous time function of a general linear interpolation.

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

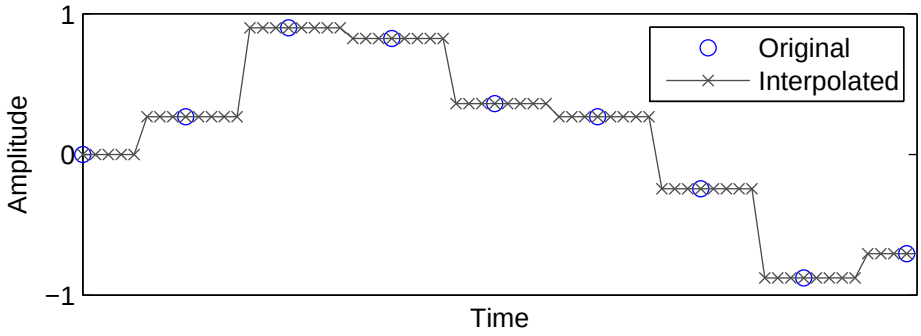
Inserting for the two known discrete time values $x[0]$ and $x[1]$ for a sampling time $T = IT'$.

$$y[n] = x[0] + \frac{x[1] - x[0]}{IT' - 0}(nT' - 0), \quad 0 \leq n < I$$

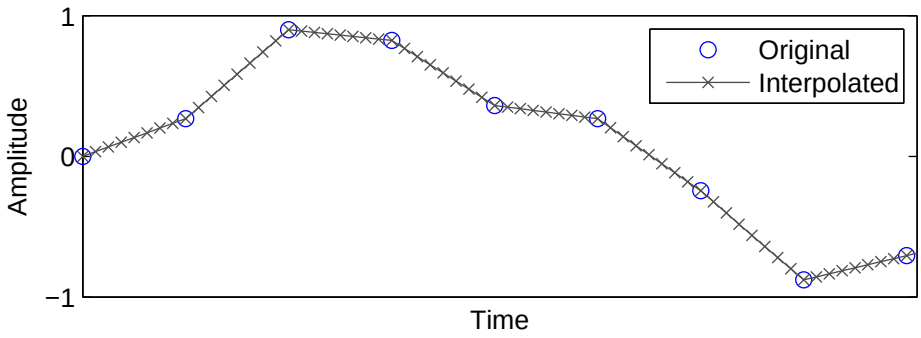
This may be rewritten as

$$y[n] = \big(1 - \frac{n}{I}\big)x[0] + \big(\frac{n}{I}\big)x[1], \quad 0 \leq n < I$$

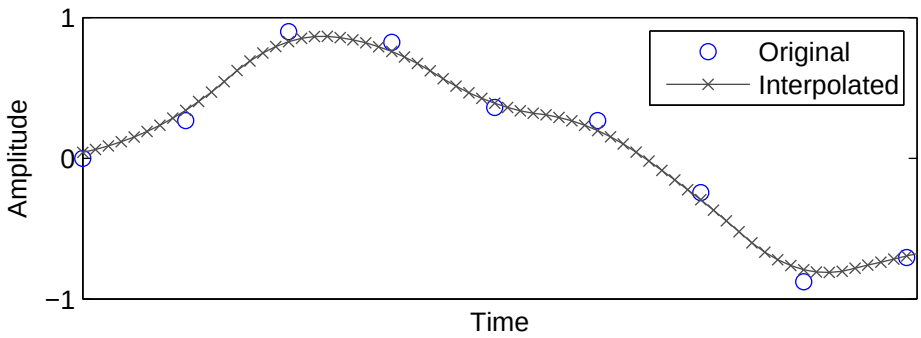
As with zero order interpolation, the length of the necessary filter is simple to find. For each output value to be a function of exactly two of the original samples, the length of the filter must be $N = 2I - 1$. The impulse response of the required interpolation filter may be



(a) Zero order



(b) First order



(c) Second order

Figure 4.2: interpolation by eight. Outputs are shifted for comparison.

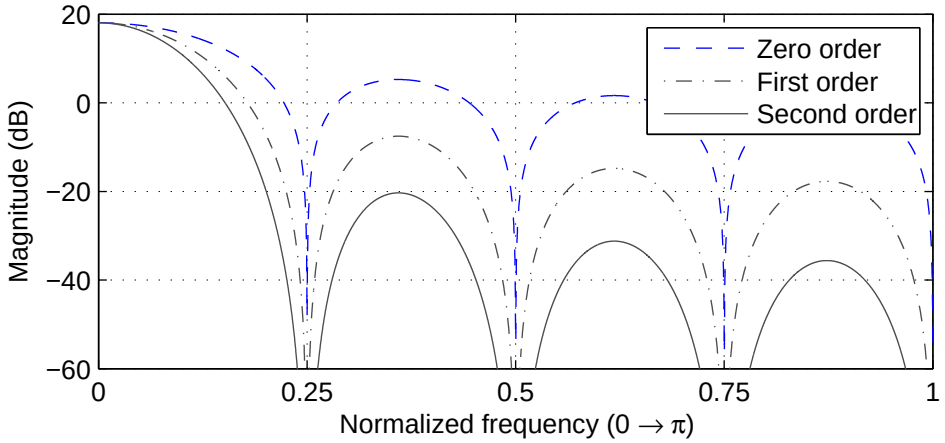


Figure 4.3: Frequency domain response of simple interpolation filters. $I = 8$.

shown to have a triangular impulse response [30]. Calculating the z-transform of the filter yields the square of the zero-order hold filter transfer function.

$$h_1[n] = h_0 * h_0 \quad \overset{\mathcal{Z}}{\leftrightarrow} \quad H_1(z) = H_0(z)^2 = \Big[\sum_{k=0}^{I-1} z^{-k}\Big]^2$$

Higher order interpolation functions are found by increasing the power of the transfer function. The first three powers are illustrated in figure 4.2 on the preceding page. Starting with the second order interpolation, the resulting function is not guaranteed to pass through the original data points. In fact, only slowly changing signals will properly pass through the correct data points due to frequency dependent attenuation. This effect is easy to see from the Fourier evaluation of the filter function. Frequency responses for these simple filters are shown in figure 4.3 and their limitations may be seen in detail. Higher frequencies within the passband are significantly attenuated. This motivates the use of a more complex filter with a flatter passband frequency response.

4.1.1 Ideal interpolation

The ideal interpolation filter is the brick-wall filter with a cut-off at half the original sampling rate and a flat passband (Fig.4.4). The impulse response of such a filter is found by the inverse Fourier transform and is well known to be the sinc function, which in continuous time is given by

$$g(t) = \frac{\sin(\frac{\pi}{T_s}t)}{\frac{\pi}{T_s}t} \tag{4.1}$$

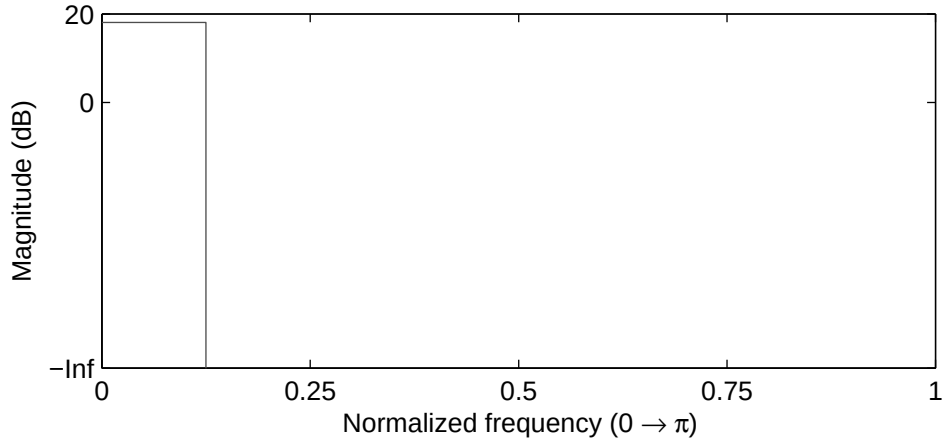


Figure 4.4: Frequency response for the ideal interpolation filter using $I = 8$. Passband gain equals I.

where T_s is the original sampling interval. This function oscillates around zero for an infinitely long interval, with zero-crossings at every integer multiple of T_s except zero. The ideal interpolator thus requires an infinitely long filter, and only an approximation is practically possible. This knowledge is the basis for design of band-limited interpolation filters with better performance than the sinc-in-frequency filters described in section 4.

Interpolation by the sampling theorem

The background for band-limited interpolation may be explained in terms of the sampling theorem. The sampling theorem states that a band-limited signal may be completely recreated from its samples given that the sampling rate is higher than two times the bandwidth. Shannon has shown how such a signal may be recreated by the ideal interpolation formula [33].

$$x(t) = \sum_{m=-\infty}^{\infty} x(mT_s) \frac{\sin\left(\frac{\pi}{T_s}(t - mT_s)\right)}{\frac{\pi}{T_s}(t - mT_s)} \tag{4.2}$$

The formula consists of a sum of the sinc function $g(t)$ in equation 4.1, shifted by nT_s and weighed by the discrete time samples $x[m] = x(mT_s)$. This describes the signal in the time domain, but requires an infinite number of samples and the infinite length sinc function, which reduces the practical use of the formula.

In theory, this is useful in showing how to find the ideal interpolation filter. Assume a resampling of the time-domain signal $x(t)$ at a higher sampling rate such that $T_s = IT_h$. Since the initial bandwidth is smaller than the new bandwidth, the requirements of the sampling theorem is still satisfied and equation 4.2 may be rewritten with the smaller

sampling interval.

$$x(t) = \sum_{m=-\infty}^{\infty} x(mT_s) \frac{\sin\left(\frac{\pi}{lT_h}(t - mlT_h)\right)}{\frac{\pi}{lT_h}(t - mlT_h)} \tag{4.3}$$

Substituting for the discrete time sequences $x[m] = x(mT_s)$ and $x[n] = x(nT_h)$, which are the initial samples and the interpolated samples, respectively.

$$x[n] = \sum_{m=-\infty}^{\infty} x[m] \frac{\sin\left(\frac{\pi}{l}(n - ml)\right)}{\frac{\pi}{l}(n - ml)} \tag{4.4}$$

For clarity, this may be rewritten as the sum of the sinc function weighed by the original sample values and shifted in time.

$$x[n] = \sum_{m=-\infty}^{\infty} x[m] g[n - ml] \tag{4.5}$$

The sinc function in discrete time may be written as

$$g[n] = l \frac{\sin\left(\frac{n\pi}{l}\right)}{n\pi} \tag{4.6}$$

The frequency response of this function may be shown to be the brick wall filter. For $l = 8$, this function is shown in figure 4.4 on the previous page.

Interpolation as convolution

Equation 4.5 resembles a convolution, except for the difference in sampling rate between the indexes n and m. This is solved by introducing the zero padded sequence $\tilde{x}[k]$, which has the same values as $x[m]$, but with $l - 1$ zero values in between. The zero valued samples does not affect the summed result, so these sequences may be interchanged freely. This allows for the exchange of summation indexes with a set of indexes $k = ml$, using the same sampling rate as n.

$$x[n] = \sum_{k=-\infty}^{\infty} \tilde{x}[k] g[n - k]$$

or, with the asterisk as convolution operator

$$x[n] = \tilde{x}[n] * g[n] \tag{4.7}$$

This result is the convolution of the ideal filter sequence with the zero padded input sequence. This means that the interpolation may be implemented using an upsampling followed by an ordinary FIR filter.

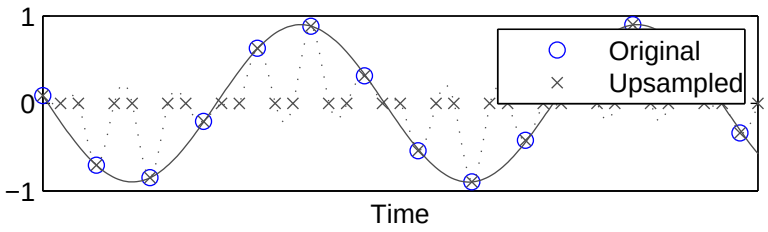


Figure 4.5: Upsampling by zero padding introduces new frequency components into the representation of the signal.

Upsampling causes imaging

Upsampling a signal by zero padding causes what is known as *images* in the output signal. The upsampled signal is a representation of the original frequency components as well as higher frequency components caused by the introduction of zeros. Figure 4.5 illustrates how zero padding of a sinusoid introduces unwanted frequency components into the signal.

Approximation of the ideal filter

The ideal discrete interpolation filter is based on two requirements that are impossible in a practical implementation. The filter sequence $g[n]$ must be infinitely long, as it asymptotically approaches zero. Secondly, the sampled sequence is required to be strictly band-limited by the sampling theorem. This implies that an infinite number of signal samples must be used in the convolution sum, because an abrupt step at the beginning or end of the sampled signal causes some signal frequency content to be outside of the required band-limit [33].

Windowing the *sinc* function is essentially the solution to the first requirement. Simply cutting the sequence down to a desired length equals the multiplication by a rectangular window. This causes significant spectral leakage and a wide transition band. Using a more suitable window function is preferred.

Alternatively, the filter may be approximated using any filter design method. Because of the simple characteristic of the ideal interpolation filter, an approximation is easy to design using any filter design software, once a few requirements are identified.

4.2 Cascaded interpolation filter

4.2.1 Requirements in a DSM system

Before designing the interpolation filter it is necessary to identify the requirements for it to work as intended. Two of the requirements may be stated precisely: The interpolation factor is equal to the OSR of the modulator and the phase response of the filter must be

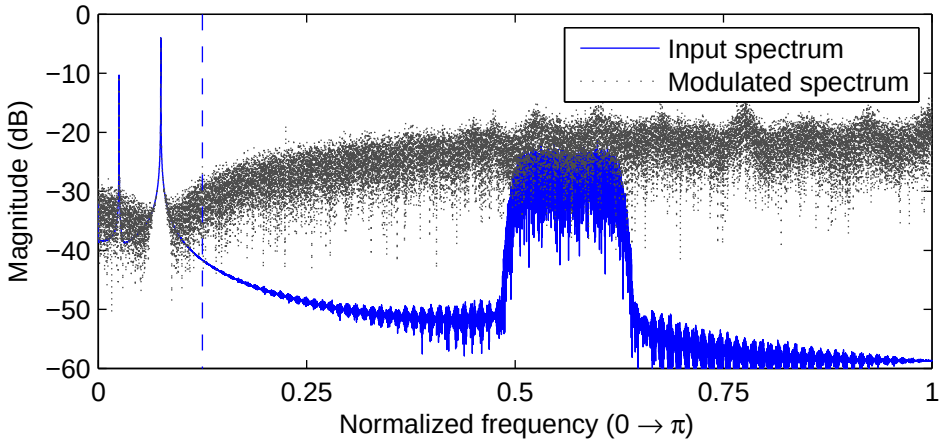


Figure 4.6: Noise frequencies above signal bandwidth are masked after modulation.

linear in the passband to avoid any signal distortion. This is guaranteed simply by using only FIR filters and CIC filters in the design.

Most requirements are not precisely stated. These are left as trade-offs between required quality and filter complexity.

Order

A crude estimate of digital filter complexity and hardware requirements is its order. In the general case, order refers to the number of delay elements in the filter implementation. For FIR filters, this also corresponds to the number of coefficients or hardware multiplications. As will be seen later this is not quite true, as it is possible to take measures to reduce complexity independently of order. For now the order will work as an estimate of the required hardware area. It should be kept as low as possible to reduce the overall power dissipation.

Passband

A good interpolation filter should have as flat a passband response as possible. This conserves the original signal and avoids frequency-dependent attenuation as shown earlier in this chapter.

Stop band

The low normalized cut-off frequency leaves a large frequency region to be attenuated. The exact frequency characteristic in the stop band is not important, as long as it is sufficiently

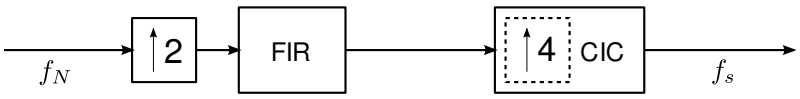


Figure 4.7: Two stage interpolation filter reduces overall complexity.

damped. As the filter is to be followed by a noise-shaping modulator, the stop band allows for signal aliasing at a level equal to or lower in power than the out-of-band noise level.

These unwanted frequency components will be hidden in the shaped quantization noise after modulation. An example is shown in figure 4.6. This shows an input signal containing band-limited noise. After modulation, the noise does not affect the signal quality in the passband compared to modulating the signal without added noise (not shown). This quick experiment also indicates that having a certain amount of noise in the stop band actually decreases the amount of idle tones in the modulated signal.

In other words, the stop band attenuation needs to be highest right above the signal band limit f_N. At this point, the attenuation must be high enough to reduce unwanted signal power below the expected noise floor of the modulator. The predictions are shown in section 3.2 to be $30 - 35$dB.

4.2.2 Multi stage filtering

A problem with the narrow pass-band is that it requires a high filter order. The multi rate nature of the filter enables a solution using multiple stages. Digital filters may be cascaded in order to take advantage of changing filter requirements at different sampling rates. The interpolation may be split into steps as shown in figure 4.7. The requirements of the first step is the same as the overall filter requirements. However, by reducing the upsampling ratio prior to the filter to a factor of two instead of the full interpolation factor of eight, the cut off frequency of the first step is moved to $\pi/2$ instead of $\pi/8$. This significantly lowers the required order of the filter.

A simple comparison using filter design software shows the difference: Two low pass FIR filters with the same requirements, but cut-off frequencies at $\pi/2$ and $\pi/8$, results in minimum orders of 28 and 114 respectively. The reduction in complexity is proportional with reduction in cut-off frequency.

In addition to the complexity savings, using a multi stage implementation also lowers the overall power dissipation due to clock rate. By splitting into steps, the most complex filter will be operating at a rate of two times the original sampling rate instead of the full OSR. This reduces dynamic power consumption proportionally to the clock rate reduction. On top of this comes a reduction in static power consumption due to a lower overall circuit complexity.

Later stages have eased demands and may be realized using simpler filters. This allows

the use of different classes of filters with other benefits. In particular this enables the use of a FIR filter at the crucial first stage, and later stages may be implemented using more hardware efficient filters such as the sinc-filter or CIC-filter.

The frequency response of a cascade of filters is the product of each individual filter response. When filter steps are operating at different rates, the frequency responses must be normalized to the highest rate before multiplication.

4.3 Step 1: Finite Impulse Response (FIR) filter

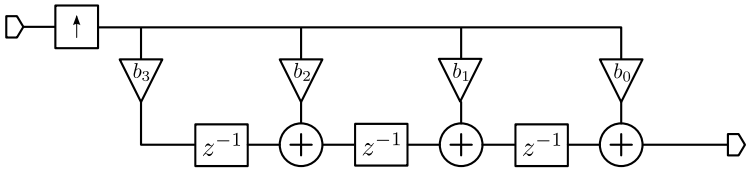


Figure 4.8: Upsampling FIR filter. Generally requires one register and one gain step per filter order.

The low change in sampling rates for the interpolation filter limits the number of choices concerning the multi stage composition. The first step of the interpolation filter should be an upsampling by two, followed by a FIR-filter with a reasonably sharp cut-off. The filter is to reduce aliases caused by the first upsampling.

The choice of a FIR filter is somewhat costly. In its general form it requires one multiplication, one summation and one storage element per added filter order. This requires a certain amount of hardware area. Careful design of the filter function may reduce this substantially, however.

4.3.1 Half-band filter

By letting the passband and the stop band have an equal bandwidth centered around $\pi/2$, we may use a class of filters known as half-band filters. These are ordinary FIR filters with every other coefficient set equal to zero, except for the middle coefficient. The necessary multiplications and additions are reduced by a factor of two.

The basis behind the savings may be seen from the ideal discrete interpolation filter given in equation 4.6. For an interpolation factor of $I = 2$, all even terms with the exception of $n = 0$ are reduced to zero.

$$h[n] = \frac{\sin\frac{n\pi}{2}}{\frac{n\pi}{2}} = \begin{cases} 0 & |n| = 2, 4, 6, 8... \\ \text{sinc}(\frac{n\pi}{2}) & \text{otherwise} \end{cases}$$

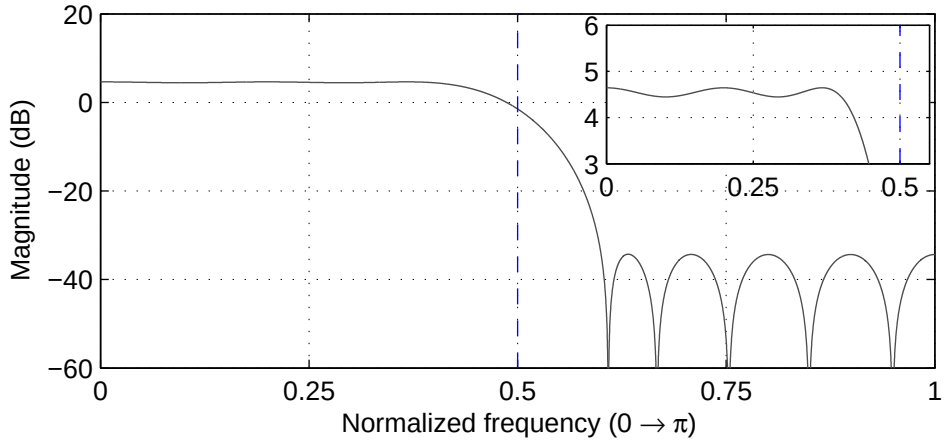


Figure 4.9: Fir half-band interpolating filter of order 18. Original bandwidth indicated as dashed line. *Inset:* Passband ripples are ± 0.1dB.

By applying a window function to this infinite filter response results in a realizable filter using about one half the number of adders and multipliers.

The windowing method does not produce the optimal filter response. Instead calculations are done with filter design software using the more complex Remez algorithm to get an equiripple filter with a sharper transition than if the window method had been used [24]. The stop band attenuation needs to be 40dB for the first step filter requirements to be satisfied. The transition band is set from 0.4π to 0.6π and results in a filter that may be realized using an order of 18. This requires only 11 coefficients in the implementation. A low hardware complexity is considered more important to the result than a high performance filter. The resulting filter response is displayed in figure 4.9. All coefficients are implemented using hardware-efficient quantization of values, as discussed in section 6.1.5.

As seen from the detail in figure 4.9, the equiripple method results in equally large ripples in the pass band as in the stop band. This means that frequency dependent variations of up to $-40\text{dB} = 0.01$ will occur in the filtered signal. For the signal amplitude of 512 steps, this results in an expected magnification or attenuation of ± 5.

4.3.2 Structural improvements

Efficient implementation of FIR filters is a huge field of research. FIR filters are highly flexible and simple to design for linear phase response and guaranteed stability, making them suitable for a wide variety of applications. The draw-back is a very high number of calculations, hence the potentially large gains in using specialized and reduced structures. Only a few such methods are used in the chip implementation of the FIR filter. A few others have been considered.

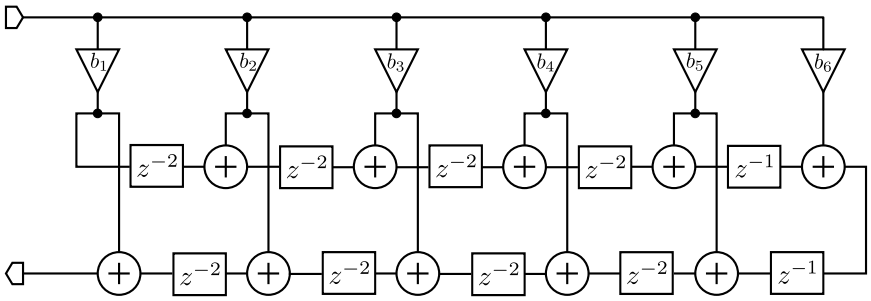


Figure 4.10: Suggested structural improvement. Single calculation of symmetric coefficients reduces the number of gain steps from 11 to 6.

Symmetric filter coefficients

For filters with a symmetric set of coefficients, simplifications are possible by restructuring so that equal coefficients are only calculated once. This makes the structure somewhat more complicated, but removes one half of the multiplications [25]. If this method is used, the half band filter is possible to implement using 6 multipliers, 10 adders and 18 registers. Figure 4.10 shows how this may be done for the transpose form of FIR filter. The result is a somewhat more complex way of interconnecting signals. This is ordinarily a smaller issue than a large number of cells, and has no impact on consumed power.

4.4 Step 2: Cascaded Integrator Comb (CIC) filters

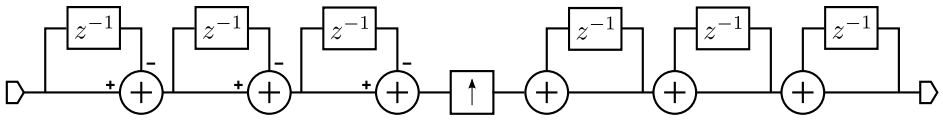
CIC filters are a class of efficient filters implementing the simple sinc-in-frequency filters discussed under section 4. They are developed with the intent of reducing the number of high frequency arithmetic operations in a multi rate system. A thorough analysis of benefits and limitations is given by Hogenauer [13].

Their main strength is simplicity and a resulting low hardware demand. Either interpolation or decimation may be performed, but the filtering functions available to these filters are limited to the form

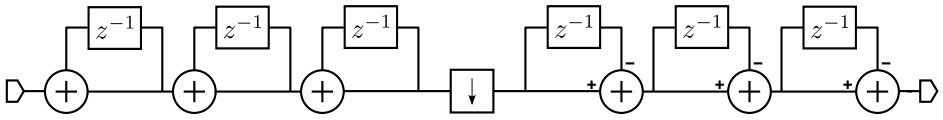
$$H(z) = \Big(\sum_{k=0}^{RM-1} z^{-k} \Big)^N \tag{4.8}$$

where R, M and N are the only available parameters. The above function is referenced to the high sample rate of the filter and shows how the CIC filter realizes the same function as a cascade of N equal FIR filter functions using only unity coefficients.

CIC filters may be considered specialized FIR filters where several measures have been taken to reduce their hardware needs. Multiplications are removed by restricting all filter coefficients to unity. Realization of a unity coefficient requires no operation at all. Secondly,



(a) Interpolating filter



(b) Decimating filter

Figure 4.11: Cascaded Integrator Comb (CIC) filters of third order

the up- or downsampling step is moved to the center of the filter, switching places with the comb filter section. This results in a reduction of necessary register space by a factor equal to the rate change R. This may be explained by what is known as the noble identities. In terms of upsampling, the identify may be stated as

$$\mathcal{U}_R\{X(z)\}H(z^R) = \mathcal{U}_R\{H(z)X(z)\} \tag{4.9}$$

where $\mathcal{U}_R$ is the operation of upsampling by R. By placing the upsampling operation after the comb filter section, the comb filter step functions are simplified from $H(z^R) = 1-z^{-RM}$ to $H(z) = 1 - z^{-M}$, thus saving much register space. Both the identity and the savings are similar for the downsampling filter.

One further benefit of this filter is that is is built using only two basic elements: Accumulators consisting of a single delay and a feedback adder, and differentiators using M delay elements and a feed-forward subtraction. The resulting regular structure is simple to implement using a hierarchical build-up. Both basic elements are built using registers and adders, and these may in turn be put together according to the number of bits and number of stages. All steps are made equal as opposed to a general FIR filter where coefficients differ. The basic structures of both the interpolating and the decimating CIC filters are shown in 4.11a and 4.11b.

4.4.1 Filter parameters

With only three available parameters, the filter class does not allow for much control over the filter function.

Order *N*

The CIC filter order decides the number of cascaded sections in the implementation. For a unity differential delay, the order directly corresponds with the order used by the simple

interpolation FIR filters discussed under section 4. The frequency responses of the first three orders of CIC filters may be seen in figure 4.3 on page 46, again referenced to the high sampling rate.

An increase in order is done by simply adding one comb filter section and one integrator section to each side of the filter. Due to the bit growth mentioned earlier, the hardware cost of adding an order grows exponentially as each added section is larger in width than the last.

Differential delay M

The parameter specifies the number of delay steps in the comb filter sections. Increasing this will increase the number of zeros in the transfer function as shown in figure 4.12. Increasing the number of zeros has a small effect on the overall attenuation of the filter, but also a slightly lower cost than adding an order. It requires only one added register instead of the two registers and two adders of an increased order.

Introducing zeros may seem useful in increasing the overall stop band attenuation. However, the placement of zeros for $M = 1$ is already near optimal when considering the pass bands from the first filter step as shown in figure 4.12a. Any additional zeros will be positioned within the stop bands and contribute little to the final response. This effect is even more pronounced when preceding filter steps result in narrower passbands. Zeros also cause the pass band to drop significantly faster than when increasing the order, such that all in all the differential delay parameter is of little use in most cases. A single step delay is the best choice for this purpose.

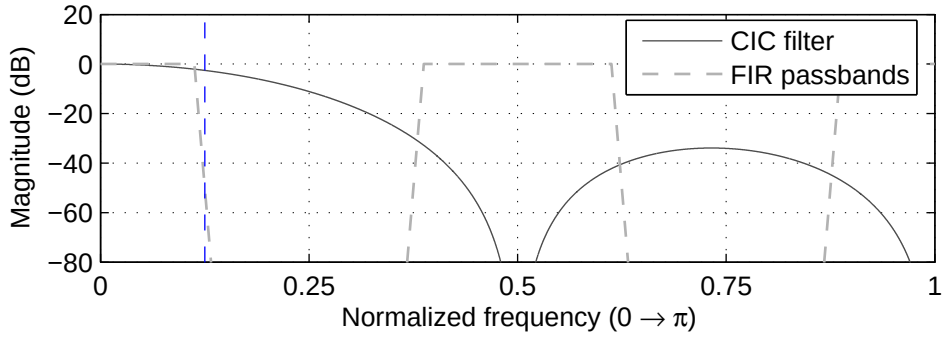
Rate change R

The up or down conversion rate for the filter is often decided by its application, and is not entirely free to be changed. It is still useful to look at the effects of this parameter, as it is important in determining the optimal composition of a multi stage filter as well as in understanding the effect of the CIC filters.

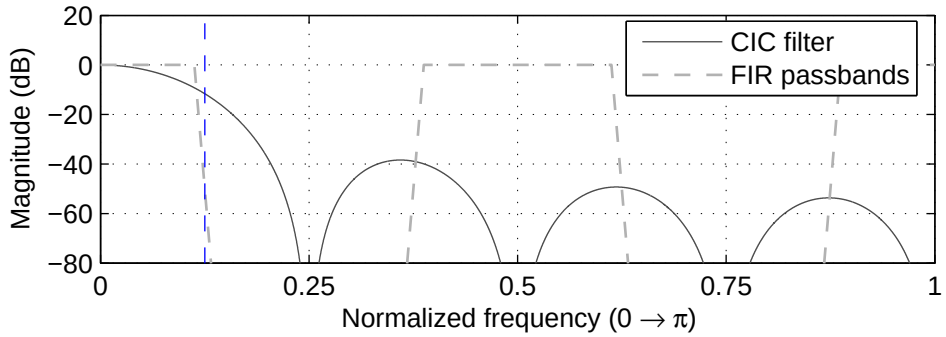
The effect on the filter response may be seen from equation 4.8 to be the same as the differential delay parameter. The rate change factor does not directly affect the filter composition and no change to the filter structure is needed to accommodate a different R. As seen in section 4.4.2, the necessary bit widths are largely dependent on the rate factor.

Increasing the rate change factor for the filter adds zeros to the transfer function in the same way as for the differential delay.

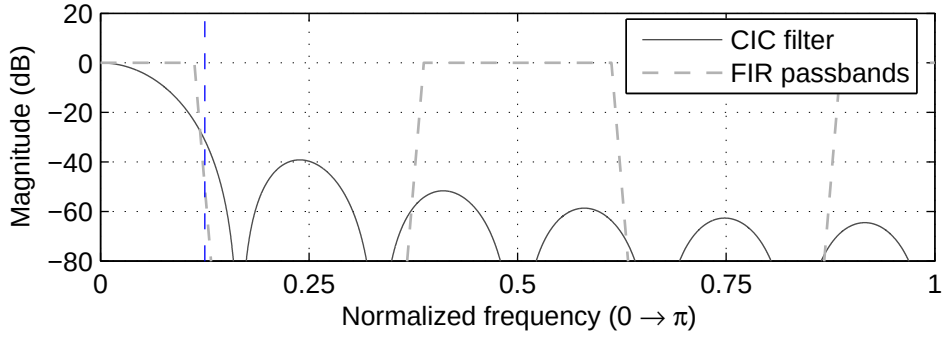
One interesting result of this is that the filter may be designed for use with different rates. The filtering function changes with the rate change factor, as opposed to other digital filters where the response is statically related to the sampling rate. This means that the same block of hardware may be used to interpolate for a varying rate change factor.



(a) $M = 1$



(b) $M = 2$



(c) $M = 3$

Figure 4.12: Various differential delays in 3.order CIC filter. Zeros are added outside the passbands of the preceding filter step.

4.4.2 Bit growth in CIC filters

One difficulty using CIC filters is the growth in word size throughout the filter. This leads to some reduction in the efficiency of the CIC filters, in the sense that the required hardware complexity does not grow linearly with an increase in the filter order. The reason is that higher orders requires a larger number of bits per step than lower orders do.

Calculations of step sizes are shown in Hogenauer's original paper [13].

Interpolation filters

The filter gain of each step may be found from Hogenauer's formula.

$$G_i = \begin{cases} 2^i & i = 1, 2, ..., N \\ \frac{2^{2N-i}(RM)^{i-N}}{R} & i = N+1, N+2, ..., 2N \end{cases} \tag{4.10}$$

The word size increase is then found by

$$W_i = \log_2 G_i$$

As opposed to the decimation filters, a shortening of word lengths is not possible for interpolation filters until after the last integrator step. The reason is that a small truncation error introduced to the integrators will grow without bounds and cause the filter to become unstable.

Decimation filters

For decimation filters the calculations are simpler. The total filter gain is

$$G = (RM)^N \tag{4.11}$$

The register size increase is then given by

$$W = \log_2 G$$

Hogenauer shows how this is the size of each step of the decimation filter, and how it might be permissible to reduce the register sizes by truncation of one or more of the LSBs in such a way that the introduced error does not affect the output of the filter [13]. This is not implemented in the CIC decimation filter and is left as a potential improvement. The method reduces total hardware size without impacting much on the filtering quality.

4.5 Simulations

Filters are simulated using the same setup as for the DSM shown in figure 3.11 on page 37.

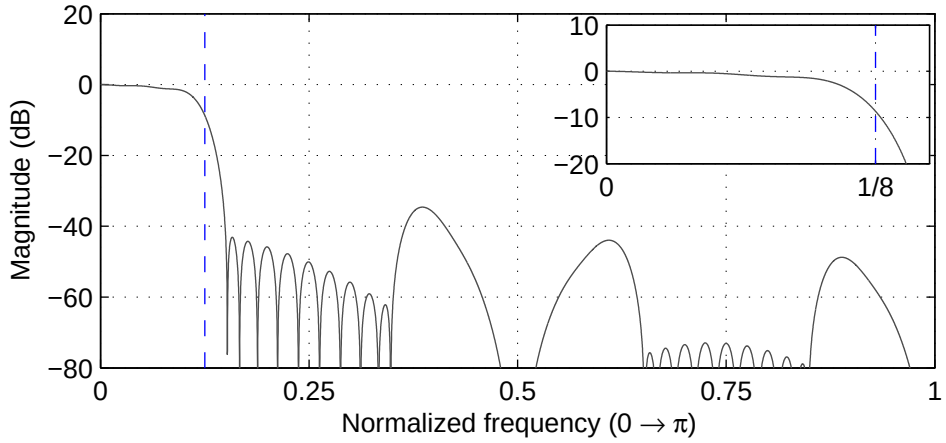


Figure 4.13: Cascaded interpolation filter response. Calculated from transfer functions. *Inset:* Passband response has 10dB attenuation close to the bandwidth.

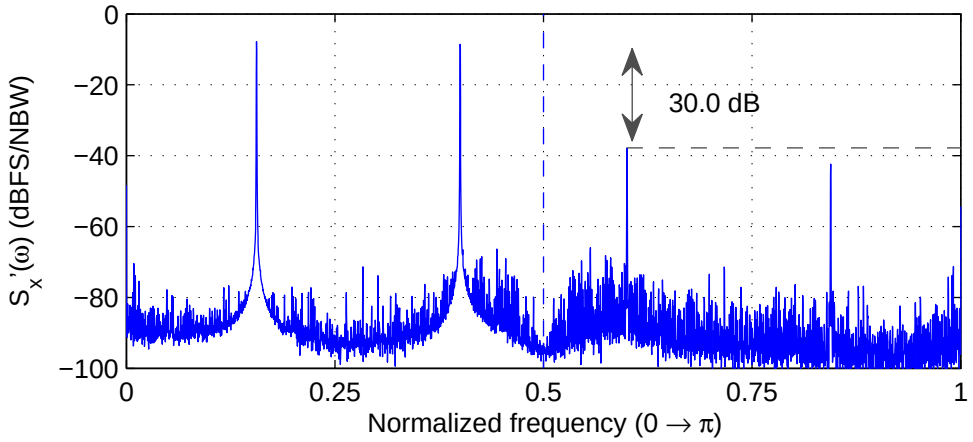


Figure 4.14: PSD for simulated FIR half-band filter. Filtered and upsampled by a factor 2. Nyquist rate signal band indicated by dashed line.

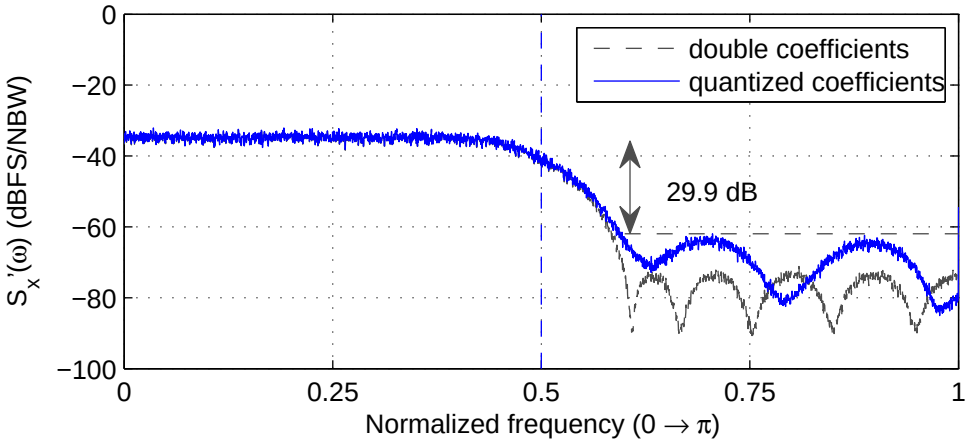


Figure 4.15: PSD of both modelled FIR interpolating filters, showing response to white noise input signal. Nyquist frequency $f_N/2$ indicated by dashed line.

4.5.1 FIR Half-band filter

The FIR half-band filter is simulated in Simulink. The effect of the upsampling by two is shown in figure 4.14. Input sines are given arbitrary frequencies, resulting in a difference in alias attenuation level due to the frequency dependence.

The filter is modelled both using full-precision multiplications and quantized coefficients. For the full-precision filter, the coefficients are represented using double length floating point numbers native to MATLAB. The quantized filter uses coefficients represented by a sum of two bit-shift operations, as detailed in section 6.1.5. The effect of the quantization on the filter response is a reduction of stop band attenuation. By characterizing the filter using a flat white noise input, shown in figure 4.15, the reduction is seen to be about 10dB. The full-precision filter shows a stop band attenuation of 40dB, as expected from the filter design software. The quantization of coefficients decreases this to about 30dB. Both filters show a pass-band width of about 0.2π, normalized to the upsampled rate.

4.5.2 CIC Interpolation filter

The CIC part of the interpolation filter is modelled in Simulink to resemble the circuit implementation. This filter is cascaded with the FIR filter, hence the upsampling ratio of this step is set to 4. Figure 4.16 shows how the upsampling results in attenuated images in the output spectrum. These are not lowered by more than about 20dB for this input. This illustrates the need for a sharper filter prior to the CIC filter. The lack of a steep transition band is seen clearer in figure 4.17. This figure also shows the level of attenuation for all image frequencies.

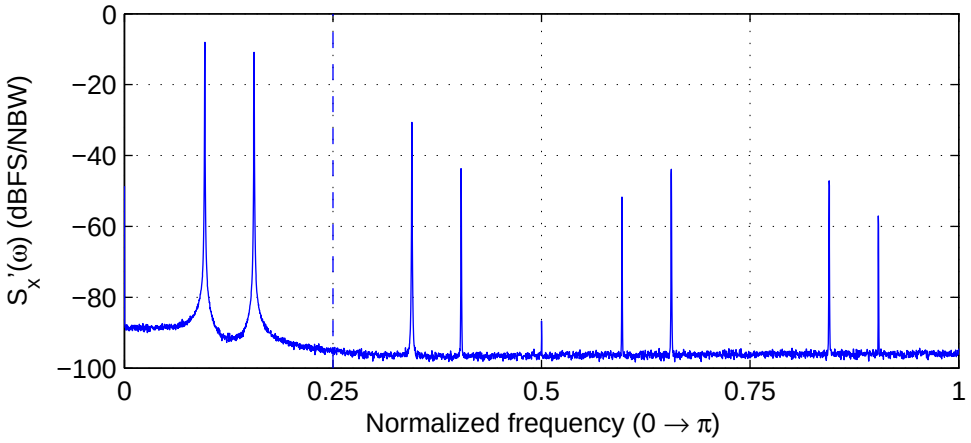


Figure 4.16: PSD of simulated CIC interpolation. Input is filtered and upsampled by 4. Low rate signal band indicated by dashed line.

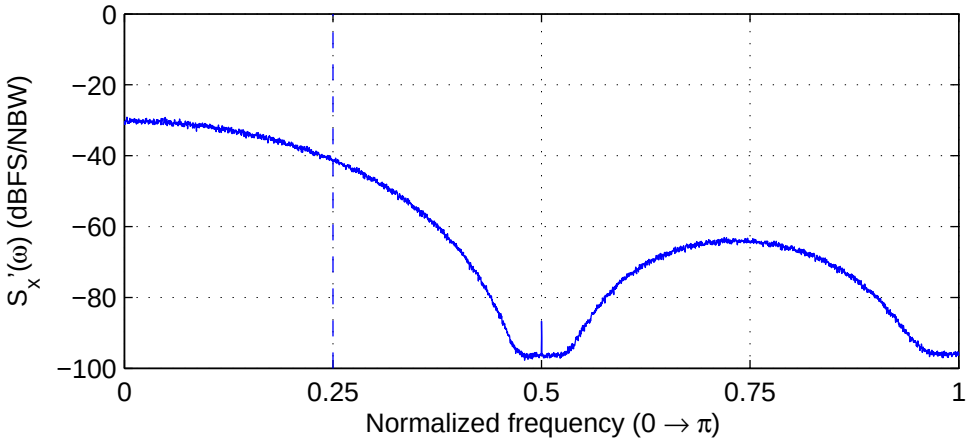


Figure 4.17: PSD of simulated CIC interpolation. Response to white noise input signal. Nyquist frequency $f_N/2$ indicated by dashed line.

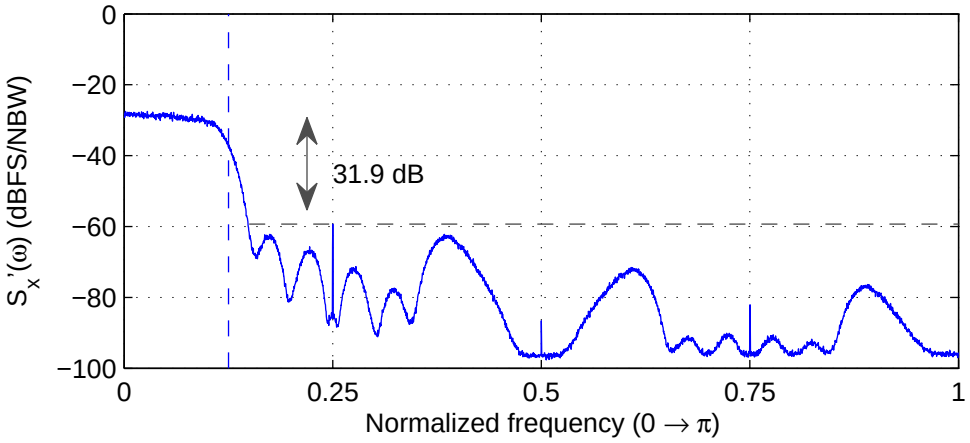


Figure 4.18: Cascaded filter response using white noise input. Cascade of quantized FIR filter and finite precision CIC filter.

4.5.3 Cascaded interpolation filter

Simulation results of the FIR filter model followed by the CIC interpolator is shown in figure 4.18. Results are according to prediction from the transfer function in figure 4.13. The overall reduction due to the FIR filter coefficient quantization is measured to 35dB − 31.9dB = 3.1dB.

4.5.4 Interpolation and DSM

Figure 4.19 on the next page shows output by the full modelled interpolation filter after modulation by the DSM. Output signal quality is according to the expectations from the DSM output using an ideal sinusoid input in figure 3.12. It should be noted that resulting SNR is only a single measure corresponding to a single input signal. A more thorough investigation using this setup have been given in 3.4. Experimenting with different frequencies yields somewhat different behaviour than shown in the figure. Results from the simulations on the modulator, using a full-resolution input signal shows an output SNR of 31.2dB. This suggests that the interpolation filter reduces the bitstream signal quality by an estimated 2 − 4dB below the potential of the modulator.

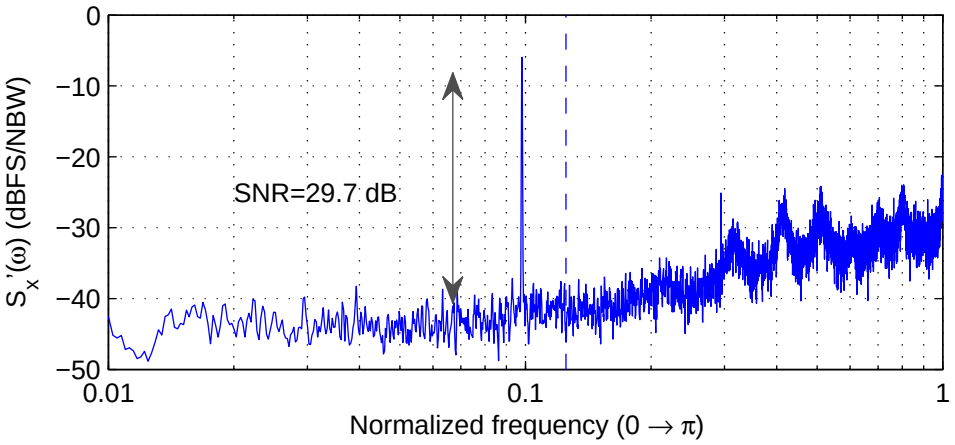


Figure 4.19: Modulation of the signal after interpolation by modelled filter.

5 Decimation

Decimation filter design is related to the design of an interpolation filter. Reverse conversion from single bit DSM to multi-bit PCM, is somewhat simpler and requires only a digital decimation filter as illustrated in figure 5.1. The purpose of the filter is dual; The bitstream is downsampled to the Nyquist rate while expanded to multi-bit PCM words. For this reason the decimation filter is sometimes referred to as a decoder in a Delta Sigma context.

The chapter contains a short introduction to decimation filters in general, before moving to a discussion of decimation filters in a Delta Sigma system in section 5.2. The implemented decimation filter is a third order CIC filter, as presented in section 4.4. The decimation filter is designed for the general case of a bitstream signal processing system. As a result of the cross-correlation operation, some special considerations apply to the decimation. These are discussed in section 5.2.1. The chapter ends by presenting simulation results of the filter.

5.1 Ideal decimation

As with interpolation, the ideal decimation filter is a lowpass brick-wall filter. This filter completely removes all frequencies above the new Nyquist frequency so that no aliasing errors occur. This means that unless the original signal is strictly band-limited to the new Nyquist frequency, information content is lost even for an ideal decimation.

A practical lowpass filter does not fully remove the folding frequencies, but attenuates them to an acceptable level. The specifications of the filter is highly dependent on the required signal quality and the original signal spectrum. Decimation is usually not done unless there is some assumption of the original frequency content. If the signal is known to be band limited, requirements of the filter stop band are lowered as there is no high power content to be removed.

Figure 5.1: Downsampling bitstream to Nyquist rate PCM.

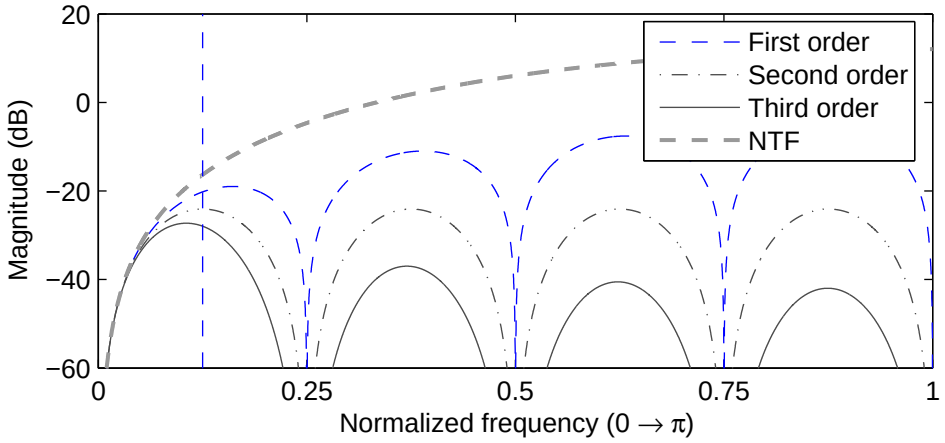


Figure 5.2: Second order Noise Transfer Function (NTF) multiplied by CIC filter functions. Equivalent filter response before downsampling is shown.

As opposed to interpolation, where the signal representation is expanded, representation after decimation is compressed in time. Fewer samples per time results in a lower information capacity per time in the resulting signal representation.

5.1.1 Aliasing

The need for filtering before downsampling is somewhat easier to see than for interpolation. The Nyquist theorem states that at least two samples per period is needed to represent a sinusoid. Downsampling a signal is equivalent to under-sampling high frequency parts of the signal. When a signal is resampled at a rate that does not satisfy the Nyquist criterion, it will only be re-creatable as an alias frequency within the Nyquist band. From a frequency domain perspective, the effect may be described as a folding of frequencies about the new Nyquist frequency $f_N/2$ and into the new signal band.

5.2 Decimation in a Delta Sigma system

The decimation filter in a Delta Sigma coded system plays a double part. Its purpose is to downsample the signal to the Nyquist rate while filtering away quantization noise. At the same time the decimation filter increases the bit width of the signal to binary coded words. A few special considerations must be taken.

Noise level attenuation

The filter must have a stop band attenuation high enough to successfully attenuate the quantization noise of the modulator below the noise floor of the signal band. The noise spectrum of a modulated signal is not flat, but has a frequency characteristic dependent on the DSM transfer function used to modulate the signal. This eases the requirements on the transition band of the filter. The transition band needs to decrease at a rate that matches the steepness of the noise shaping. For the single loop modulators described in section 3.1, this slope is shown to be 20dB per decade for each modulator order. Figure 5.2 shows the NTF of a second order modulator, filtered by CIC filters of the first three orders. The response is shown before downsampling to illustrate the noise power before folding into the signal band. A third order filter must be used to avoid increasing the in-band noise floor after folding.

Word size increase

To increase the number of bits to the desired output word size, the filter must be of a sufficiently high order. Using a very simple filter function may limit the output dynamic range of the signal because the bit width is not increased enough to fully represent the signal resolution.

Using a single CIC filter for decimation of a one bit signal at a low oversampling ratio may result in too low bit growth to increase the output resolution to the desired level. Using equation 4.11 for $OSR = 8$ and $M = 1$ shows that the bit-growth is 3, 6 and 9 for the first three filter orders. With the expected modulator ENOB of 6.4, at least a second order filter is needed to output words wide enough to avoid a limitation in resolution. In the general case, the modulator ENOB grows with an increase in OSR at a slower rate than the CIC filter bit-growth per rate factor R. In other words, this is not a practical problem as long as the above frequency response demands of the filter are satisfied.

5.2.1 Decimation of cross-correlated signal

In the implemented system, the decimation filter works on the output from the cross-correlation block. This signal processing block increases the bit-width from a single bit to ten bits, while performing a convolution of two signals. To investigate the expected cross-correlation output spectrum, the algorithm is implemented in MATLAB and tested on two Delta Sigma coded sinusoid input signals.

Results of the cross correlation between two modulated sequences are shown in figure 5.3 on the following page. Input sinusoids are of the same frequency, making the operation the equivalent of an auto-correlation.The figure shows how the shape of the noise is preserved by the operation. Close inspection reveals an increase in SNR from about 25dB to 50dB after correlation. This is as expected from the auto-correlation, which is commonly used

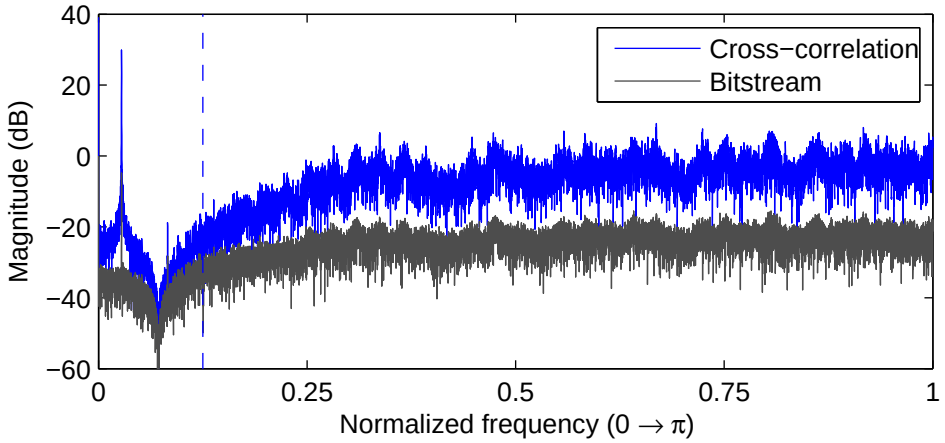


Figure 5.3: Simulated spectrum from bitstream cross-correlator. Input signals are the same frequency. Noise shaping is preserved.

to detect signals buried in noise [30]. Some increase is also seen in the steepness and level of out-of-band noise. This slightly alters the premises of the decimation filter, by increasing its potential effect on the output. A higher stop-band attenuation is required to fully attenuate all noise components below the in-band noise floor. This is not implemented on the chip and is left to future improvements.

Bitstream Cross-Correlation implemented in MATLAB

```
% Computes cross correlation of bitstreams x and template
function[s] = crosscorr(template, x)
N = length(template)

for i = 1:length(x)-N
    % Shift input signal by a single step.
    x_i = x(i:N-1+i)
     % Sum of equal valued bits.
    s(i) = sum(not(xor(template, x_i)))
end
```

5.3 Simulations

A third order CIC decimation filter is modelled in Simulink using the same setup as for the interpolation filters. The downsampling rate is 8. The filter is modelled and simulated to resemble the circuit implementation as closely as possible. Input signals are quantized to 10 bit signed integers. The filter is simulated by restricting each step to a wordsize of 19

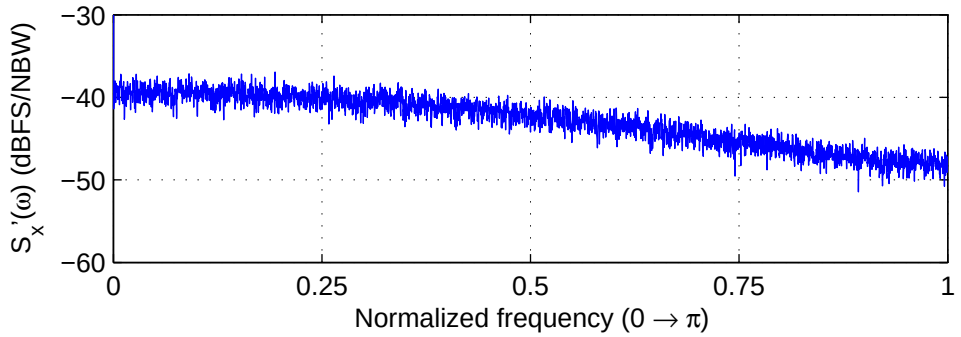


Figure 5.4: PSD of simulated CIC decimation using white noise signal. Illustrates the large signal band roll-off using CIC filters.

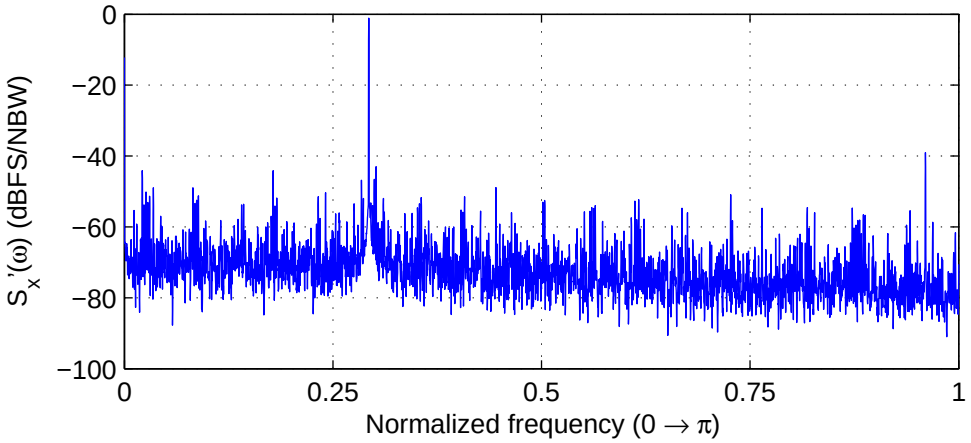


Figure 5.5: Power spectral density of simulated CIC decimation normalized to the Nyquist frequency $f_N/2$. Rightmost sinusoid is folded into the signal band and attenuated by 40dB.

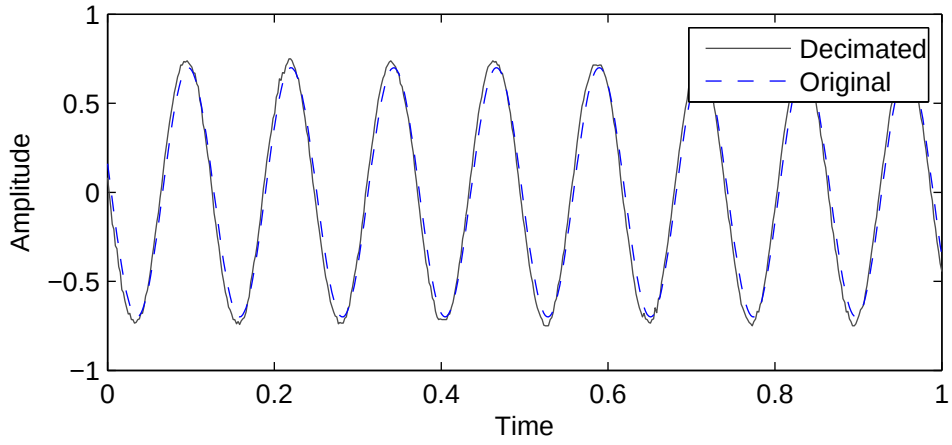


Figure 5.6: Decimated output of modulated signal shows some distortion. $SNR \approx 40$dB.

bits, as used in the circuit implementation. The output is bit-shifted down to 10 significant bits, giving the expected output of the chip decimator.

Figure 5.5 on the previous page shows the estimated PSD of both the input and the output to the filter. The input signal is a sum of two sines, each with an amplitude of 0.5 times full scale, such that the full range is used without overflow. The lowest frequency is set well inside the signal band, and should be unaltered by the filter, while the high frequency component lies outside the signal band, at 0.37π, close to the first maxima in the stop band. It is clear from the figure how the high frequency signal component folds into the signal band after downsampling, and is attenuated by about 40dB.

In-band attenuation is shown in 5.4 on the preceding page. Signal frequencies close to the Nyquist frequency are reduced by $5 - 10$dB, or about half the input amplitude. The simulation is done on the decimation filter, but because this is a result of the CIC filter characteristic this also applies to the interpolation filter.

6 Circuit implementation

The system is implemented in a 90nm CMOS process on a 1mm^2 microchip. It is divided into five main blocks and additional control circuitry:

FIR interpolation filter An order 18 half-band filter with an upsampling rate of two.

CIC interpolation filter A third order CIC filter with unit delays and selectable upsampling rate of either four or eight.

Delta Sigma modulator A second order delta-sigma modulator performing conversion to a bitstream representation.

Cross-correlator Partially asynchronous implementation of efficient cross-correlation on bitstreams.

CIC decimation filter A third order CIC filter with unit delays and a downsampling rate of eight.

Control circuitry Clock division, signal path multiplexers and serial I/O interface.

The cross-correlator block is designed and tested by Olav Liseth as part of his master thesis. It utilizes an interesting asynchronous implementation of bubble sorting to count the number of equal valued bits in two bitstreams [20].

The modulator, three filters and control circuitry is designed as part of this thesis and will be further detailed here. These blocks may be broken down into a few basic components, presented in section 6.1. These are designed and simulated separately before combined to make up the main blocks. Design is done to allow for testing of each individual block or of the system as a whole. Multiplexers (MUXs) in the signal paths as shown in figure 6.3 enables selection of various combination of paths by configurating the MUX control signals. Section 6.2 discusses layout of the chip using Cadence and its extension language SKILL. The last section 6.3 discusses the interface as implemented on chip.

6.1 Basic blocks

All system components are built using a few cells from a standard CMOS library provided by STMicroelectronics. Transistors are counted from the schematics of these cells and given in table 6.1 to provide a basis for chip area estimation.

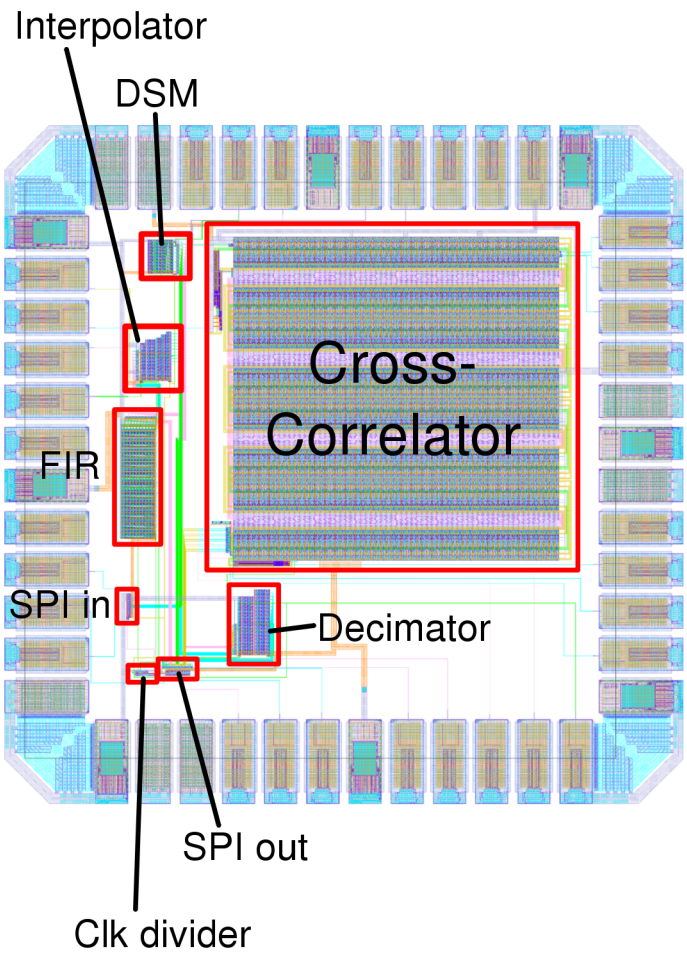


Figure 6.1: Chip layout. Size is 1mm × 1mm including pad frame.

6.1.1 Adders

The adders are implemented as straight forward ripple carry adders. A quick simulation shows that the ripple delay of such an adder is about 700ps for 8 bits of ripple. Even for a full CIC accumulator step of up to 19 bits in word size, this is only a few nanoseconds of delay. Given the low oversampling rate of 8 and a target Nyquist frequency in the audible range, a full period at the high rate is still over 2.8µs. There is good time to perform summation.

The bit encoding required by the CIC filters is *two's complement*. This does not change the operation of addition, but subtraction must be handled in a certain way. Generally, an adder circuit may implement both addition and subtraction by using control logic to make

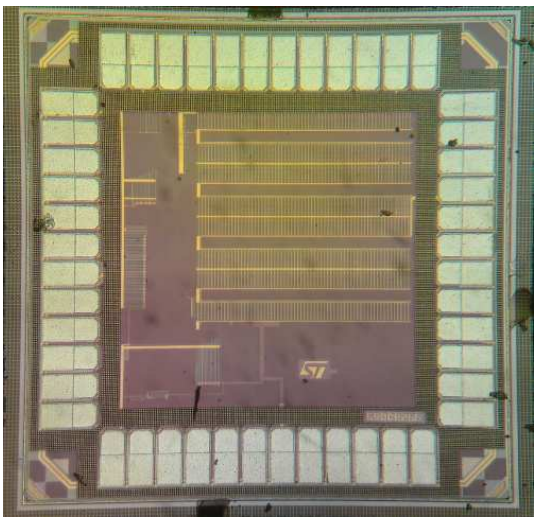


Figure 6.2: Micro photography of 1mm × 1mm chip die. Only the top power lines are visible.

Function	Number of transistors	Cell name
Full adder	28	FA1SVTX
Flip flop /w clear	32	FD2QSVTX
Flip flop /w clear, invert	32	FD2QNSVTX
Inverter	2	IVSVTX

Table 6.1: Standard cells and required number of transistors used in the main system components.

the choice between the two. All the mathematical operations in the system are used in implementing non-changing filtering functions and are constant. It is convenient to use blocks performing only addition or subtraction thereby eliminating extra logic needed to select between them. A two's complement subtractor is made by inverting all input bits of the number to be subtracted, forming the *one's complement*. Adding a value of one forms the two's complement and is done simply by hard wiring the carry-in port of the subtractor to logic one. The two numbers may then be added as normal.

6.1.2 Registers

The registers in the system needs to be non-transparent. The reason being that both the CIC filters and the DSM contain feed-back loops of registers and adders. Transparent

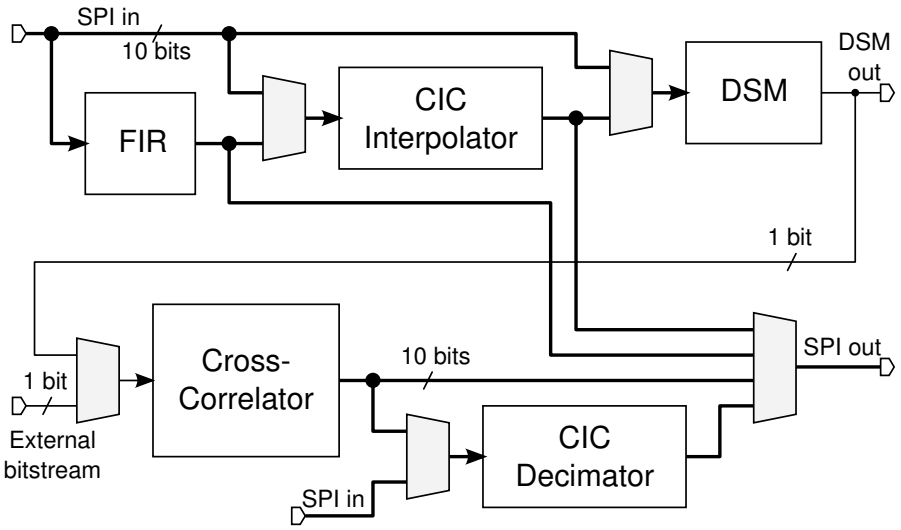


Figure 6.3: Schematic view of system. Only main signal paths are shown.

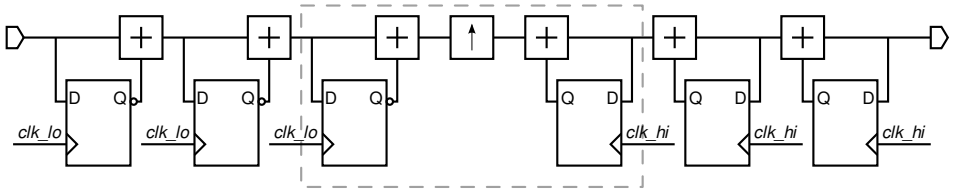
latches would cause these to continuously accumulate incorrect transition values. Registers are implemented as rows of flip-flop circuits, changing on a rising clock edge. For simplicity, the same registers are used in all blocks, although it may be possible to use more effective latches in places without feedback loops. No further analysis is done on whether flip-flop or latch registers are more power-efficient.

Global clear signal

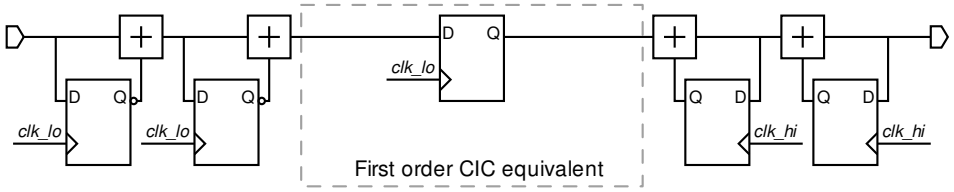
Another point of consideration when going from high level simulations to a circuit implementation is the assumption of a zero initial value. On power-up, hardware registers may contain some random value. When registers are part of a feedback loop as in the CIC filters, this value accumulates on each period of the clock, resulting in an increasingly large error in the output of the filters. This is solved using a global clear signal that resets all registers to a zero initial value.

The problem does not exist in places where the registers are not a part of a feedback loop. Only the first value in each register will be wrong, and correct values will propagate from the input through the registers. Refer to the schematics for CIC filters (Fig. 4.4) and FIR filters (Fig. 4.8) for the difference between registers in a loop and not in a loop, respectively.

The global clear signal is perhaps not the optimal solution; Automatic initialization of problem registers at power-up would have simplified the measurement process somewhat



(a) Upsampling by zero padding.



(b) Hardware efficient upsampling by zero order hold.

Figure 6.4: Equivalency between third order CIC interpolator using zero padding and second order CIC filter using zero order hold.

by not requiring explicit resetting. For testing purposes however, having all registers cleared proved useful in detecting the start of correctly calculated output values.

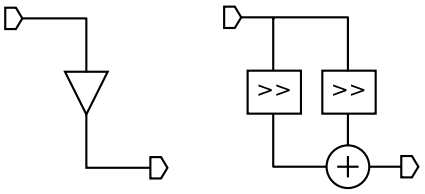
Sample timing in filter structures

Placement of the registers in a filter structure affects the power consumption of the filter. Digital filters are commonly presented either by its *direct form* or its *transpose form*. The transpose form is chosen for implementation of the FIR filter and is illustrated in figure 4.8. On a high abstraction level both representations are equal. For a circuit implementation however, the transpose form avoids congesting all adders in a chain at the output. In terms of dynamic power, such a chain causes a number of unnecessary transitional calculations while the signal propagates through to the output.

The transpose form requires only one addition per sample period, and is better suited for a silicon implementation. The capacitive load on the input is larger, but this is solved simply by inserting a few buffers.

6.1.3 Upsampling

Upsampling is usually implemented by zero padding the signal by the missing number of samples. This preserves the spectrum shape after rate change and is the common way of upsampling in digital signal processing. This may be thought of as a multiplexer switching between the input word and a number of zero value words. This is used as the upsampling step in the FIR filter.



(a) Gain symbol (b) Sum of two hard-wired shift operations

Figure 6.5: Quantized gain steps implemented by bit shift operations.

An alternative upsampler is used in the CIC interpolation filter. Recognizing the first order CIC filter as having a zero order hold response leads to a simplification of the structure. Swapping a single filter step for a hold function amounts to the same result. This will interpolate by copying existing samples at the higher rate, hence operating as a zero order interpolation as discussed in section 4.1. In combination with a CIC filter, this response is desired and increases the filter order by one.

Implementation of this upsampling in hardware is also simpler than for zero padded upsampling, as no extra logic is needed to vary between zero samples and signal samples. In other words, only a resampling at the high rate is needed. This is illustrated in figure 6.4. The method is also discussed in [21].

Applying the method to the third order CIC interpolation filter results in an effective fourth order filter at no added cost. Alternatively the order may have been preserved while removing the first and smallest integrator step together with the last and largest comb filter step.

6.1.4 Downsampling

Similar to upsampling, this may be implemented using two registers where one is operating at the high sample rate and the other at the lower sample rate. Simplifications may be done, such as removing the high rate register if the signal is already at stable values synchronized with the high clock rate.

6.1.5 Gain steps

Gain steps are required in the transfer functions of both the DSM and the FIR filter. The ideal gain step requires a full precision multiplication of the input value by a given constant value. An estimate of multiplier size is given in [18]. This is a high-speed multiplier and might not be the optimal solution for use in this system. It provides an estimate of required multiplier size. For a full ten by ten bits multiplication, the estimate results in 2800 transistors per multiplier unit.

Coefficient	Full value	Quantized value	Absolute Error
b_1	0.0230	$2^{-6} + 2^{-7}$	0.000437
b_3	-0.0404	$-2^{-5} - 2^{-7}$	0.00135
b_5	0.07864	$2^{-4} + 2^{-6}$	0.000520
b_7	-0.161	$-2^{-3} - 2^{-5}$	0.00430
b_9	0.531	$2^{-1} + 2^{-5}$	0.000403
b_{10}	0.844	$1 - 2^{-3}$	0.0311
b_{11}	0.531	$2^{-1} + 2^{-5}$	0.000403
b_{13}	-0.161	$-2^{-3} - 2^{-5}$	0.00430
b_{15}	0.0786	$2^{-4} + 2^{-6}$	0.000520
b_{17}	-0.0404	$-2^{-5} - 2^{-7}$	0.00135
b_{19}	0.0230	$2^{-6} + 2^{-7}$	0.000437

Table 6.2: Coefficient values of the FIR half-band filter.

Simplifications of these gain steps are possible. Delta Sigma modulators commonly use single bit feedback and requires gain steps only on the feedback signal. This enables an implementation of constant gain using multiplexers. The single bit value switches between the gain factor and zero. The chosen error-feedback DSM architecture uses multi-bit feedback, and requires more complex circuits for its gain steps.

By sacrificing precision of the filter coefficients, the gain steps may be significantly simplified. Each coefficient is approximated by a sum consisting of either multiples of two or divisions by a multiple of two.

$$b_n = 2^{b_{n1}} + 2^{b_{n2}}$$

Where b_1 and b_2 are positive or negative integers. This way, the quantized gain steps are implementable as a sum of bit shift operations on the input signal as shown in figure 6.5. While constant bit shift operations are trivial to perform by hard-wiring, each additional term beyond the first requires an additional adder. In other words, precision is traded for hardware area. Two ten bit full adders are required per step, resulting in a number of 560 transistors, or 0.2 times that of the estimated multiplier.

All gain steps in both the FIR filter and the DSM are built from one adder and two hard-wired bit shift operations. As seen from table 6.2, individual quantization errors from this method are small. A comparative simulation showing the effect of gain step quantization for the FIR filter may be seen in figure 4.15 on page 60. Although the quantization reduces filter quality by about 10dB of stop band attenuation, the required number of circuit elements are significantly lowered from that of a full precision filter.

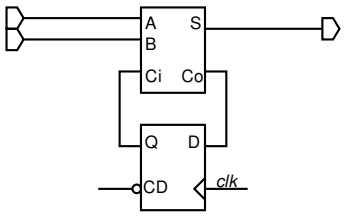


Figure 6.6: Serial adder circuit. Carry is stored from one bit to the next. Additional logic is necessary for clearing carry between words.

Optimization of quantized coefficients

The choice of using quantized filter coefficients has been done without extensive investigation of its effects. Later simulations have shown that FIR filter stop band attenuation is reduced by about 10dB from the desired filter specifications. A better solution may have been found by looking further into the trade-offs between multiplier complexity and additional filter coefficients.

Secondly, the calculation of coefficients is done manually by rounding the full precision values to the nearest sum of powers-of-two. This method is found to be commonly used, and forms the basis of a more efficient method involving translation into Canonical Signed Digit (CSD) coefficients [34, 10, 12]. Software is also available for calculating optimized filter coefficients while taking this quantization method into account [24]. Using such software may result in better overall filter characteristics than the plain rounding used here

6.1.6 Serial arithmetic

A suggested alternative implementation of registers and adders is by serial arithmetic. Although the serial interface to the chip has been a result of the large number of pads needed for test functions, it has also lead to considering an interesting alternative solution. The parallelization of the PCM signal between the input and output serial form and the modulated serial form seems somewhat unnecessary. Although parallel words are quite intuitive and easy to work with, there may be something to gain in terms of simplicity and area by maintaining the input signal in its serial form. Serial arithmetic is possible by working directly on the serial PCM signal.

As the design procedure is similar for all three filters and the DSM encoder, it is enough to target any one of them when considering the trade-offs in using serial approach.

Serial registers

The use of a serial approach to arithmetic operations suggests a higher degree of pipelining, with the ability of starting calculations on a word before the previous step is fully finished. This is only partially true for digital filter applications, because of the intentional use of delays in calculations. The sample delay requirements of the filter is the same independently of a parallel or serial implementation. Hence the minimum number of registers will still, generally, be given by the filter order multiplied by the word size. Pipelining seems to have more use in simplifying multiplications [1, 34].

A power estimate for the registers can be made for serial arithmetic. Taking the increased clock rate into account this results in about ten times that of the parallel arithmetic.

Serial addition

A serial ripple-carry adder is made from a single adder unit and a single bit register storing the carry signal from one clock period to the next. (Fig.6.6.) The single complicating element as compared to the parallel implementation is that the word sizes must be kept track of to clear the carry bit between words. In other words, the clocking scheme still includes both the input bitrate and the input word rate.

Serial addition requires clocking at the bitrate instead of the word rate. Although number of adder elements are reduced to one per word instead of one per bit, all elements are clocked at the faster rate. A parallel implementation of an N bit wide ripple carry adder requires N full adders at the word rate f_N. A serial implementation will instead require one full adder and one register, both clocked at a rate of Nf_N. Using a bit width of $N = 10$, the estimated dynamic power dissipation for the both adders is

$$P_{serial} \propto 10 f_N (28 + 32) = 500 f_N$$
$$P_{parallel} \propto f_N (28 \times 10) = 280 f_N$$

Because of the increased clock rate, serial addition does not have much to offer in terms of dynamic power. However, the gains will be larger if static power is also accounted for because of the significant reductions in chip area.

Serial shift operations

Shift operations may be implemented serially as an increase or reduction in length of shift register delay chains. In a long delay chain such as is used in a FIR filter, this offers the possibility of combining registers used for shift operations with registers already used by the delay line. This is done by tapping the delay line at points corresponding to the shift value. An example of this method is used in implementation of an asynchronous bit serial FIR filter. By this method a total of 481 single bit registers is reduced to 460, a reduction by less than 5% [28]. This is due to the fact that the main sample delay line

still needs to retain its original length. Compared to the hard wired bit shifting used in the parallel structure, the result is the same: One additional adder is needed per term in the coefficients.

In conclusion, the costs seem to be higher than the gains for a serial implementation of the filter blocks. Serial signal representation may have benefits in higher-end systems and for smaller scale chip technologies as it addresses problems such as synchronization of parallel lines, difficult interconnect routing and high static power dissipation. For low speed systems where dynamic power is dominant, a parallel representation seems to be more effective.

6.2 Layout using SKILL

Layout of the chip is done using scripts written in SKILL, a Cadence extension language. The method has proven very useful in generating the large and highly regular structures that make up the digital filters.

SKILL is an interpreted language based on LISP. To meet half-way with designers and CAD-engineers, who are commonly more accustomed with C than LISP, the language provides both C-style and LISP-style syntactic constructs [5]. A few peculiarities still remain. The two most notable are the way whitespace is used instead of punctuation and scope handling; all variables are global unless explicitly declared otherwise. These are obvious sources of much confusion, for someone used to working with C-style languages.

All layout of the system is based around a standard cell library provided by STMicroelectronics. The SKILL scripts builds multi-bit registers and adders from single standard cells, and connects them to form filter structures.

An interesting side effect of using scripted layout is that the process may be done in parallel with filter design. Final specifications for filters in the system depends on the total layout size of the cross-correlator block; Total register length of the correlator decides the bit-width used for chip interconnections and in filters. Benefits are naturally limited for a one-man project, but it may have some use when applied to a design team. Using SKILL makes layout a much more flexible process.

Parametrized CIC filters

Some of the incentive to use an automatic layout procedure is the low number of parameters and high regularity of the CIC filters. This makes it suitable for layout using functions taking the R, M and N parameters directly. Listing 6.2 on the facing page outlines the parametrized function for generating layout for the CIC integrator block. A more complete example of this function is included in appendix B. The scripts calculate word sizes for each step from the equations 4.10 and 4.11.

Pseudo code for the SKILL script generating a CIC interpolation filter

```
procedure(generate_CIC_interpolator(num_bits R M N output_lib)
    for(i 1 2*N
        wordsizes = cons(bitgrowth_at_step(i R M N) wordsizes)
    )
    foreach(num_bits wordsizes
        when(step < N generateCstep_And_CtoCConnections(num_bits M))
        when(step == N generateCstep_And_CtoIConnections(num_bits M))
        when(step > N generateIstep_And_ItoIConnections(num_bits))
    )
)
```

FIR filter layout

The main challenge in creating a layout for the FIR filter is implementation of filter coefficients. This is solved in SKILL by using functions to generate interconnections that are bit-shifted according to a given parameter.

The structural approach to FIR filter layout also makes for easy generation of the DSM layout. A high degree of code is reused directly from the FIR filter layout scripts, either by function calls or small modifications to the code. Due to both experience and code reuse, the full DSM layout is done in a matter of hours.

Improvements

The SKILL scripts are developed only with the described system in mind. Several layout design rules have been obeyed using poor programming practices such as global variables and hard-coded constants. Further work is possible to create generally valid scripts for generation of filter layout. FIR filters commonly require a large number of filter taps, with every tap needing some degree of individual layout. Scripting the layout process allows for fast and simple generation of filters for SoC applications. There are also benefits in extending this scripting approach to generation of the schematics to be used for LVS verification and simulation. An example of this is found in [20].

6.3 Control and interface

The interface to the chip is intended to be as simple as possible. All configuration signals, i.e global clear, mux control signals, and control signals for the cross-correlator, are considered not time-critical. Each of these are input through a separate pad and are not latched and not synchronized with the system clock. This requires the measuring equipment or microcontroller to always keep these nodes driven to the correct level.

6.3.1 Clock division

A system containing several multi rate filters requires some attention to the clock signals. The clocking scheme is summed up in table 6.3. It is centered around the fastest sample rate internal to the system $f_s = OSR \times f_N$. This is the rate of the bitstream and is the only clock rate used for the correlation block. It provides a basis for the lower rate clocks required in the interpolation and decimation. A simple counter divides down by four and eight.

The clock source is provided externally by either the measuring device or a micro-controller. This clock runs at a rate ten times the internal clock rate and synchronizes the SPI interface circuits. The global clear signal is necessary to synchronize all clock signals' phases before valid measurements may be done.

Clock name	Relative rate	Usage in circuit
SPI_clk	$10f_s$	External clock; SPI in/out
clk_hi	f_s	Bitstream rate; Cross-correlator, DSM, CIC filters
clk_div4	$f_s/4$	FIR and CIC interpolation filters
clk_div8	$f_s/8$	Nyquist rate; All filters

Table 6.3: Clock signals used throughout the system.

The clocking scheme is somewhat artificial to provide for testability. The SPI must be able to transmit full words at the oversampling frequency for individual testing of the filters and modulator, where PCM samples need transfer at the oversampling rate. In the imagined event of a fully functional system where testability is no longer as important, the SPI clock may instead be based on the lowest internal system clock, the Nyquist rate f_N. If SPI word length is set equal to the OSR, the clocking scheme would be further simplified by allowing bitstream signals to be synchronized directly with the clock source. The power savings of such a reduction are not very large, as only the small SPI circuits operate at the fastest rate.

6.3.2 Serial Peripheral Interface Bus (SPI)

Due to the limited number of I/O pads on the chip, the 10-bit input and output signals are transferred in serial form over two lines, one in each direction. An SPI inspired solution using shift registers are used on chip. This is a simplified form of an SPI, intended to be compatible with the SPI interface on a micro-controller. The controller is set up as master with the chip as the only slave unit. Hence the SPI functionality of being able to switch between several slave elements is left out to save complexity on chip.

To decrease the probability of bit errors and clocking errors, Schmitt trigger buffers are used at both the SPI input pin and the SPI clock pin.

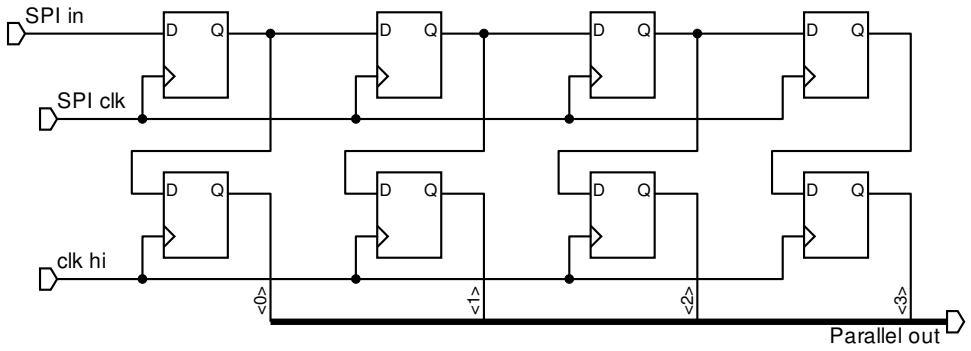


Figure 6.7: SPI in registers. Only four bits are shown.

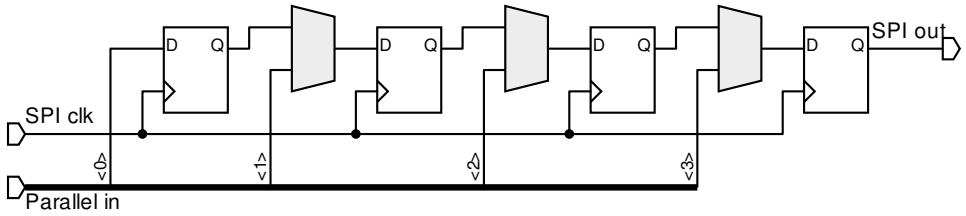


Figure 6.8: SPI out register. Only four bits are shown. Muxes are controlled synchronously with the internal clock rate f_s.

SPI input

Each word is input in series to a shift register synchronized with the SPI clock. The shift register is then sampled in parallel by another register at the highest internal system clock. This is shown in figure 6.7. No means of error detection or separation between words are used in the SPI implementation. This means that a single clocking error will inevitably cause the serial word sequence to become out of synch with the internal system clock, rendering the system useless until reset by the global clear signal.

SPI output

Parallel sample words are clocked into a shift register at the internal system clock rate f_s. The shift register then outputs words in serial form over the SPI output line synchronized with the SPI clock as shown in figure 6.8. Multiplexers connect the parallel input to the register for the duration of one SPI clock period at the beginning of each system clock cycle.

Synchronization is a potential problem in this block as well. There is no form of separation

between words in the output bit sequence, and it is left to the measurement device to split the sequence back into words. The only way of synchronizing the system after an error is to reset all registers, including the clock division circuits, using the global clear signal.

6.3.3 Interconnection and routing

Due to the low target operating frequency of the system, no considerations have been given to interconnection timing. The lack of output signal buffering beyond minimum transistor sizes did cause some concern, but no problems.

The overall word size of 10 bits is somewhat difficult to handle during chip input and readout. Both measurement setups requires software or firmware translation to 8-bit byte sizes due to this choice. More attention to the available external equipment at an early stage would likely have resulted in different solutions, using an interface of either 8 or 16 bits in word size.

The optimal solution in terms of power, is to further reduce word sizes within the chip. Simulated results have shown that the effective resolution of the filters and modulator blocks are in the range of $5 - 7$ bits. This implies that further hardware simplifications are possible by reducing word-sizes in filter steps.

7 Chip measurements

The chip is tested using a custom made circuit board described in section 7.1. Two different measurement setups for digital readout are used, and both are discussed under 7.2. Results are presented in section 7.3. The DSM and FIR filter circuits performs as intended, while a design fault in both CIC filters has rendered both useless. This is explained in 7.3.3. All measurements of the Cross-correlator DSP block is documented by Liseth [20]. Power measurements are not done, but simulated results and their impact on system evaluation are discussed in section 1.2

7.1 Printed Circuit Board (PCB)

The Thin Quad Flat Pack (TQFP) packaged chip is mounted on a circuit board for connection of measuring equipment. All logic and signal routing in the system is done on the chip itself. This reduces the requirements for the card down to distribution of power supply lines and analog conditioning of signals for input and readout by external equipment. Voltage level shifts are needed between the 1.0V signal levels of the chip and the 5.0V or 3.3V signal levels of connected measuring equipment. The card also assembles wires into a 40 pin flat cable for easy handling.

Logic level shifting

Electrical levels of the digital device used for measurements are 0.0V and 5.0V, which necessitates logic level shifting between the chip and the device. This is solved differently on the inputs and the outputs to the chip.

Input level shifting is done using a series of passive voltage clamps intended for GTL-type logic levels. These are able to convert either way, between any two voltage levels from

Device	Logic family	V_{dd}	V_{OL}	V_{IH}
DAQm	TTL	5.0V	$0.0 - 0.35$V	$1.5 - 5.0$V
ATMega32	CMOS	$2.7 - 5.0$V	$0.0 - 0.7$V	$1.6 - 5.5$V
Chip	CMOS	1.0V	$0.0 - 0.5$V	$0.5 - 1.0$V

Table 7.1: Typical voltage levels for the devices.

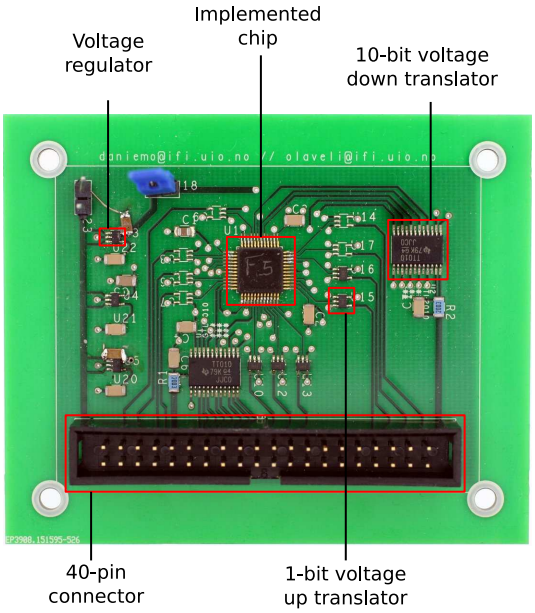


Figure 7.1: Four layer PCB for measurements. Chip in center, with analog level translators and capacitors spread out.

1.0 to 5.0 volts, but they are pass-transistor based and does not provide any signal drive. Chip outputs are not properly buffered for driving a large load. For this reason it seemed somewhat risky to use the pass-transistor solution on the chip's outputs.

The output logic levels are translated using an active circuit able to translate to any level from 1.0V to 3.6V, as decided by two supply voltage levels. The measuring equipment uses Transistor-Transistor Logic (TTL) logic levels, meaning that a voltage higher than about 1.5V will be regarded as a logic high level. Thus it is not necessary for the level shifter to be able to drive a signal value to 5.0V. 3.3V is regarded a good margin. This kind of circuit was also considered for the input level translation, but no version was found having more than a 4.6V guaranteed maximum input voltage tolerance.

Both logic level shift circuits are chosen to keep the option of replacing the data acquisition device with a 3.3V micro-controller. The voltage clamp inputs are able to translate between either 1.0V and 5.0V or 1.0V and 3.3V, without any changes to the circuit board.

Power supply

Three regulators are used for voltage supplies to the chip. The on-chip supply nets are divided in two, to enable current measurements and separate shutdown of the correlator circuit. Because of the correlator block's somewhat analog nature, it is considered less The

correlator supply net is powered by a separate 1.0V static regulator, while the rest of the chip and the external level shift packages are powered by another. An additional regulator at 3.3V is used for the output level shifters. All regulators are in turn powered by a 5.0V supply line and a ground plane connected directly to the data acquisition device.

7.2 Measurement setup

Measurements of the chip are done using two different setups. The simplest solution was seemingly to use an available Data Acquisition (DAQ) module from National Instruments, a device intended for multi-channel digital measurements. It turned out to be more complicated than anticipated, so a solution using a micro-controller and a test board is highly preferred.

Setup 1: National Instruments DAQ

Initial measurements of the chip are done using a DAQ device able to do buffered read and write on a number of digital lines in parallel. Unfortunately device drivers and software support for MATLAB is unavailable. Instead, the device C Application Programming Interface (API) has been integrated in a Python wrapper script. The script configurates the device for use with the PCB and chip. Details beyond simple read, write and initialization functions are hidden in the back-end, allowing for a higher-level interface to the device.

Test signal generation, post-processing and presentation of data are done using the Python scripting/programming language and its Numpy and SciPy expansions. These allow handling of large numerical data sets in a fashion inspired by MATLAB.

SPIserialize

The DAQ device does not handle numbers in the bit-serial form, but has good support for writing multi-bit values in parallel form. The translation is done in software on the connected computer. The initial implementation in Python proved quite slow, so a Python extension module is written in C for serializing Numpy data arrays. (Appendix C.) The compiled module enables fast translation between serial and parallel integers. Functions convert from 10 bit numbers contained in 16 bit integers native to the computer, to sequences of 10 single bit values contained in 8 bit words as handled by the DAQ device. This serialization is necessary to exchange data over the SPI interface to the chip.

Setup 2: STK600 w/ATMega32

The preferred setup is using a micro-controller connected to the PCB. An Atmel STK600 test board with an ATMega32 Micro-controller Unit (MCU) was available for measure-

ments. Using this setup requires a much smaller amount of written code, both for the micro-controller firmware and for the host computer.

The ATMega32 includes an SPI interface in hardware, removing the need for serialization of values. The interface does not have configurable word sizes, meaning that a firmware routine is needed for translation between the 10 bit wide samples on chip and the 8 bit bytes used in the micro-controller SPI unit. A second difficulty is data transmission between the controller and computer. Neither the STK600 or the ATMega MCU have hardware support for Universal Serial Bus (USB) data transfer, so the final solution is to transfer across the serial Universal Synchronous/Asynchronous Receiver/Transmitter (USART) interface.

Olimex header board /w SAM7-256

A third measurement setup using a different MCU is presented by Liseth [20]: Although requiring an additional routing board between the chip PCB and MCU test board, this is the preferred solution. The SAM7-256 micro-controller includes a configurable length SPI interface as well as fast data transmission over USB.

7.3 Results

Chip measurements are presented as spectrograms. These are produced to be comparable to those from the simulations. The Periodogram method is described in section 3.4.1.

7.3.1 DSM

The modulator is characterized using an oversampled digital input sinusoid of same frequency as in the simulated model. Figure 7.2 shows the power density spectrum calculated from the output bit stream, and results are very similar to that of the simulated modulator (Fig 3.12). This is not surprising due to its deterministic nature as a fully digital system. No deviating behaviour from that of the model is found.

7.3.2 FIR filter

The chip FIR filter response is shown in figure 7.3. The input signal consists of a sum of two sinusoids of the same frequency and magnitude as used for the simulated models. After upsampling and filtering, the stop band alias signals are attenuated by about 30dB. This is the same results as predicted by the simulation shown in figure 4.14. A comparison of the measured noise signal response (fig. 7.4) to the noise signal response simulated on the quantized coefficients model (fig. 4.15), reveals no apparent differences. This is not surprising, as the simulation model takes into account the quantization of coefficients by using shift operations. The only difference is that the simulation model handles samples

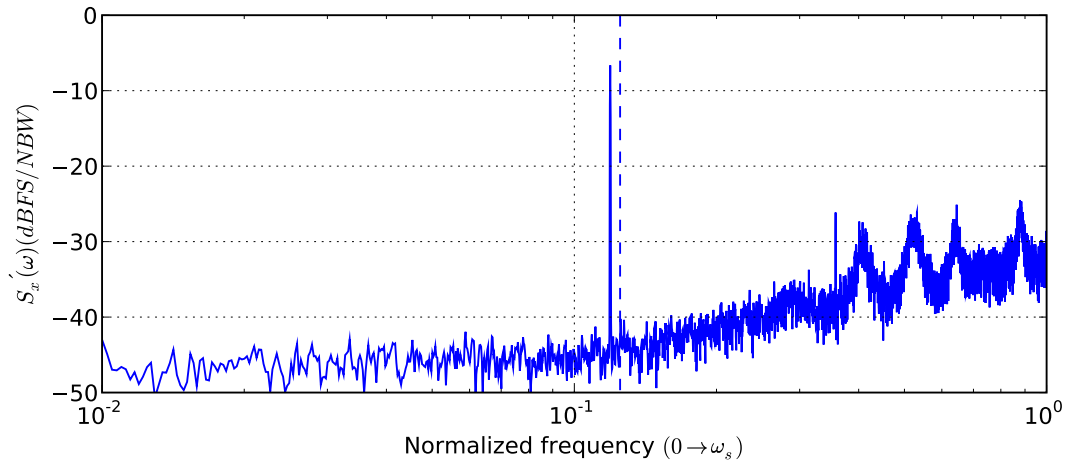


Figure 7.2: Chip measurement of Delta Sigma Modulator (DSM). Signal bandwidth indicated by dashed line.

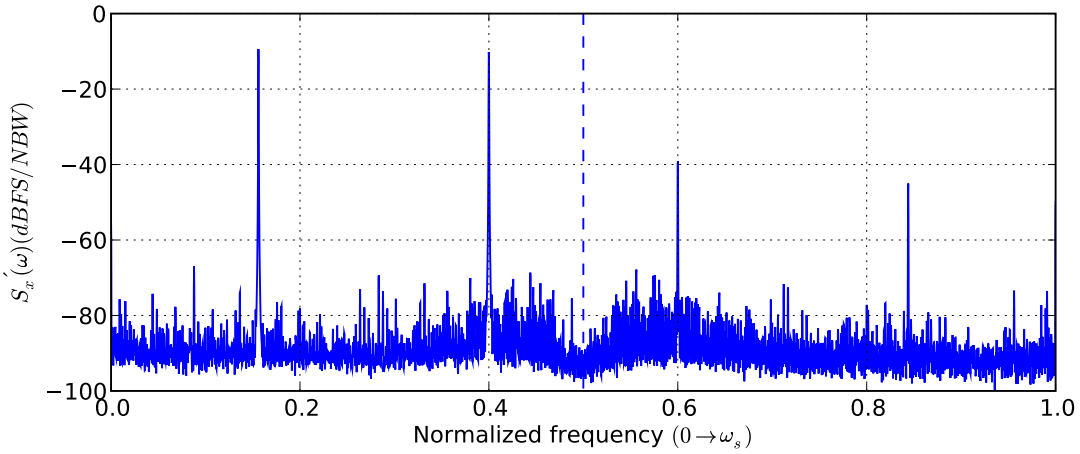


Figure 7.3: Chip measurement of FIR Half-band interpolation filter.

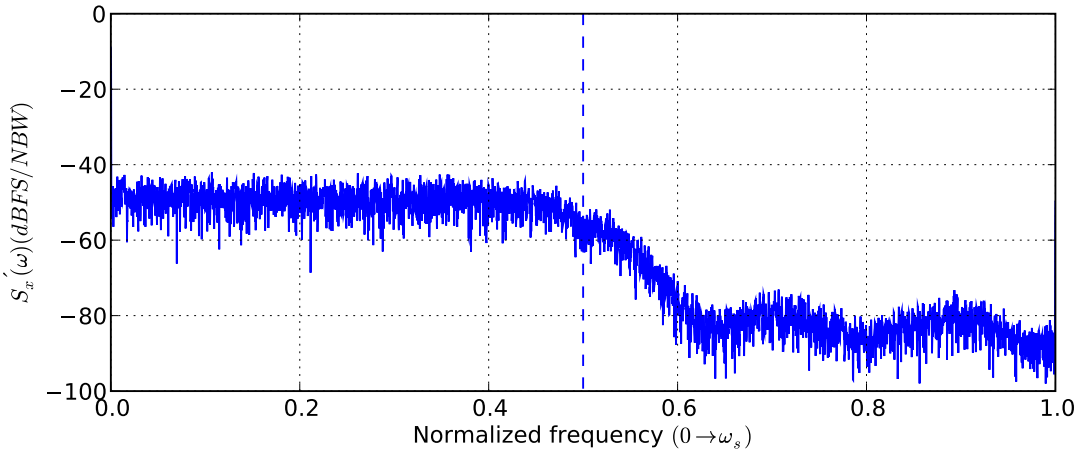


Figure 7.4: Chip measurement of FIR Half-band interpolation filter using white noise input. Nyquist frequency indicated by dashed line.

using 16 bit precision without truncating to 10 bits at every internal step, possibly resulting in minor truncation errors throughout the filter.

7.3.3 Faulty CIC filter design

Both CIC filters are tested using simple sinusoids as input. The time domain results in figures 7.5 and 7.6 show clearly how the filters are not working correctly. By careful investigation of the graphs, it may be seen how the filters are internally overflowing resulting in a wrap-around behaviour. This is seen as a change from near maximum to a near minimum value in a single sample period. The continuous and periodic parts of the output signal signifies that internal filter states are stable. Due to a design fault, neither of the CIC filters work as intended in their chip implementation. The problem lies in the implementation of the resampling step of both filters.

High level filter design

Simulation and design of the filters have been done using a high level approach with MATLAB and Simulink as tools. After design verification in Simulink, only a few circuit level simulations have been done in the Cadence environment. The analog nature of the Spectre simulator makes simulations on a fully digital circuit very time consuming. Circuit level simulations revealed errors due to initial states in the integrator registers as mentioned in 6.1.2. No suitable simulation was set up to reveal problems with resampling steps.

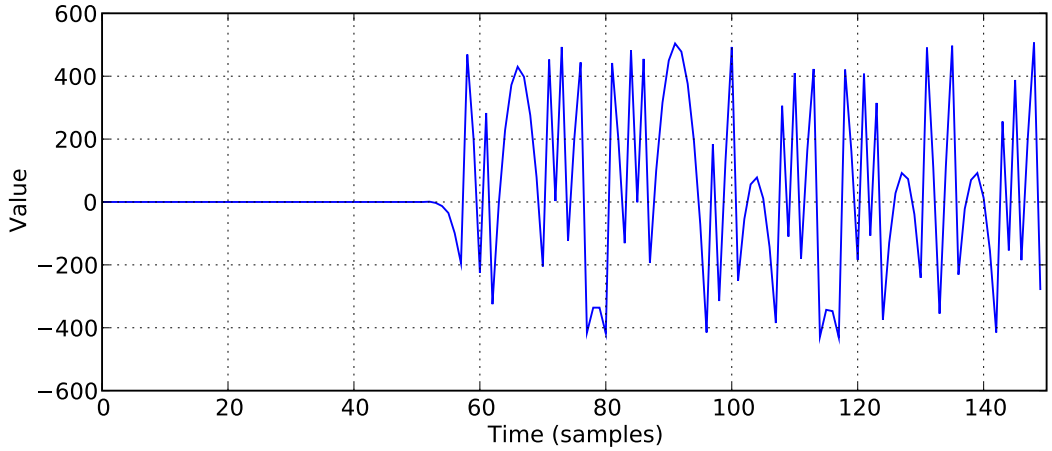


Figure 7.5: Time domain plot of output from flawed CIC interpolator.

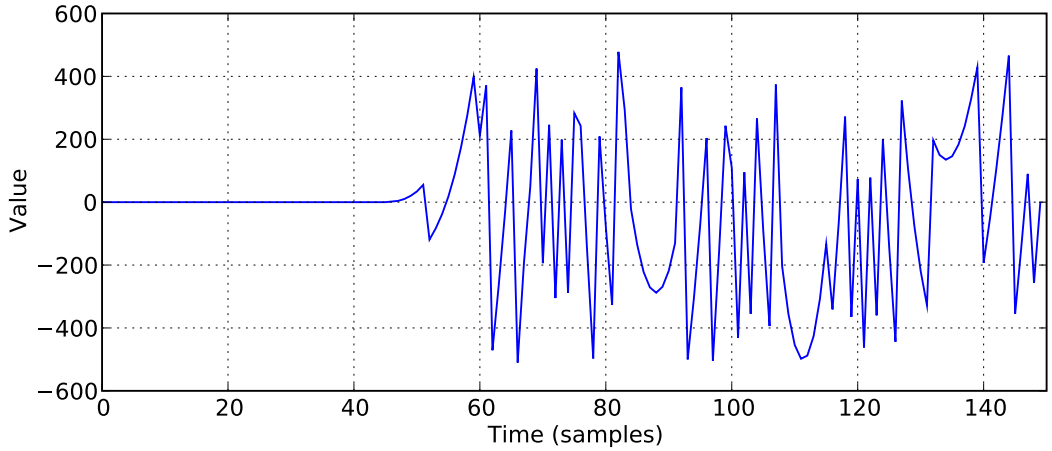
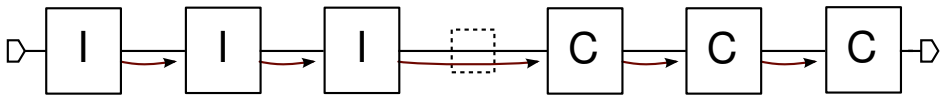
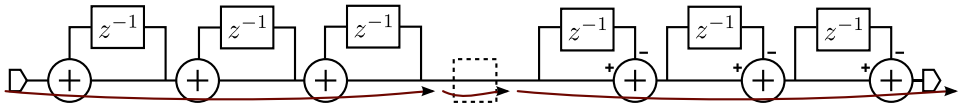


Figure 7.6: Time domain plot of output from flawed CIC decimator.



(a) Block level schematic obscures actual signal path.



(b) Signal path directly from input to output.

Figure 7.7: CIC abstraction levels. High level approach led to incorrect assumptions of the sample timing in the filter steps.

Cause

As discussed under section 6.1.3 and 6.1.4, resampling may be simplified in both filters. Due to the high level approach, this is not done correctly during translation from block schematic to a circuit level schematic, resulting in the resampling step being completely left out in both filters. Each comb and integrator step is built separately and parametrized using SKILL for easy combination into either an integrator or a decimator filter. The handling on a block level led attention away from the fact that there is a signal path not separated by a register, through each block. This is illustrated in figure 7.7.

During layout blocks have been incorrectly handled as if they were sample-time separated by a register at each output. (Fig.7.7a.) With this assumption, both upsampling and downsampling is done simply by clocking the registers in each filter section at either the high or low rate. However, as seen in figure 7.7b, there is a signal path through each block that is not latched by a register. The result is that both the final CIC filter implementations have signal feed-through from input to output during each sample period. This causes incorrect behaviour of the filters.

Decimation filter

The high sampling rate integrator sections of the decimator operates as intended. The differences occur in the low rate comb filter sections. The low rate registers sample at the low rate, while all adders see an input signal from the accumulators changing at the high sample rate. This means the adders are working at the high sampling rate, which in turn causes the filter output to change at the high rate.

Fortunately, internal state of the filter is decided by the registers, which are working correctly. Under certain circumstances, this means that the first output sample of each clock period is correctly calculated. Adding a downsampling step at the output will allow only the correct values to be output, by discarding seven intermediary samples at the output

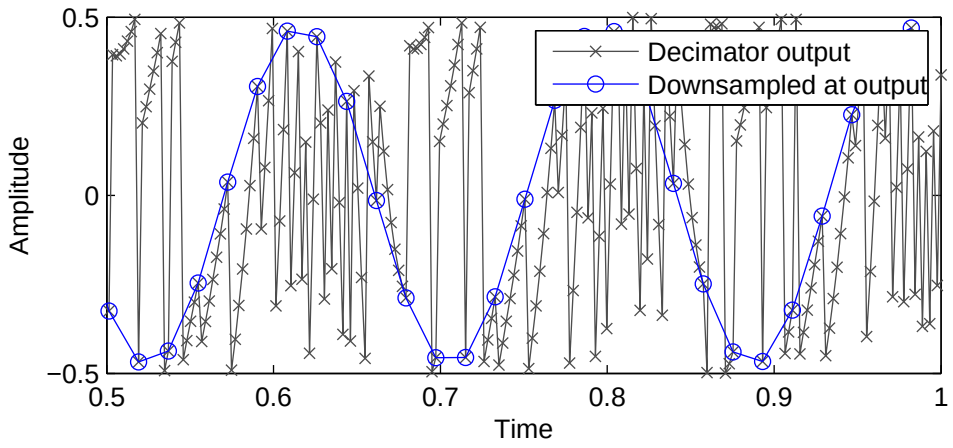


Figure 7.8: Circuit simulation of the decimation filter. Downsampling is done at the output, showing how every 8 output samples is correctly calculated.

instead of in the middle of the circuit.

This suggests a fix to the problem. A downsampling may be implemented quite simply in software, either on the micro-controller or in the Python scripts performing readout and post-processing.

A single requirement is that the two clock rates of the filter are synchronized in such a way that comb step registers sample at a time when adders output a correct value, and not one of the seven intermediate incorrect values. This is not the case with the implemented clock division circuit. For this reason, the software work-around is not possible. This is confirmed in simulation, by setting the filter up with correctly phased clock signals. Figure 7.8 shows the output of the decimation filter, together with the output downsampled to the low rate after filter output.

Interpolation filter

The same reasoning also applies to the interpolation filter. The low rate comb steps of the filter are unaffected by the missing resample step. The integrator step registers will perform the upsampling, and if the two clock rates are correctly phased the filter will perform exactly as intended. This has also been verified from simulations using clock signals synchronized at the rising clock edges.

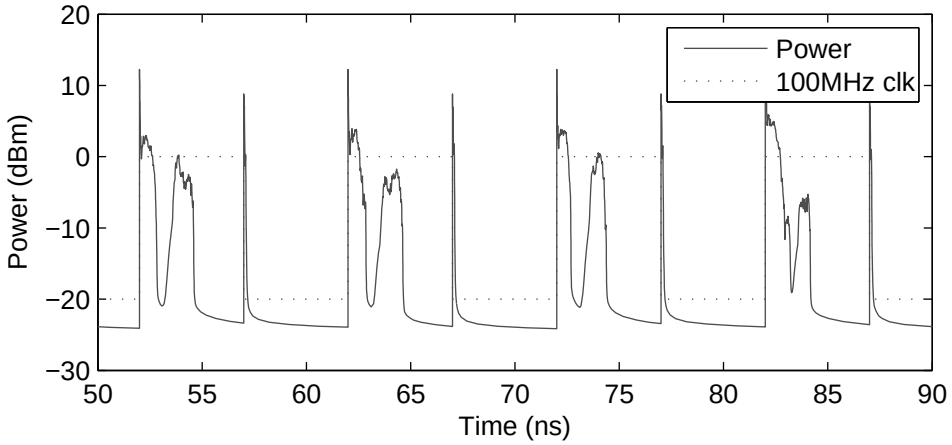


Figure 7.9: Simulated power consumption for CIC decimation filter clocked at 100MHz. Values in dB referenced to 1mW.

7.4 Power simulations

The PCB has not been designed for measurements of power consumption for the chip as a whole. The correlation part of the chip has a separate supply voltage net to allow measurements of this part. The same is not done for the filters and DSM. All filter blocks are instead supplied using a single 1.0V power net. The PCB uses this same power network to supply all logic level shifter packages, hence making current measurements of the chip difficult.

This may have been solved differently. Keeping the main chip power network separate from surrounding packages would have enabled current measurements for the chip as a whole. This could have been with a jumper, in the same manner as is done for the cross-correlator power supply line.

Individual measurements of each block would require the ability to shut down individual blocks in the system, or supplying each block on an individual net. The current implementation uses neither clock gating nor individual supply nets to facilitate such a shutdown.

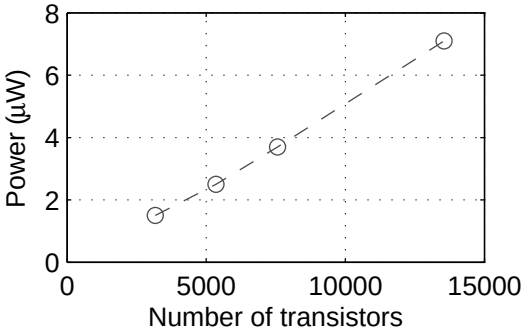
Simulation setup

Power consumption is simulated using the Spectre analog environment from Cadence. The CIC decimator is simulated using a 100MHz clock rate. Although much faster than designed for, this gives a good view of power dissipation in the adder circuits at each clock pulse. The resulting waveform is shown in figure 7.9.

As expected from a synchronous circuit, power dissipation is concentrated at each rising clock pulse. The last clock period in the figure is a rising edge for both the high and the

Circuit	P_{static}	P_{total}
DSM	1.5µW	1.5µW
FIR filter	7.1µW	7.1µW
CIC decimator	3.7µW	3.7µW
CIC interpolator	2.5µW	2.5µW
Total	15µW	15µW
Cross-correlator	650µW	730µW

(a) Power per block



(b) Power vs number of transistors

Table 7.2: Simulated power dissipation for a clock rate $f_s = 18.8$kHz. Dynamic power is negligible at this rate.

low rate filter clock, thus consuming more power. The difference is not larger due to the missing downsampling step, causing all adders to work at the high rate.

Power peaks occur at the rate of the clock and have instantaneous values of about 1000 times the static power dissipation. During adder activity, the power lies around $0.1-1.0$mW. The figure also displays some of the signal-dependency of power consumption. Each clock period result in calculations on different values across the circuit, leading to differences in consumed power.

Average power over time for the filter at 100MHz is calculated to 0.23mW. From the graph, static power dissipation is estimated at -24dBm $= 4$µW.

Static power vs. dynamic power

For comparison with the cross correlation measurements done by Liseth, the main system blocks are simulated using the same clock rate of 18.8kHz [20]. Input signal is a high amplitude sinusoid which is assumed to cause a realistic activity level on the input bits. Both CIC filters are measured with the correct resampling steps inserted. Figure 7.2b shows simulated power and transistor counts for each of the four filter blocks. The linearity indicates that the power approximations discussed in section 1.2 are valid for these four blocks.

Static power is calculated at moments where internal states are stable, while total power is calculated from average power over several clock periods. As noted from table 7.2a, the dynamic power dissipation at such a low operating frequency is very small compared to the static power dissipation. The value is smaller than variations in static power, and is only estimated to lie around $0.01 - 0.1\%$ of the static power. This result is somewhat surprising and heavily emphasizes the importance of reduction in hardware area as a means

of reducing power consumption for the 90nm implementation.

One important extrapolation of this result is that the cost of increasing the overall clock rate in the system is small, up to a certain point. Simulations done using a higher system clock rate of 160kHz still shows static power accounting for more than 95% of the total power, hence no significant increase from the results at 18.8kHz.

This corresponds with results measured on the cross-correlator block. At 18.8kHz the cross-correlator is measured to consume 0.74mW, where 0.65mW is due to static leakage currents. The dependency on operating frequency is higher in this block, due to a higher internal activity level resulting from each pulse of the clock. As a result, dynamic power makes up 57% at 150kHz [20].

Serial arithmetic revisited

For applications requiring a low operating frequency, the above simulations have shown that static power is dominant. This result increases the significance of hardware re-use as a means for lowering overall power. This is seen by considering the 10 bit serial adder as discussed earlier in section 6.1.6, while accounting for static power dissipation. Normalized to the sampling rate, the power estimate may be re-stated as

$$\begin{aligned} P_{total} &= P_{static} + P_{dynamic} \\ P_{serial} &= (28+32)\alpha + 10(28+32)\beta \\ P_{parallel} &= (28\times 10)\alpha + (28\times 10)\beta \end{aligned}$$

where α and β are proportionality constants for static and dynamic power. For an operating speed no higher than 160kHz, the above simulation results have shown that the ratio of dynamic power to static power is lower than 0.05. This gives $\beta < 0.05\alpha$.

$$\begin{aligned} P_{serial} &= 60\alpha + 600\times 0.05\alpha = 90\alpha \\ P_{parallel} &= 280\alpha + 280\times 0.05\alpha = 294\alpha \end{aligned}$$

Resulting in estimated power savings by a factor of 3 for the serial adders, due to the lower hardware area.

Extending this reasoning to the cross-correlator structure brings to mind a modification of the initial Figure of Merit (FOM) for estimating power consumption, as presented in section 2.2 on page 16. For operation in a low dynamic power environment, the frequency dependency is reduced or even neglected. Hence a reduction of the dependency of the cross-correlator power consumption on the OSR parameter. This has two significant implications. First, the advantage of the bitstream cross-correlator over the compared multi-bit correlation architecture will be larger than the values predicted by figure 2.2 on page 17. Secondly, a decreased dependency of power on OSR allows for significant savings also for higher values than $OSR = 8$. This in turn, allows more freedom and a higher resulting signal quality of the DSM.

8 Conclusion

The thesis has presented an implementation of a single-chip cross-correlator. This is done by recoding a digital signal into its Delta Sigma bitstream representation using a digital-to-digital modulator. To facilitate for such a conversion, the signal is interpolated prior to modulation. After signal processing, a decimation filter downsamples the filter back to a Nyquist rate representation for readout and post-processing.

A study of the cross-correlator show very promising results with regards to power efficiency and performance. The main drawback of the cross-correlator is that savings in power consumption have to be traded with Oversampling Ratio (OSR). Together with the requirement of a single-bit Delta Sigma code, a low OSR is directly conflicting with encoding of a high quality bitstream signal. Single-bit modulators operating under an OSR as low as 8 have been found to be limited upwards to about 35dB of SNR. The implemented modulator performed according to these predictions, resulting in a measured SNR of 31dB. The complexity of such a modulator is not overwhelming; an implementation in 90nm is done in $40 \times 52\mu m$. Nevertheless, the modulator remains as the single constraint on the overall signal quality of the system.

As discussed in conclusion of chapter 7, the implemented system may be improved by consideration of static power dissipation as well as dynamic. For the intended application fields in detection and processing of real-world processes, static power is shown to be the main source of power consumption. This is found both from circuit simulations on the synchronous digital filters and from measured results on the partially asynchronous cross-correlator [20]. All measures taken to reduce power in the system are concentrated around dynamic consumption.

A further study is required into the implications of static power in the system. Preliminary results from assuming a neglectable dynamic power dissipation indicate that the advantage of bitstream cross-correlation over compared binary correlation is significantly increased. Of particular interest is the relationship between oversampling ratio and static vs. dynamic power dissipation. Static power domination may be the case for typical ubiquitous applications, requiring a low operating speed and utilizing a small featured manufacturing process.

8.1 Future work

Both Delta Sigma modulation and digital filtering are very large fields of research with a vast amount of publications presenting effective and specialized methods. Only a few of these are investigated as part of this work. Many more are left to future improvements.

If principles of bitstream signal processing are to be extended to higher speed systems, dynamic power is expected to be much larger than in the presented results. An interesting direction for further exploration is in asynchronous design methods. This is only touched upon in looking for ways to improve the system power dissipation by thinking in terms of clock-less design. Benefits of asynchronous design includes closer connection between cell activity and actual performance, thus saving power in a high speed system. An asynchronous full adder for dual-rail data-path coding may be implemented with only 34 transistors, compared to 28 in the cell library used in this project [23] Asynchronous implementation of DSMs has also been successfully done [9].

Power saving methods on other levels than the algorithmic and architectural have not been looked into in this thesis. Due to the high static power consumption in the system, substituting for high threshold transistors may be a good way of reducing overall power consumption.

Only a single DSP system utilizing bitstream calculations is looked into in this work. Others are mentioned, but will be left to further investigations.

Another improvement of the system may be found by exploring the way the correlation template is created. Because of the close connection between cross-correlation and filtering, pre-processing of the correlator template may allow mild compensation filters to be included directly in the template, hence altering hardware filter requirements.

The SKILL implementation provides a good way to quickly adapt the system to changing requirements. Future work on the SKILL implementation is possible to not only allow fully parameterized generation of CIC filters, but also FIR filters. A script for generation of FIR filter layout masks is of obvious value in many applications.

A Paper

The enclosed paper is submitted for the 27th NORCHIP Conference. Acceptance status is not known at the time of writing.

Power efficient Cross-correlation using Bitstreams

Olav E. Liseth, Daniel Mo, Håkon A. Hjortland, Tor Sverre "Bassen" Lande and Dag T. Wisland
Dept. of Informatics, University of Oslo, Norway
Email: olaveli@ifi.uio.no, daniemo@student.matnat.uio.no, haakoh@ifi.uio.no, bassen@ifi.uio.no, dagwis@ifi.uio.no

***Abstract*—The fundamental operation of cross-correlating signals is viable in a number of signal processing applications. In typical pattern-matching applications, cross-correlation is desirable. In this paper we present a power efficient implementation of a time-domain cross-correlator suitable for integration in CMOS. Bitstream coding of both data and template simplify multiplication operations. Measured performance of a CMOS implementation in 90 nm technology is reported.**

I. Introduction

A major trend in microelectronics is integration of specialized solutions in an increasing number of applications. The idea of ubiquitous computing is certainly exciting, but at the same time demanding. Both sensing and controlling real world processes demand mixed-mode solutions combined with challenging signal conditioning and processing. The notion of small, portable, battery-operated systems often organized in a wireless sensor network (WSN) has initiated significant research activity. In these applications size is limited and low-power operation is mandatory for battery operation. The benefit of adopting specialized silicon systems is evident in applications like WSN motes [1].

An overall characteristic of these ubiquitous computational devices is mixed-mode operation. Sensing of external states is accomplished with analog-to-digital converters (ADCs) and controlling of external processes requires DACs. A popular and power-efficient converter architecture is sigma-delta, or delta-sigma, converters well suited for integration in digital technology.

A recurring signal processing task in real-world signal analysis is pattern-matching recovering special features of some sensed signal. As an example in this work we will use signal processing for Electrocardiogram (ECG) classification. Although filter-based solutions have been developed with great sophistication over the last decades [2], recent publications [3] indicate that cross-correlation based methods are preferable. Furthermore, the miniaturization of ECG-monitoring devices are pursued both in research and in industry [4]. The notion of heartbeat detectors embedded in the ECG electrode and configured in a wireless sensor network is tempting and would enable long-term ECG analysis. Substitution of the quite clumsy Holter monitors used today would make life easier.

In this work we will show how power-efficient cross correlators are implementable in standard CMOS technology, exploring bitstream-coded signals. The internal signal representation called *bitstream* is found in sigma-delta converters, but is usually decimated to Nyquist rate binary coded numbers. However, some signal processing like filtering using bitstreams are reported [5], [6], [7], [8]. Another application of bitstreams is found in the audio coding format named Direct-Stream-Digital used in SACD, developed by Sony and Philips [9].

The idea of cross-correlation using bitstreams was proposed in [10] and evaluated for heart rate variability study in [11]. In this paper we present an implementation of a complete cross-correlation chip with a binary coded interface.

II. Bitstream Cross-correlator

A discrete estimate of the cross-correlation of two sequences is found by the equation:

$$r(t) = \sum_{k=0}^{n-1} y(k)x(t+k)$$

The finite, time-variant sequence $x(t)$ of length n is cross-correlated with a template sequence, $y(t)$, by multiplying each element of the two sequences over a window of length n and summing the result. The result, $r(t)$, is a good estimate of the cross-correlation between the two finite sequences. For every new sample of the incoming signal another $r(t)$ may be computed creating another element of a cross-correlation sequence between the incoming signal and the template.

From a computational perspective we need to do n multiplications and sum the results for each sample of the incoming signal. We either need n multipliers running in parallel or to speed up the clock with a factor of n. Then we need to figure out an efficient summing operation in the simplest form requiring another n iterations following the multiplications. No wonder alternative pattern-matching measures are sought when power is precious.

In this paper we explore the idea proposed in [10] of implementing cross-correlation by processing bitstreams. By doing so, quite power efficient single chip heartbeat detectors embedded in the ECG electrode are feasible.

III. Single-chip Cross-correlator

To allow for a simple interface to the chip, both input and output signals are assumed to be Nyquist rate binary encoded. Conversion to and from oversampled bitstream representation is done on-chip. Fig. 1 shows how the bitstream is only necessary internal to the system. However, a Serial Peripheral Interface Bus (SPI) is used for interfacing, and on-chip multiplexers facilitate testing of individual blocks.

A. Bitstream Conversion

The binary-to-bitstream signal conversion is done using a two step interpolation filter and a sigma-delta modulator. The interpolation filter upsamples an input signal at the Nyquist rate by the oversampling ratio required by the modulator. Interpolation in several steps is commonly used. This allows for a combination of an anti-alias filter, with a narrow transition band, and a more hardware efficient Cascaded Integrator Comb (CIC) filter [12]. This is a compromise between filter quality and silicon area, which in turn affects power consumption.

Similarly, a decimation function is required for removing the high-frequency quantization noise present in bitstream coded signals, shown as the CIC decimator in Fig. 1.

Using a low oversampling ratio (OSR) when modulating the bitstream obtains the largest power savings [10]. As a consequence, the possible signal quality of the bitstream is limited. Compromising between these two considerations, the system OSR is set to 8. Still the bit sequence may be of significant length depending on the time-span of the correlation window. In our test-chip we use a correlation length of 1024 bits.

B. Bitstream Operations

Multiplication between bitstreams has a significant advantage compared to its multibit counterpart and can be carried out using basic logic gates. The dynamic range of the implemented modulator is normalized to $-1 \leq x \leq 1$, where x is the input. The probability that the output of the modulator, X, is 1 or 0 is then given by $P(X = 1) = (1 + x)/2$ or $P(X = 0) = 1 - P(X = 1) = (1 - x)/2$. The modulation of two input signals x and y is regarded as uncorrelated and results in two bitstreams, X and Y. The XNOR of the two bitstreams results in:

$$\begin{aligned} P(X\overline{\oplus}Y = 1) =& P(X = 0) \cdot P(Y = 0) \\ &+ P(X = 1) \cdot P(Y = 1) \\ =& \frac{1}{2}(1 + xy) \end{aligned}$$

which indicates that operations usually requiring complex digital circuitry can be done with a simple logic gate when processing bitstreams. The results from bitstream multiplications are just a single bit from each multiplier. A summing operation still remains.

For power efficiency we try to avoid clocks exceeding the oversampled clock frequency. Knowing the multiplier results are all single bits, the summing is reduced to counting the number of '1' after multiplications. A novel asynchronous solution is explored to compute the sum during the same clock cycle as the multiplication. The counting operation is split in two operations:

1) Sorting bit sequence from multipliers. The sorting method used is inspired by the software bubble sort algorithm, but is implemented completely asynchronously.
2) Encoding sorted result as binary number. By interpreting the sorted bit sequence as a thermometer coded result, a binary number is encoded.

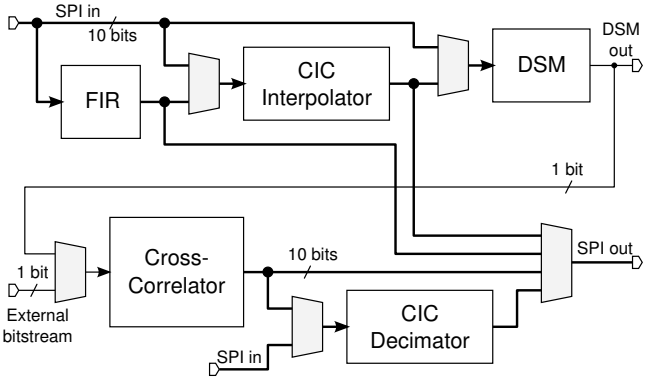


Fig. 1. Block schematic showing main signal paths. The multiplexers allow each block to be tested individually. Only the SPI in and out pins are used for the full signal path.

The asynchronous operation is achieved using inherent gate delays explained below.

C. Bitstream Cross-correlation

During setup phase, the binary-to-bitstream converter may be used or the template may be shifted in directly as a bitstream coded sequence of up to 1024 bits. Then the incoming signal is shifted into the correlation register coded as a bitstream or converted by the binary-to-bitstream converter. All bits in the two registers are "multiplied" by XNOR gates at the start of every clock cycle. The bubble register is loaded with the results from the XNOR operation after an adequate delay. Then the "bubbling" is started and the rest of the clock cycle is reserved for the asynchronous sorting operation followed by latching of results. During the next clock cycle the thermometer coded result of the bubbling is converted to a binary representation in parallel with computation of the following correlation result.

IV. Implementation

In these dedicated systems, register lengths are hard-coded by design. Depending on application, correlation window lengths must be adapted for minimal power consumption. For easy generation of different cross-correlators we have used SKILL, a LISP-like CAD system extension language. The produced SKILL scripts facilitate fast and easy generation of both schematics and layout of cross-correlators of different sizes.

A. Bitstream Cross-correlator

The bubble register used for sorting is shown in Fig. 2 and each element consists of one ordinary RS-latch, one AND gate and inverters used as delay-elements. The latches are loaded with the result from the bitstream multiplication in the beginning of the clock cycle. The bubble sorting is initiated after a predefined delay for proper settling of the latches.

It is important to notice that the exchange of bits in the bubble register is local, enabling parallel operation. Basically,

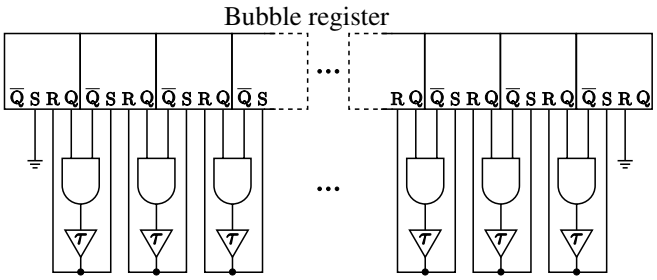


Fig. 2. Asynchronous bubble sorter.

the operation $(1,0) \rightarrow (0,1)$ can be carried out, provided data is stable. To ensure failsafe operation the actual exchange operation is delayed using some inverters. In this way all '1' are "bubbled" to the right while the '0' is "bubbled" to the left.

The final and stable condition of the bubble register has all '1's stacked to the right and all '0's to the left. This code is known as a thermometer code. In fact there is one and only one $(0,1)$ sequence in the sorted result, which may be used for unique binary encoding.

The thermometer code is converted to its binary representation in the following clock cycle, while the next cross-correlation result is computed. The thermometer encoded result is fed to the binary encoder assuming a single $(0,1)$ transition. This transition is identified by a row decoder, comparing two consecutive values from the thermometer coded array using an XOR-gate. The selected row pulls down the correct precharged output lines which encodes the desired binary result. The output lines are sampled after a predefined delay and clocked out through the SPI-interface.

B. Data Conditioning

Hiding of bitstream coding is an integral part of the cross-correlator chip. The following modules are included:

1) Interpolation filter: The anti-aliasing step of the interpolation filter is a 20-tap halfband FIR filter following an upsampling by two. This filter type has every other coefficient set to zero and is an efficient way to achieve a narrow transition band around 0.5π. Each tap in the filter is in turn simplified. Gain steps are quantized such that each are realizable as a sum of maximum two hardwired bit-shift operations. This removes the requirements for full multiplications in the filter.

The CIC filter step is of third order and performs an upsampling by four, resulting in the desired oversampling rate.

2) Sigma-Delta modulator: The modulator has a second order error feedback structure. This structure uses multibit-bit feedback and is well suited for digital input. The most important design constraint for the modulator is the requirement that the output is a bitstream. This rules out good modulator types such as cascaded or MASH architectures giving word streams. Quality of the modulator is also limited by the low OSR in the system. For an OSR of 8, modulator orders of two and more give a maximum expected signal to quantization noise ratio (SQNR) of 35–40 dB.

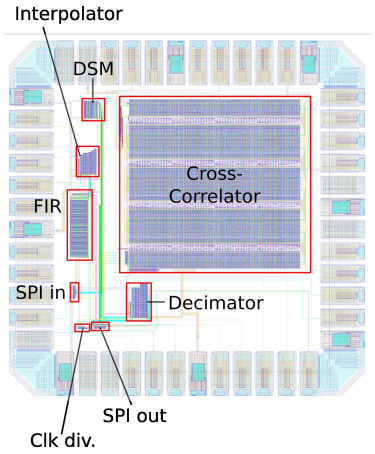


Fig. 3. Chip layout designed in STMicroelectronics 90 nm technology, delta-sigma converter and 1024-bits cross-correlator. Chip size included pads is 1×1mm

3) Decimation filter: The filter used for decimation is an ordinary CIC decimation filter of third order with a downsampling ratio of 8. This converts the oversampled output from the cross-correlation block back to a Nyquist rate signal, while reducing high frequency noise inherited from the bitstream representation.

V. Measurement Results

The chip is measured using a simple microcontroller enabling measurements of the different chip modules.

1) Interpolation filter: The frequency response of the cascaded interpolation filter is shown in Fig. 4. The response of the FIR filter step was confirmed by chip measurements, using both white noise input signals and sinusoidal inputs. Total stop band attenuation is 35–40 dB, close to expected performance.

2) Sigma-Delta modulator.: The chip modulator was tested using a full-scale sinusoidal input. Output spectrum is shown in Fig. 5 and shows the expected SQNR level.

A. System Performance

Here we briefly show the system performance by component-based ECG-analysis [3].

In Fig. 6 the cross-correlation from the test-chip is shown together with cross-correlation using the $xcorr()$-function in MATLAB. The data used is taken from the QT database [13]. The first QRS-wave from the annotated database is selected as template and loaded into the template-register as a bitstream. Then a sequence of three heartbeats are fed to the chip and the cross-correlated result is decimated and plotted. Just by inspection the cross-correlation results are promising and very close to results obtained using "ideal" cross-correlation with MATLAB. A proper evaluation of the cross-correlation chip for QRS-detection demands extensive analysis, far beyond the scope of this paper.

The main computational block doing cross-correlation is measured to 2.1 mW power consumption when active. With a clock frequency of 480 kHz and an OSR of 8 this is equivalent

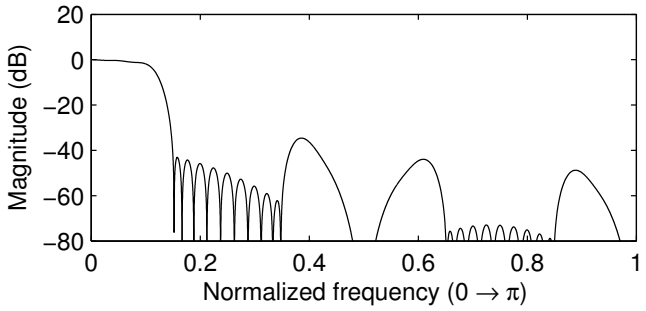


Fig. 4. Frequency response of cascaded interpolation filter. Normalized to half the oversampling frequency.

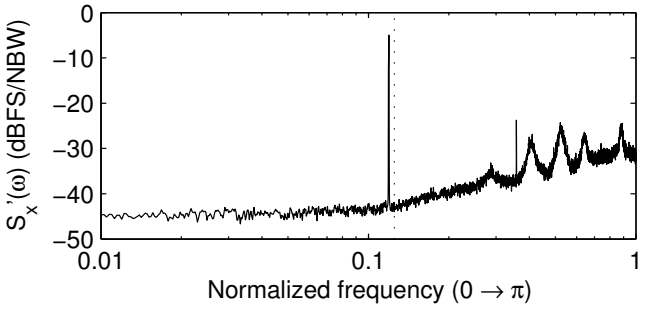


Fig. 5. Power spectral density of the sigma-delta modulator. Nyquist frequency indicated by dotted line.

to $\approx$ 7.7 M multiplications each second at Nyquist rate. In addition this first chip was a proof-of-concept with far from optimal layout. We consider these results to be very promising for low-power, single-chip pattern-recognition.

As indicated in the introduction, cross-correlation is a generic pattern-matching computational element suited for several signal processing tasks. The bitstream processing solution presented in this paper aims at low-power operation at low or moderate signal frequencies. There is a growing demand for low frequency sensor interfacing where filtering is required. The proposed cross-correlator chip may also be used as a programmable filter by time-warping the template for convolution. The cross-correlator chip is suitable both for low-frequency operation and as a programmable filter. Many of these applications are emerging with biomedical applications.

VI. Conclusion

In this paper we have presented a novel single-chip cross-correlator suitable for power-efficient pattern matching. Bitstream encoded signals found in sigma-delta converters are used for efficient multiply-and-sum operations basically substituting multipliers by simple gates.A novel asynchronous bubble-register is utilized avoiding increased clock frequency. The running cross-correlator is implemented in 90 nm technology and measured results are provided.We expect these kind

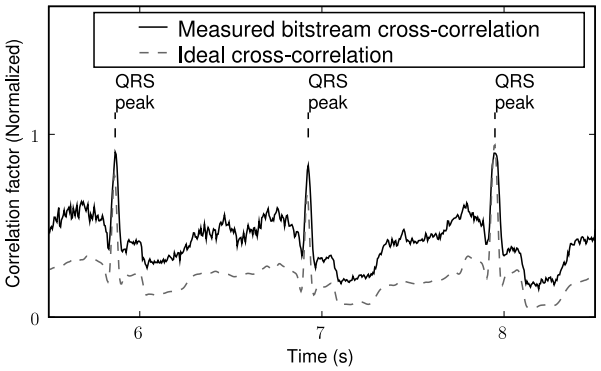


Fig. 6. Bitstream cross-correlation results.

of cross-correlators to be viable in power-limited, low signal frequency applications like QRS-detection of ECG-signals.

Acknowledgment

The authors would like to thank Dept. of Informatics, University of Oslo for providing chip fabrication and lab facilities for this project.

References

[1] B. Warneke, M. Last, B. Liebowitz, and K. Pister, "Smart dust: communicating with a cubic-millimeter computer," *Computer*, vol. 34, no. 1, pp. 44–51, jan 2001.

[2] [Online]. Available: http://www.openecg.org/

[3] T. Last, C. Nugent, and F. Owens, "Multi-component based cross correlation beat detection in electrocardiogram analysis," *BioMedical Engineering OnLine*, vol. 3, no. 1, p. 26, 2004. [Online]. Available: http://www.biomedical-engineering-online.com/content/3/1/26

[4] [Online]. Available: http://www.toumaz.com/

[5] P. O'Leary and F. Maloberti, "Bit stream adder for oversampling coded data," *Electronics Letters*, vol. 26, pp. 1708–1709, Sept 1990.

[6] F. Maloberti and P. O'Leary, "Processing of signals in their oversampled delta-sigma domain," in *Circuits and Systems, 1991. Conference Proceedings, China., 1991 International Conference on*, Jun 1991, pp. 438–441 vol.1.

[7] M. F., "Non conventional signal processing by the use of sigma delta technique: a tutorial introduction," in *Circuits and System, ISCAS '92. Proceedings, 1992 IEEE International Symposium on*, vol. 6, May 1992, pp. 2645–2648.

[8] S. Summerfield, S. Kershaw, and M. Sandler, "Sigma-delta bitstream filtering in vlsi," in *Circuits and Systems, 1994., Proceedings of the 37th Midwest Symposium on*, vol. 2, Aug 1994, pp. 1200–1203 vol.2.

[9] E. Janssen and D. Reefman, "Super-audio cd: an introduction," *Signal Processing Magazine, IEEE*, vol. 20, no. 4, pp. 83–90, July 2003.

[10] T. Lande, T. Constandinou, A. Burdett, and C. Toumazou, "Running cross-correlation using bitstream processing," *Electronics Letters*, vol. 43, no. 22, pp. –, 25 2007.

[11] M. Ho, T. Lande, and C. Toumazou, "Efficient computation of the lf/hf ratio in heart rate variability analysis based on bitstream filtering," in *Biomedical Circuits and Systems Conference*, Nov 2007, pp. 17–20.

[12] E. B. Hogenauer, "An economical class of digital filters for decimation and interpolation," *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, vol. 29, no. 2, pp. 155–162, Apr 1981.

[13] P. Laguna, R. Mark, A. Goldberger, and G. Moody, "A database for the evaluation of algorithms for measurement of qt and other waveform intervals in the ecg," *Computers in Cardiology*, vol. 24, pp. 673–676, 1997.

B SKILL code excerpt

The following listing is a code example showing the interpolator layout routine written in SKILL. The total chip layout is done in about 3000 lines of code, generating complete layout for all filters and the Delta Sigma Modulator (DSM).

CIC.il

```
; Parameterized function for generation of a CIC interpolator layout.
procedure(generate_CIC_interpolator(num_bits R M N output_lib)
    prog((dbcv source_lib cell_name view_name xoff handle steps i j val
        pitchC pitchI)
    ;init constants
    pitchC = 5.32
    pitchI = 4.76
    xspace = 3.92
    source_lib ="CORE90GPSVT"
    cell_name="CIC_interpolator"
    view_name="layout"

    ; calculate word sizes for each C-step
    for(i 1 N
        steps = cons(num_bits+i steps)
    );for
    if(M == 1 then ; special case for last C-step when M=1, see Hogenauer
        steps = cons(num_bits + N-1 cdr(steps))
    );if
    ; and for I-steps
    for(i N+1 2*N
        ; Hogenauer's word size formula.
        val = ( 2**(2*N-i) )*( (R*M)**(i-N) )
        val = val / R
        val = val * N ; My variation of the upsampling.
        ; calculating ceil(log2(val))
        j = 0
        while(val > 2**j
            j++
        )
        steps = cons(num_bits+j steps)
    );for
    steps = reverse(steps)
    printf("Word sizes are: \n")
    foreach(elm steps
```

```
35          printf(" %d" elm)
        )
        printf("\nPlease verify word sizes, just to be sure.")

        generate_adder(output_lib)
40      generate_register(output_lib)

        dbcv=dbOpenCellViewByType(output_lib cell_name view_name
            "maskLayout" "w")

        ; align at x=0
45      xoff=0.06+3.98*M
        if(mod(M 2) == 0 then xoff = xoff + 0.47)
        step=1 ; step counter
        ; inputs with pins and labels
        for(i 0 car(steps)-2
50          let((net_name net pin term text)
                ;input
                sprintf(net_name "In<%d>" i)
                net = dbCreateNet(dbcv net_name)
                term = dbCreateTerm(net nil "input")
55              pin=dbCreatePin(net
                    dbCreateRect(dbcv '("M2" "pin") list(1.485:1.75+i*pitchC
                        2.275:1.89+i*pitchC))
                )
                text = dbCreateTextDisplay(term term '("M2" "pin") t
                    1.84:1.82+i*pitchC
                    "centerRight" "R0" "swedish" 0.7 t nil t nil t "name")
60              text~>parent = pin~>fig
            );let
        );for
        foreach(num_bits steps
            when(step < N
65              ; C-step and C to C type connection
                handle = generate_C(num_bits pitchC M output_lib)
                dbFlattenInst(dbCreateInst(dbcv handle nil xoff:0 "R0") 1
                    nil)
                handle = bus_connection_CC(num_bits pitchC output_lib)
                dbFlattenInst(dbCreateInst(dbcv handle nil xoff:0 "R0") 1
                    nil)
70              handle = bus_extension_CC(nth(step steps)-num_bits pitchC
                    output_lib)
                unless(handle == nil
                    dbFlattenInst(dbCreateInst(dbcv handle nil
                        xoff:(num_bits-1)*pitchC "R0") 1 nil)
                )
                if(step == 1 then
75                  gen_supplies(dbcv pitchC xoff 0 num_bits "C" "L")
                else
```

```
            gen_supplies(dbcv pitchC xoff 0 num_bits "C")
        )
        if(mod(M 2) == 0
            then
            xoff=xoff+4.33+3.98*M ; dist between C-steps
            else
            xoff=xoff+3.21+4.63*M ; dist between C-steps if even M
        )
    )
    when(step == N
        ; C-step and C to I connection
        handle = generate_C(num_bits pitchC M output_lib)
        dbFlattenInst(dbCreateInst(dbcv handle nil xoff:0 "R0") 1
            nil)
        handle = bus_connection_CI(num_bits pitchC pitchI output_lib)
        dbFlattenInst(dbCreateInst(dbcv handle nil xoff:0 "R0") 1
            nil)
        handle = bus_extension_II(nth(step steps)-num_bits pitchI
            output_lib)
        unless(handle == nil
            dbFlattenInst(dbCreateInst(dbcv handle nil
                xoff+0.60:(num_bits-1)*pitchI "R0") 1 nil)
        )
        gen_supplies(dbcv pitchC xoff 0 num_bits "C" "R")
        xoff=xoff+8.44 ; dist between C and I
    )
    when(step > N
        ; I-step
        handle = generate_I(num_bits pitchI output_lib)
        dbFlattenInst(dbCreateInst(dbcv handle nil xoff:0 "R0") 1
            nil)
        if(step < 2*N
        then
            ; I to I connection unless last step.
            handle = bus_connection_II(num_bits pitchI output_lib)
            dbFlattenInst(dbCreateInst(dbcv handle nil xoff:0 "R0") 1
                nil)
            handle = bus_extension_II(nth(step steps)-num_bits pitchI
                output_lib)
            unless(handle == nil
                dbFlattenInst(dbCreateInst(dbcv handle nil
                    xoff:(num_bits-1)*pitchI "R0") 1 nil)
            )
            if(step == N+1 then
                gen_supplies(dbcv pitchI xoff 0 num_bits "I" "L")
            else
                gen_supplies(dbcv pitchI xoff 0 num_bits "I")
            )
        else ;last step, make output pins and labels
```

```
                for(i num_bits-10 num_bits-1
                    let((net_name net pin term text)
                        ;input
                        sprintf(net_name "Out<%d>" i-num_bits+10)
                        net = dbCreateNet(dbcv net_name)
                        term = dbCreateTerm(net nil "output")
                        pin=dbCreatePin(net
                            dbCreateRect(dbcv '("M2" "pin")
                                list(xoff+1.720:1.55+i*pitchI
                                    xoff+2.23:1.69+i*pitchI))
                            )
                        text = dbCreateTextDisplay(term term '("M2"
                            "pin") t xoff+2:1.62+i*pitchI
                            "centerLeft" "R0" "swedish" 0.7 t nil t nil t
                                "name")
                        text~>parent = pin~>fig
                        );let
                    );for
                gen_supplies(dbcv pitchI xoff 0 num_bits "I" "R")
            );if
            xoff=xoff+7.84 ; dist between I steps
        )
        step++; next step
    );foreach
    ;bus extension on first input.
    contact = dbCreateInstByMasterName(dbcv "cmos090" "M3_M2" "symbolic"
        nil
        3.07:1.63+num_bits*pitchC "R0")
    dbCreateProp(contact "row" "int" 2)
    contact = dbCreateInstByMasterName(dbcv "cmos090" "M3_M2" "symbolic"
        nil
        3.07:1.63+(num_bits-1)*pitchC "R0")
    dbCreateProp(contact "row" "int" 2)
    dbCreatePath(dbcv "M3" list(
        3.07:1.375+num_bits*pitchC
        3.07:1.885+(num_bits-1)*pitchC) 0.14)
    ;horizontal supplies
    dbCreateRect(dbcv "M6" list(0:0 xoff-xspace:6))
    dbCreateRect(dbcv "M7" list(0:11*pitchC xoff-xspace:11*pitchC-6))
    dbSave(dbcv)
    return(t)
    );prog
);procedure
```

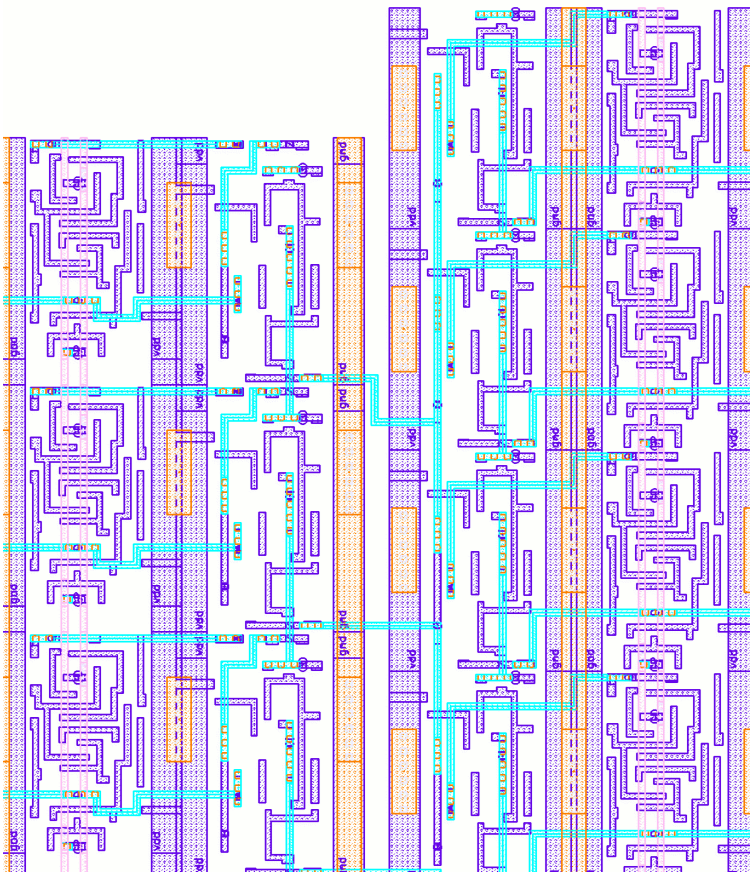


Figure B.1: Detail from CIC interpolator layout as generated by the SKILL code. Left side shows 3 bits of the final comb filter step. Right side shows 4 bits of the first integrator step.

C SPIserialize

Source code for a compiled Python extension module performing conversion between parallel and serial form. Used for interfacing between the chip SPI and the National Instruments DAQ device.

spi.c

```
#include <Python.h>
#include <numpy/arrayobject.h>
#include <stdlib.h>
#define BITS 10
#define MAX_VAL 511
#define MIN_VAL -512

/**
 * Serialization routine - MSB First
 */
static int c_serialize(long* in, char* out, int len){
    int i, j;
    // Conversion loop
    for(i = 0; i < len; i++){
        if((*in < MIN_VAL) || (*in > MAX_VAL)){
            // Raise ValueError
            PyErr_Format(PyExc_ValueError,
                    "Array values must be within range (%d, %d)",
                    MIN_VAL, MAX_VAL);
            return 1;
        }
        // Serialize a word into single bits. MSB first
        for(j = 0; j < BITS; j++){
            out[BITS*i+j] = *in >> (BITS-1-j) & 0x1;
        }
        in++;
    }
    return 0;
}

/**
 * Converts array consisting of samples max BITS wide to array of
 * serialized single bits wrapped in c_uint8. This is the required data
 * in DAQmx calls. The samples are in sequences of length BITS
 */
```

```
static PyObject * serialize(PyObject *dummy, PyObject *args)
{
    /* Parse Numpy array from argument *args */
    PyArrayObject *wordArray = NULL;
    if(!PyArg_ParseTuple(args, "O!", &PyArray_Type, &wordArray))
        return NULL; // Raises exception

    /* Create byteArray 10 times length of input array */
    int len = PyArray_DIM(wordArray, 0);
    npy_intp dims[1];
    dims[0] = BITS*len;
    PyArrayObject* bitArray = (PyArrayObject*)PyArray_SimpleNew(1, dims,
                NPY_BYTE);
    if(bitArray == NULL){ // raise TypeError
        PyErr_Format(PyExc_TypeError,
                "Could not create numpy array from data");
        return NULL;
    }

    int status = c_serialize((long *)PyArray_GETPTR1(wordArray, 0),
            (char *)PyArray_GETPTR1(bitArray, 0), len);
    if(status) return NULL;

    return PyArray_Return(bitArray);
}

/**
 * Conversion back to parallel form - MSB First
 */
static int c_unserialize(char* in, npy_int16* out, int len){
    int i;
    int j;
    // Conversion loop; outer loop traverses shortest array
    for(i = 0; i < len; i++){
        *out = 0;

        // leading sign bits of int must be set.
        if(in[BITS*i]){
            // OR with mask to set all leading sign bits.
            *out |= (npy_int16)-0x400; // ~0011 1111 1111;
        }

        for(j = 0; j < BITS; j++){
            *out |= in[BITS*i+j] << (BITS-j-1);
        }
        out++;
    }
    return 0;
}
```

```
/**
 * Converts a back from a sequence consisting of serialized single
 * bits wrapped in c_uint8 as read from the DAQmx device.
 * Words are assumed to be of length BITS and MSB first. Returns a
 * Numpy Array object of 16 bit samples.
 */
static PyObject * unserialize(PyObject *self, PyObject *args)
{
   /* Parse bitArray from argument *args */
    PyArrayObject *bitArray = NULL;
    if(!PyArg_ParseTuple(args, "O!", &PyArray_Type, &bitArray))
        return NULL; // Raises exception

    /* Create new bytearray w/length 1/BITS of input array */
    int len = PyArray_DIM(bitArray, 0) / BITS;
    npy_intp dims[1];
    dims[0] = len;
    PyArrayObject* wordArray = (PyArrayObject*)PyArray_SimpleNew(1, dims,
        NPY_INT16);

    if(wordArray == NULL){ // raise TypeError
        PyErr_Format(PyExc_TypeError,
                "Could not create numpy array from data");
        return NULL;
    }

    int status = c_unserialize((char *)PyArray_GETPTR1(bitArray, 0),
        (npy_int16 *)PyArray_GETPTR1(wordArray, 0), len);
    if(status) return NULL;

    return PyArray_Return(wordArray);
}

/* Python module information */
static PyMethodDef spiMethods[] = {
    {"serialize",  serialize, METH_VARARGS,
     "Serializes array for SPI interface"},
    {"unserialize",  unserialize, METH_VARARGS,
     "Unserializes array for SPI interface"},
    {NULL, NULL, 0, NULL}        /* Sentinel */
};

PyMODINIT_FUNC
initspi(void)
{
    (void) Py_InitModule("spi", spiMethods);
    import_array();
}
```

D Micro-controller firmware

The following code is source code for the ATMega32 μC firmware, written in C. It provides an interface between the host computer and chip.

mcu.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <inttypes.h>
/* Defines mapping between MCU ports and PCB pins */
#include "mapping.h"

/* Aliases for ports */
#define DDRSPI DDRB
#define SCK PORTB7
#define MISO PORTB6
#define MOSI PORTB5
#define SS PORTB4

#define BAUD (19200)
#define UBRR (F_CPU/16/BAUD - 1)

#define uchar_max (1<<8)
#define mask 0x1F

/* Very simple protocol defining command words for each signal path
 * configuration of the chip.
 */
#define CMD_CICI 1
#define CMD_CICD 2
#define CMD_FIR 3
#define CMD_DSM 4
/* Global variable storing the chosen chip configuration. Must be set
 * as first byte of the transmission.
 */
volatile uint8_t command = 0;

/*
 * Interrupt routine for USART RX complete.  First byte of any
 * transmission is assumed to be a command, initializing the config
```

```
 * bits during  the command loop. Translation between five-bits
 * USART bytes and eight-bit SPI bytes are done using a 16-bit
 * shift register buffer. Bits are shifted in, MSB first, from
 * USART. When more than eight bits are present, these are sent
 * over SPI to the chip, and shifted out of the buffer.
 *
 * Watchdog is also reset on USART recieve.
 */
ISR(USART_RXC_vect)
{
    /* Buffer position counter */
    static uint8_t off = 16;
    static uint16_t buffer;
    /* Do not reset chip while transmissions are incoming in short
     * intervals. */
    wdt_reset();
    /* First byte of the transmission is a command. */
    if (!command){
        command = UDR;
        return;
    }
    /* Shift incoming into buffer*/
    off -= 5;
    buffer &= ~((uint16_t)mask<<off);
    buffer |= ((uint16_t)UDR<<off);
    if (off <= 8){
        /* If buffer contains more than 8 bits of valid data, send
         * across SPI */
        SPDR = buffer >> 8;
        /* Shift sent data out of buffer */
        buffer <<= 8;
        off += 8;
    }
}

/*
 * Interrupt routine triggering on complete SPI transmission of a
 * single 8-bit byte incoming from the chip. A two-byte shift
 * register is used as buffer to split incoming data into five-bit
 * bytes. These are sent to host across UART.  A static int SPIoff
 * keeps track of the postition in the buffer between interrupt
 * calls.
 */
ISR(SPI_STC_vect)
{
    /* Position counter */
    volatile static uint8_t SPIoff = 8;
    static uint16_t buffer;
    /* Read SPI data into buffer */
```

```
        SPIoff += 8;
        buffer <<= 8;
        buffer |= (uint16_t)SPDR;
89      while (SPIoff >= 8){
            /* If buffer contains enough data, send across USART. */
            SPIoff -= 5;
            UDR = buffer >> SPIoff;
        }
94  }

    /*
     * Main loop configurates chip mux control bits for the selected
     * signal path as given by the command byte, which is the first
99   * byte recieved in any transmission. A configuration is set and the
     * rest of the program is interrupt-driven by SPI and USART
     * interrupts.
     */
    void mainLoop(void)
104 {
        while(1){
            /* Command Loop */
            switch (command){
                /* Switch by given command byte.
109              * Applies one CONFIG set for each valid command */
                /* FIR FIlter config set */
                case CMD_FIR:{
                    PORTC = (1<<CFG_OUT_A) |// SPI output mux
                            (0<<CFG_OUT_B) |  // SPI output mux
114                         (1<<CLEAR);
                }break;
                /* CIC Decimate config set */
                case CMD_CICD:{
                    PORTC = (0<<CFG_OUT_A) |// SPI output mux
119                         (0<<CFG_OUT_B) |  // SPI output mux
                            (0<<CFG_CIC_D) |  // SPI input
                            (1<<CLEAR);
                }break;
                /* CIC Interpolate config set */
124             case CMD_CICI:{
                    PORTC = (0<<CFG_OUT_A) |// SPI output mux
                            (1<<CFG_OUT_B) |  // SPI output mux
                            (0<<CFG_CIC_I) |  // SPI input
                            (1<<CFG_CIC_CLK) |// CLK div 4
129                         (1<<CLEAR);
                }break;
                /* Delta-Sigma Modulator config set */
                case CMD_DSM:{
                    PORTC = (1<<CFG_DSM) |  // SPI input
134                         (1<<CLEAR);
```

```
            }break;
            default: continue;
        }
        /* Wait for interrupts after command byte is set */
        while(command) PORTA = ~SPSR;
    }
}

/*
 * Initializes ports and registers. Watchdog resets MCU and chip
 * between transmissions. In practice, this enables only a single
 * transmission of data, suitable only for a test setup.
 */
int main(void)
{
    /* Port init */
    DDRA = 0xFF;
    PORTA = ~0x01;

    /* Single input. Rest of PORTB are output */
    DDRB = ~(1<<MISO);
    DDRC = 0xFF;

    /* USART init */
    UBRRL = UBRR;
    /* Enable RX, TX, RX interrupts */
    UCSRB = (1<<TXEN) | (1<<RXEN) | (1<<RXCIE);
   /* Set 1 stop bit, frame format to 5bits */
    UCSRC = (1<<URSEL) | (0<<USBS) | (0<<UCSZ1) | (0<<UCSZ0);

    /* SPI Init PORTB */
    /* SPI Enable, Master, SPI_clk: fosc/16, Interrupt enable */
    SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0) | (1<<SPIE);
    /* Watchdog Init, about 1 sek. */
    WDTCR = (1<<WDE) | (6<<WDP0);
    /* Clear on init */
    PORTC = (0<<CLEAR);

    /* Enable interrupts and go to command loop */
    sei();
    mainLoop();
}
```

E Python host script

Most of the measurements are done using the DAQ device setup. The below script shows how the micro-controller setup is done very efficiently, requiring only a small amount of code. For comparison; an estimated 800 lines of Python code is written for host side control of the DAQ setup.

hostside.py

```
#!/usr/bin/env python
from __future__ import with_statement
import serial, sys, time
import numpy
from pylab import *

# Command byte protocol
CMD_CICI = chr(1)
CMD_CICD = chr(2)
CMD_FIR = chr(3)
CMD_DSM = chr(4)

def com_write(datastr):
    """ Writes an array to the serial port """
    def int_to_bytes(i):
        """ Converts input integer to tuple of two five-bit bytes """
        low = i & 0x1F
        hi = (i>>5) & 0x1F
        return "%c%c"% (hi, low)

    for i in datastr:
        ser.write(int_to_bytes(i))

def com_read():
    """ Reads array from serial port """
    def bytes_to_int(bytes):
        """ Combines two five-bit bytes to a single integer """
        hi = ord(bytes[0])
        low = ord(bytes[1])
        if hi & 0x10: # negative value
            return -0x400 | (((hi<<5) | low))
        else:
            return (hi<<5) | low

```

```
    # read bytes, combine two and two bytes into array values
    i = 0
    while ser.inWaiting() >= 2:
        bytes = ser.read(2)
        val = bytes_to_int(bytes)
        readArray[i] = val
        i += 1

def plot_result():
    """ Plot input and filtered data. """
    figure(1); clf()
    xval = range(0, len(yval))
    plot(xval, yval)
    xval = range(0, len(readArray))
    stem(xval, readArray)
    axis([0, len(yval), -520, 520])
    show()

# Setup serial port
ser = serial.Serial(1, baudrate=19200, bytesize=5, parity='N',
    stopbits=1, timeout=0.1)

try:
    print ser.portstr
    print "\n---input:---"
    # Write CIC configuration command to MCU
    ser.write(CMD_CICD)
    # Generate test signals
    f_s = 44100
    f_max = f_s/4
    w1 = 0.0063*2*pi*f_max
    xval = arange(start=0, stop=0.02, step = 1.0/f_s*10)
    yval = 511*sin(xval*w1)
    yval = require(yval, dtype='int16')
    # To test CIC decimator, SPI input requires repeated values
    # to emulate Nyquist clock rate operation.
    yval = yval.repeat(8)
    readArray = zeros(len(yval), dtype='int16')

    # Write data to chip through MCU
    com_write(yval)
    print "\n---output:---"
    # Some delay to allow computer serial buffer to fill
    time.sleep(0.2);
    com_read()
    plot_result()
finally:
    ser.close()
```

Bibliography

[1] K. Adaos, G. Alexiou, and N. Kanopoulos. Development of reusable serial fir filters with reprogrammable coefficients designed for serial dataflow architectures. In *Electronics, Circuits and Systems, 2000. ICECS 2000. The 7th IEEE International Conference on*, volume 1, pages 567–570 vol.1, 2000.

[2] ZigBee Alliance. Zigbee success story: Perfect ice conditions ensure faster speed skating times. Success story. Available online at www.zigbee.org, 2009.

[3] S. Ardalan and J. Paulos. An analysis of nonlinear behavior in delta - sigma modulators. *Circuits and Systems, IEEE Transactions on*, 34(6):593–603, Jun 1987.

[4] Avr121: Enhancing adc resolution by oversampling. Application note. Available online at www.atmel.com, 2005.

[5] T.J. Barnes. Skill: a cad system extension language. In *Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE*, pages 266–271, Jun 1990.

[6] Maciej Borkowski. *Digital Delta-Sigma Modulation. Variable modulus and tonal behaviour in a fixed-point digital environment.* PhD thesis, Faculty of Technology, Department of Electrical and Information Engineering, University of Oulu, 2008.

[7] R. Brodersen, A. Chandrakasan, and S. Sheng. Low-power signal processing systems. *VLSI Signal Processing, V, 1992., [Workshop on]*, pages 3–13, Oct 1992.

[8] Cadence design systems, Inc. *SKILL Language Reference*, product version 06.30 edition, 2007.

[9] J. Daniels, W. Dehaene, M. Steyaert, and A. Wiesbauer. A 350-mhz combined tdc-dtc with 61 ps resolution for asynchronous $\delta\sigma$ adc applications. In *Solid-State Circuits Conference, 2008. A-SSCC '08. IEEE Asian*, pages 365–368, Nov. 2008.

[10] A.E. de la Serna and M.A. Soderstrand. Trade-off between fpga resource utilization and roundoff error in optimized csd fir digital filters. In *Signals, Systems and Computers, 1994. 1994 Conference Record of the Twenty-Eighth Asilomar Conference on*, volume 1, pages 187–191 vol.1, Oct-2 Nov 1994.

[11] H. Fujisaka, R. Kurata, M. Sakamoto, and M. Morisue. Bit-stream signal processing and its application to communication systems. *Circuits, Devices and Systems, IEE Proceedings* -, 149(3):159–166, 2002.

[12] R.M. Hewlitt and Jr. Swartzlantler, E.S. Canonical signed digit representation for fir digital filters. In *Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on*, pages 416–426, 2000.

[13] Eugene B. Hogenauer. An economical class of digital filters for decimation and interpolation. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, 29(2):155–162, Apr 1981.

[14] ITRS. International technology roadmap for semiconductors, 2007 edition. Available online at http://www.itrs.net, 2007.

[15] Erwin Janssen and Derk Reefman. Super-audio cd: An introduction. *IEEE Signal Processing Magazine*, pages 83–90, July 2003.

[16] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, Dec. 2003.

[17] E.T. King, A. Eshraghi, I. Galton, and T.S. Fiez. A nyquist-rate delta-sigma a/d converter. *Solid-State Circuits, IEEE Journal of*, 33(1):45–52, Jan 1998.

[18] T.S. Lande, T.G. Constandinou, A. Burdett, and C. Toumazou. Running cross-correlation using bitstream processing. *Electronics letters*, 43(22), October 2007.

[19] Yong Liang, Qiao Meng, and Zhi-Gong Wang. The implementation of 1-ghz bit-stream adder used in signal processing in a 0.18-μm cmos technology. In *Solid-State and Integrated-Circuit Technology, 2008. ICSICT 2008. 9th International Conference on*, pages 1871–1873, Oct. 2008.

[20] Olav E. Liseth. Low power bitstream running cross-correlator/convolver. Master's thesis, University of Oslo, 2009.

[21] R.A. Losada and R. Lyons. Reducing cic filter complexity. *Signal Processing Magazine, IEEE*, 23(4):124–126, July 2006.

[22] J. Markus and G.C. Temes. An efficient delta-sigma adc architecture for low oversampling ratios. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 51(1):63–71, Jan. 2004.

[23] Alain J. Martin. Asynchronous datapaths and the design of an asynchronous adder. *Formal Methods in System Design*, (1):117–137, July 1992.

[24] The MathWorks, Inc. *MATLAB Documentation*, r2008b edition, 2008.

[25] Zhi-Jian (Alex) Mou. A study of vlsi symmetric fir filter structures. *The Journal of VLSI Signal Processing*, 4(4):371–377, Nov. 1992.

[26] C.-I.C. Nilsen and S. Holm. Distortion-free delta-sigma beamforming. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 55(8):1719–1728, August 2008.

[27] Steven R. Norsworthy, Richard Schreier, and Gabor C. Temes, editors. *Delta-Sigma Data Converters - Theory, Design and Simulation*. IEEE Press, 1997.

[28] Kiyoshi Oguri, Yuichiro Shibata, and Akira Nagoya. Asynchronous bit-serial datapath for object-oriented reconfigurable architecture pca. In *Advances in Computer Systems Architecture*, volume 2823/2003, pages 54–68. Springer Berlin / Heidelberg, 2003.

[29] P. O'Leary and F. Maloberti. Bit stream adder for oversampling coded data. *Electronics Letters*, 26(20):1708–1709, Sept. 1990.

[30] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing*. Pearson Prentice Hall, 4 edition, 2007.

[31] Richard Schreier and Gabor. C. Temes. *Understanding Delta-Sigma Data Converters*. John Wiley & Sons, Inc, 2005.

[32] E. Schuler and L. Carro. Reliable digital circuits design using sigma-delta modulated signals. *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on*, pages 314–324, Oct. 2005.

[33] C.E. Shannon. Communication in the presence of noise. *Proceedings of the IEEE*, 86(2):447–457, Feb 1998.

[34] L. E. Turner, P. J. W. Graumann1, and S. G. Gibb. Bit-serial fir filters with csd coefficients for fpgas. In *Field-Programmable Logic and Applications*, volume 975/1995, pages 311–320. Springer Berlin / Heidelberg, 1995.

[35] Mark Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.