NTNU

Norwegian University of
Science and Technology

Realization of sigma-delta DAC for audio application on FPGA

Håkon Tobias Jøsok

Master of Science in Electronics
Submission date: June 2017
Supervisor: Kjetil Svarstad, IES
Co-supervisor: Lars Sundell, Nordic Semiconductors

Norwegian University of Science and Technology
Department of Electronic Systems

Problem Description

Candidate name: Håkon Tobias Jøsok

Assignment title: Realization of Sigma-Delta DAC for audio application on FPGA

Assignment text: By reducing the analog circuitry at the expense of more digital circuitry, the Sigma-Delta DACs are able to get high dynamic range with only a two level signal on the output. This makes it very suitable for implementation on a FPGA for testing and development. The main goal of this task is to implement a complete solution for a Sigma-Delta DAC for audio application on a FPGA. The task will consist of making an interpolation filter for the proposed Sigma-Delta modulator solution made in an earlier project, and to implement the the complete solution on to a FPGA for testing. If necessary, methods for removing non-linear effects in the Sigma-Delta modulator such as dithering could be tried out to enhance its performance. The final solutions should be compared to a PWM solution.

Assignment proposer / Co-supervisor:
Lars Sundell
E-mail: lars.sundell@nordicsemi.no

Supervisor: Kjetil Svarstad

Abstract

Today, data converters are extensively used in embedded systems for a large number of applications. Data converters can have a big impact on power consumption on small embedded systems. Thus, designing data converters with lower power consumption and high performance is an important subject.

In this thesis, a proposed solution for a complete Sigma-Delta (S-D) digital to analog converter (DAC) for audio application is presented. The DAC is implemented on a field programmable gate array (FPGA), and the audio performance of the implementation is tested. A single sided pulse width modulation (PWM) DAC is implemented in register transfer level (RTL) code, and used as a comparison to the S-D DAC implementation. Both the S-D and PWM DAC is synthesized in TSMC's 55-nm technology, and a power estimation on the netlists is performed.

The S-D DAC implementations worked as expected, and successfully played music on an audio system. The performance of the S-D DAC implementation is limited by poor switching characteristics of the digital pad on the FPGA and Intersymbol interference (ISI). A total harmonic distortion (THD) of -82.2dB was measured at 3kHz using 16 bit samples. This is equivalent to an effective number of bits (ENOB) of 13.4 bits. The harmonic distortion varied depending on the input frequency, and increased for the lower frequencies. At 100Hz, a THD of -56.3dB was measured, equivalent to an ENOB of 9.1 bits.

In scenarios where the single sided PWM scheme and the designed S-D DAC are both applicable, the PWM DAC used up to 40 times less power. Suggestions for reducing the S-D DAC's power consumption are presented, which could reduce this power gap. The S-D DAC has a clearly better audio performance than the single sided PWM DAC. In scenarios where both are applicable, the choice is between a good audio performance or a low power consumption.

Preface

This report is the result of the work with my master thesis, and is written on behalf of Nordic Semiconductors at the Norwegian University of Science and Technology (NTNU), Department of Electronics and Telecommunications.

Starting this project back in August of 2016, as a project assignment, my knowledge of data converters in general, was limited. The learning curve has been steep, but also very educational. The thesis combined theory and practical work, which suited me well. The final result from my work was a working DAC for audio, and it was exciting to actually hear the results from my work.

I want to thank my supervisor Lars Sundell at Nordic Semiconductors, and Kjetil Svarstad from NTNU for the help and guidance with my thesis. I also want to thank Lars Lundheim at NTNU for the help with the digital signal processing (DSP) related work, and Tore Fotland for the proofreading. A special thanks to my girlfriend Mathilde for the support throughout my work with the thesis.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AMBA Advanced Microcontroller Bus Architecture

ASIC Application-Specific Integrated Circuit

AUX Auxiliary Port

CD-DA Compact Disc Digital Audio

CIC Cascaded Integrator-Comb

CMOS Complementary Metal-Oxide-Semiconductor

D-F Direct-Form

DAC Digital to Analog Converter

dBFS Decibels relative to Fulls Scale

DDR3 Double Data Rate type three

DMA Direct Memory Access

DR Dynamic Range

DSP Digital Signal Processing

DUT Device Under Test

ENOB Effective Number Of Bits

FF Flip Flop

FFT Fast Fourier Transform

FIFO First In First Out

FIR Finite Impulse Response

FLAC Free Lossless Audio Codec

FPGA Field Programmable Gate Array

GPIO General-Purpose Input/Output

HDL Hardware Description Language

IF Interpolation Filter

IIR Infinite Impulse Response

ILA Integrated Logic Analyzer

IP Intellectual Property

ISI Intersymbol Interference

JTAG Joint Test Action Group

LPF Low Pass Filter

LUT Look Up Table

MAC Multiplier-Accumulator

NTF Noise Transfer Function

NTNU Norwegian University of Science and Technology

OS Operating System

OSR Oversampling Ratio

PC Personal Computer

PL Programmable Logic

PS Processing System

PSD Power Spectral Density

PWM Pulse Width Modulation

QSPI Quad Serial Peripheral Interface

RMS Root Mean Square

ROM Read-Only Memory

RTL Register Transfer Level

S-D Sigma-Delta

SDK Software Development Kit

SNR Signal to Noise Ratio

SoC System on Chip

STF Signal Transfer Function

THD Total Harmonic Distortion

Chapter 1

Introduction

1.1 Motivation

Because of the vast use of battery-powered embedded systems, it has become essential to make energy efficient designs. Embedded systems are used in a wide range of applications, including ones that can require a cross between the digital and analog domain. This requires efficient data converts which are suited for the specific task. An example of this is audio generation on a microcontroller. Which usually requires a digital to analog converter (DAC) with high dynamic range (DR) and low power consumption, a feature which can be difficult to achieve.

Two types of DAC's which can be used in microcontrollers are the pulse width modulation (PWM) and sigma-delta (S-D) DAC's. The S-D DAC is widely used today because of its reduced analog circuitry compared to conventional Nyquist-rate DACs. The analog circuitry is the most critical part of the DAC, and can easily limit the performance of the DAC [7]. The S-D DAC uses less analog circuitry at the cost of more digital logic. Using the concepts of oversampling and noise shaping, the S-D DAC is able to achieve high DR using only a 1-bit internal DAC. The most essential part of the S-D DAC is the S-D modulator, which performs the noise shaping. The design of the S-D modulator is crucial, since it impacts the overall complexity and power consumption of the S-D DAC.

The PWM DAC works by generating square pulses on its output, where the average *on* time of the square pulse represent the desired voltage level. The PWM DAC is easy to implement on a microcontroller using a simple binary counter, but this implementation can have severe harmonic distortion and is not suited for high end audio conversion.

1.2 The thesis and previous work

In this thesis a S-D DAC is implemented on a field programmable gate array (FPGA) to test what audio performance can be achieved. The S-D DAC solution will also be compared to a PWM DAC solution in terms of power and audio performance.

In the previous work [6], a S-D modulator was designed and implemented in C code. The modulator was designed with the compact disc digital audio (CD-DA) format as a

standard for the performance goal. The S-D modulator is therefore designed for 16 bit samples with a sample rate of 44.1kHz. The C model was extensively simulated for a large range of frequencies and amplitudes. The results from the simulations showed that the modulator remained stable for inputs lower then -2.5 decibels relative to fulls scale (dBFS), and had a DR>96dB.

This thesis uses the modulator designed in [6], and proposes a solution for a complete S-D DAC implementation on a FPGA. The design process in this thesis uses an iterative design method.

1.3 Main contributions

1. Designed an interpolation filter (IF) with an oversampling ratio (OSR) of 128, which is compatible with the CD-DA format, and implemented the IF in register transfer level (RTL) code.

2. Implemented the S-D modulator from [6] in RTL code, and connected the S-D modulator to the IF.

3. Designed and implemented a S-D DAC intellectual property (IP) for a FPGA implementation, with a control register and a data streaming interface compatible with the advanced microcontroller bus architecture (AMBA) AXI4 bus system.

4. Implemented the S-D DAC IP on a Zynq-7000 system on chip (SoC), and measured the audio performance of the implementation with a spectrum analyzer.

5. Compared the performance, area and power consumption of the S-D DAC solution to a PWM DAC solution.

1.4 Thesis Organization

Chapter 1 presents the motivation, this thesis and previous work, the main contributions, and the thesis organization. Chapter 2 presents the background on DAC's in general, PWM DAC's, S-D DAC's, IF design and a overview of the ZedBoard. Chapter 3 describes the design flow for implementing the complete S-D DAC solution on the FPGA, and the design of a PWM DAC used for testing. Chapter 4 presents the measurements of the S-D DAC implementations, and the results from the power estimations of the S-D DAC and the PWM DAC. In chapter 5, the conclusions and future work is presented. Appendix A lists the source code attached to this work, and appendix B shows the detailed reports from the power estimations.

Chapter 2

Background

2.1 DACs in general

General DAC theory will be explained in the next subsections.

2.1.1 Bits and resolution

DACs are generally specified by the N bits on the input, also called the resolution of the DAC. The bits on the input represent the number of voltage levels which the DAC can generate. A N-bit DAC can generate 2^N voltage levels ranging from zero to its full scale value V_{FS} [7]. The resolution of the DAC is sometimes defined as the smallest analog value the DAC can generate, but in this thesis it is defined as the number of bits on the DAC input.

2.1.2 Sampling rate

The Nyquist criteria for sampling say that the sampling frequency f_s must be minimum twice the signal frequency f_B to avoid aliasing [1]. This criteria limits the signal bandwidth to $f_B = f_s/2$ for a DAC, where f_s represent the maximum sample rate where the DAC can generate the correct value on its output.

A signal is oversampled if it is sampled with a higher frequency then the Nyquist frequency $f_N = 2 \cdot f_B$, given by the Nyquist criteria. How much the signal is oversampled can be specified by the oversampling ratio (OSR). The OSR is the ratio between the sample frequency f_s and the Nyquist frequency f_N, and is defined in equation (2.1).

$$\text{OSR} = \frac{f_s}{f_N} = \frac{f_s}{2f_B} \tag{2.1}$$

2.1.3 Dynamic range

The DR of a DAC is the ration between the largest and smallest signal it can generate. A N-bit DAC has a theoretical DR given by equation (2.2) [8].

$$\text{DR} = 6.02N + 1.76 \text{ [dB]} \tag{2.2}$$

In practice the theoretical DR in equation 2.2 is not achievable, due to nonlinearities and semiconductor noise sources in the DAC [8].

2.1.4 dBFS

dBFS is a unit measurement used in digital signal processing (DSP) for amplitudes. Zero dBFS is at the full scale or maximum amplitude of the signal, and smaller amplitudes are negative values.

2.1.5 Total harmonic distortion (THD)

Total harmonic distortion (THD) is the ratio between the root mean square (RMS) value of the fundamental signal, and the RMS sum of all its harmonic components. In practice only the first 5 harmonics are included in the measurement since the rest of the harmonics have minor contributions to the result [1]. THD is defined in equation (2.3).

$$\text{THD} = \frac{\sum_{\infty}^{n=2} \text{harmonics}}{\text{fundamnetal}} \tag{2.3}$$

2.1.6 THD+N

THD plus noise is the ratio between the RMS value of the fundamental signal and the RMS sum of all its harmonic and noise components. THD+N is defined in equation (2.4).

$$\text{THD} + \text{N} = \frac{\sum_{\infty}^{n=2} \text{harmonics} + \text{noise}}{\text{fundamnetal}} \tag{2.4}$$

2.1.7 Signal to noise ratio (SNR)

Signal to noise ratio (SNR) is defined as the ration between the RMS value of the signal, and the total RMS noise in baseband. If the THD+N is measured in baseband the SNR can be defined as equation (2.5).

$$\text{SNR} = \frac{\text{fundamnetal}}{\sum_{\infty}^{n=2} \text{harmonics} + \text{noise}} \tag{2.5}$$

2.1.8 Effective number of bits (ENOB)

Effective number of bits (ENOB) is a way of expressing the SNR of a DAC in terms of bits. The theoretical limit for DR expressed in equation (2.2) is also the theoretical limit

for the SNR of the DAC. Solving equation (2.2) for N, and replacing N with ENOB and DR with SNR gives the following expression, represented by equation (2.6).

$$\text{ENOB} = \frac{\text{SNR} - 1.76}{6.02} \tag{2.6}$$

2.1.9 DAC settling time

The input to output settling time is defined as the time between the digital input code changes, to the output is stable in a defined error band [1]. In figure 2.1 a DAC transition is shown where the four periods of the settling time is characterized.

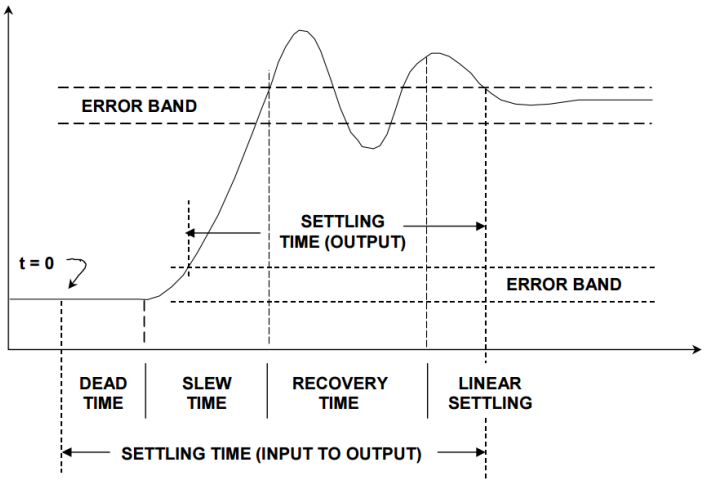

Figure 2.1: Settling time for DAC, from [1]

The first period is the *dead time*. Here the digital logic is switching but there is no change on the output. The second period is the *slew time*, where the rate of change on the output is limited by the DAC's slew rate. The third period is the *recovery time*. In this period the DAC is recovering from its fast slew rate, and may overshoot and/or undershoot. The fourth period is the *linear settling time*, where output converges to its final value defined within an error band [1].

Ideally, the transition of the DAC from one value to another should happen monotonically, but in practice it will overshoot and/or undershoot during the settling time [1]. This uncontrolled movement on the output, when the DAC is transitioning, is called a glitch. Glitches is often characterized by the glitch impulse area, which is the net area of the glitch [1]. A measurement of the glitch area is show in figure 2.2.

As can be seen in figure 2.2, the net glitch area can cancel out when the glitch has equal amount of under- and overshoot.

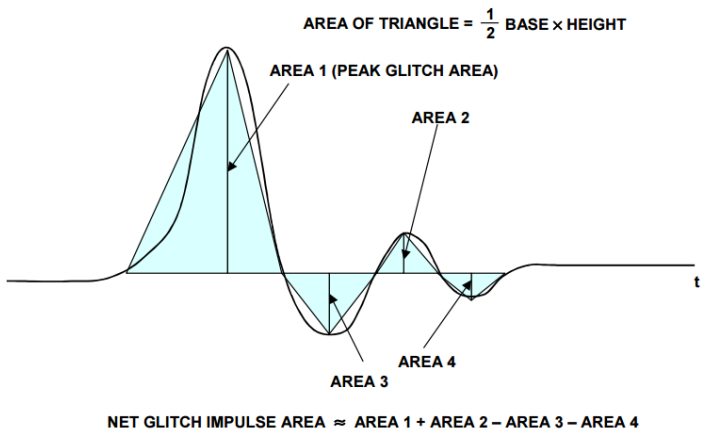

Figure 2.2: Glitch area measurement, from [1]

2.1.10 Intersymbol interference (ISI)

Intersymbol interference (ISI) is the error produced by the non-idealities when the DAC is transitioning. This can be caused by asymmetric switching, clock skew and capacitive memory effects [9]. A ISI error is dependent on the value of the previous symbol, since two consecutive equal symbols will not produce the switching non-idealities. A ISI error sequence is therefore dependent on the switching pattern of the DAC, and is nonlinear quantity [9]. ISI is a major source of distortion for S-D DAC [9].

2.2 PWM DACs

The PWM DAC is a type of a DAC where the idea is to modulate a stream of square pulses, and filter them in an analog low pass filter (LPF). Each square pulse has a time period T_s where the square pulse can either be on or off. The time period T_s is given by the sample rate of the input signal, and is equal to $T_s = 1/f_s$. The average time the square pulse is on in the time period T_s, is called a duty cycle. The duty cycle is decided by the digital input code. The pulse stream is filtered with a LPF to remove noise in the out-of-band frequencies, and to smoothen out the square pulses. Figure 2.3 show a block diagram of a PWM DAC where the N bit digital input code is called DAC_{IN}.

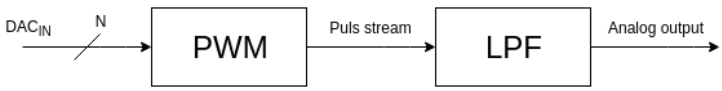

Figure 2.3: Block diagram of a N-bit PWM DAC

There are several differrent PWM schemes, but in this thesis only the single sided PWM scheme will be used. The single sided PWM scheme is the simplest scheme to

implement, but also the one with the worst harmonic distortion [10]. There are ways of reducing the harmonic distortion with signal processing algorithms called predistortion, but these are not explored in this thesis.

The single sided PWM scheme can be implemented by comparing a sawtooth wave from a simple binary counter to the digital input code. An example of this implementations, using a 5-bit binary counter, is shown in figure 2.4. The sawtooth wave from the binary counter is shown in the top plot, with a red line representing the value of input data code. The lower plot in figure 2.4 is the pulse stream out of the PWM block in figure 2.3. The x-axis of both the plots in figure 2.4 is time, with one integer step equal to a period of the binary counter's clock, $T_c = \frac{1}{f_c}$. In this example the period T_s of the PWM block is equal to the time the 5-bit counter uses to reach its max value. Thus, one period is equal to $T_s = 2^5 T_c$. In this example the digital input code is equal to 16, and represents a duty cycle of 50%, as shown in figure 2.4.

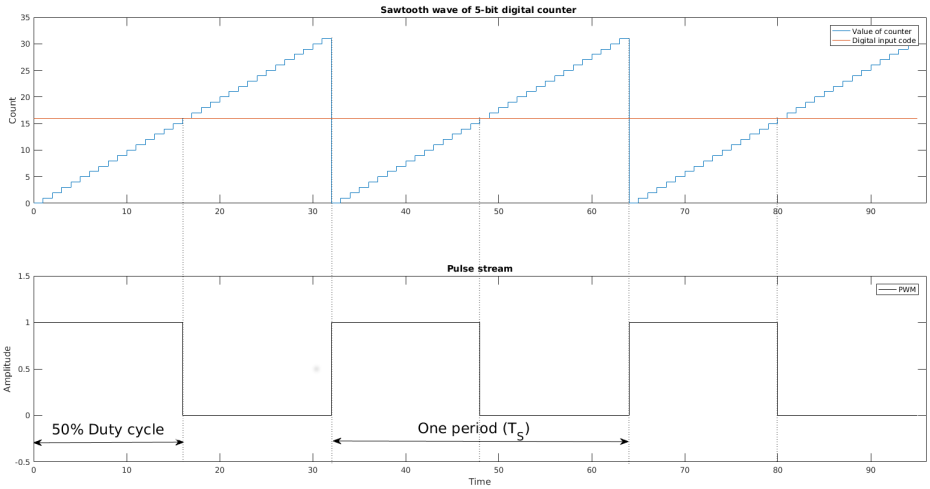

Figure 2.4: Sawtooth wave and PWM output

When a binary counter is used to implement the singled sided PWM scheme, the counter's clock frequency will limit what bit depth and signal bandwidth which is possible for the PWM DAC. The relationship between the sampling frequency f_s, bit depth N and the frequency of the counter clock f_c is shown in equation (2.7).

$$f_c = f_s \cdot 2^N \tag{2.7}$$

The clock frequency of the digital counter doubles for every extra bit of bit depth, and is also linearly dependent on the sampling frequency. Because of this property, the PWM DACs are in practice not capable of generating high end audio due to the extreme clock rate and switching capacities needed [10].

2.3 Sigma-Delta DACs

The most critical part of a DAC is the analog circuitry. The analog circuitry can limit the resolution and speed of the DAC because of component mismatch and nonlinearities, drift and aging, and parasitics to mention a few [7]. The S-D DACs uses less analog circuitry at the expense of more digital circuitry compared to the Nyquist rate DACs. This is done using the concepts of oversampling and noise shaping.

In figure 2.5 a block diagram of a S-D DAC is shown. The digital parts of the S-D DAC are the IF and the digital S-D modulator, while the analog parts are the M-bit internal DAC, and the analog LPF.

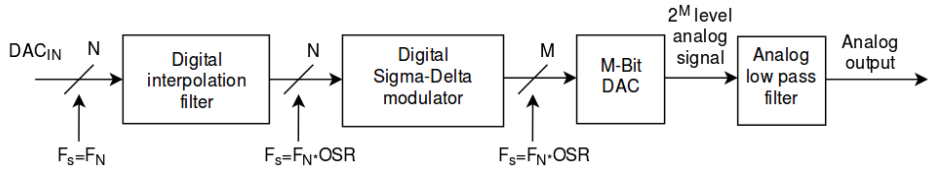

Figure 2.5: Block diagram of S-D DAC

The first block in figure 2.5 is the IF. The IF's task is to raise the sampling frequency from F_S to $F_S \cdot \text{OSR}$, and suppress the spectral replicas. The digital S-D modulator in the next block quantizes the N-bit word on its input, to usually 1-6 bits on its output [11]. This produces high amounts of quantization noise, which the S-D modulator filters out of baseband and up to the out-of-band frequency. This is done without significantly affecting the baseband spectrum in the process, and the whole process is called noise shaping. The S-D modulator can become unstable, and the input magnitude must be limited to a stable input range to avoid this [11]. This usually a few dB under the full-scale range of the S-D modulator [11]. The next block in figure 2.5 is the internal DAC. The internal DAC is usually 1-6 bits, depending on the output of digital S-D modulator in the previous block. If a 1-bit DAC is used, its output is a two level analog signal and the DAC will be a very simple structure. The last block in figure 2.5 is the analog LPF. The LPF's task is to remove all the out-of-band noise produced by the S-D modulator, without affecting the baseband signal in the process. Ideally the spectrum on the output of the LPF is the same as the input to the S-D DAC, without added noise or distortion.

2.4 Implementing interpolation filters for audio Sigma-Delta DACs

The IF is an essential part of the S-D DAC and can have a big impact on the overall power consumption and performance of the DAC. The design space for a IF is large with many methods for improving the efficiency of the design. The next sections will go through background for the most common design and implementation strategies for IFs.

2.4.1 Interpolation filters and specifications

Oversampling of a discrete-time signal by an integer factor I consists in principle of two processes. The first process raises the sampling frequency f_{si} by the integer factor I to $f_{so} = f_{si} \cdot I$. This is done by inserting $(I-1)$ zeros between the existing signal samples, and is called zero-stuffing. The second process suppresses the spectral replicas which is centered at $k \cdot f_{si}$, where $k = [1..(I-1)]$ [11]. This is done by filtering the signal with a digital LPF. The two processes are combined using an IF. The oversampling process with a factor of $I = 2$ is shown in figure 2.6. The original signal is at the top, the zero-stuffed signal is in the middle, and the final filtered signal, with the digital LPF's frequency response stippled, at the bottom.

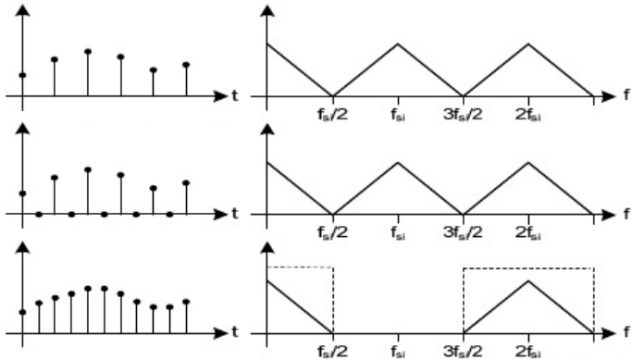

Figure 2.6: The process of oversampling in the time and frequency domain, from [2]

Oversampling is a lossless process, and ideally the output signal only contains the spectral content of the input signal. The frequency response of an ideal IF is categorised by equation (2.8) [2]:

$$H_{\text{IF}}(f) = \begin{cases} 1, & 0 < f < \frac{f_{si}}{2} \\ 0, & \frac{f_{si}}{2} \leq f < \frac{I \cdot f_{si}}{2} \end{cases} \tag{2.8}$$

This frequency response gives an impulse response equal to equation (2.9) [2]:

$$h_{\text{IF}}[n] = \text{sinc}\left[\frac{n}{I}\right] \quad , n \in \langle -\infty, \infty \rangle \tag{2.9}$$

This impulse response is infinite and non causal, and is not realizable for a real filter. The frequency specter of a real IF will therefore deviate from its ideal counterpart. These deviations are noticeable in the transition width, passband ripple, and stopband attenuation of the filter [2]. Depending on the IF's application, specifications must be set which describe the amount of deviation which is allowed for the particular IF.

The specification for a typical IF used in a S-D DAC are somewhat diffuse. The S-D DAC will not have subsequent sampling, which will lead to aliasing of the spectral replicas into baseband. Therefore, the LPF after the zero-stuffing could theoretically be omitted

[2]. The large content of high-frequency energy would however have a negative impact on the performance of the S-D DAC, by saturating the S-D modulator, and by making the internal DAC very jitter sensitive [2]. The amount of high-frequency energy the S-D modulator tolerates on its input, depends on the modulator design. The S-D modulator produce a high amount of high-frequency energy on its output, and if the spectral images are attenuated under the noise floor of the modulator, they will not affect the internal DAC and analog LPF [2]. The required attenuation of the spectral replicas at the higher frequencies are therefore reduced, and can be exploited to make more efficient IFs.

The CD-DA format, which was used as a performance goal in the design of the S-D modulator in [6], uses a sample rate of 44.1kHz. The passband for this format is defined to be at the presumable audible limit at 20kHz, and the stopband usually is defined from 24.1kHz [2].

The passband ripple should be under the audible limit, and for a typical high-end converter this usually ranges from 0.001dB to 0.0001dB [2]. The requirements for the stopband attenuation are as mentioned a little diffuse for IFs used in S-D DACs, but for a typical high-end converter it usually ranges from 75dB to 120dB [2].

2.4.2 FIR and IIR filters

When designing an IF one the first things to consider is if the LPF should be a finite impulse response (FIR) filter, or a infinite impulse response (IIR) filter. The FIR filter is a filter with a finite duration impulse response, and the output of the filter is only dependent on previous input values [12]. The output of a N^{th} order FIR filter with a impulse response $h[n]$, and a arbitrary input sequence $x[n]$ is equal to the convolution sum of the two. The convolution sum for a FIR filter is shown in equation (2.10) [12]. The b_m in equation (2.10) is the m^{th} coefficient of the filter, and determines the locations of the FIR filters zeros in the z-domain.

$$y[n] = h[n] * x[n] = \sum_{m=0}^{N} b_m x[n-m] \tag{2.10}$$

The IIR filter has in contrast to the FIR filter an infinite duration impulse response, and the output of the filter is dependent on both previous input and output values. The output response of a IIR filter with a impulse response $h[n]$, and an arbitrary input sequence $x[n]$ is a convolution of the two. This convulsion is shown in equation (2.11) [3]. The b_m in equation (2.11) is the coefficient which determines the zeros of the IIR filter in the z-domain. The a_k in equation (2.11) is the k^{th} feedback coefficient of the filter, and determines the locations of the IIR filter poles in z-domain.

$$y[n] = h[n] * x[n] = \sum_{m=0}^{N} b_m x[n-m] - \sum_{k=1}^{M} a_k x[n-k] \tag{2.11}$$

Since the FIR filter output is only dependent on previous input values, all FIR filters with bounded coefficients will be bounded-input bounded-output (BIBO) stable [12]. The IIR filter is not necessarily BIBO stable. To ensure that an IIR filter is BIBO stable, the poles of the filter must be inside the unit circle.

An important characteristic of the FIR filter is the ability to precisely manage the phase response of the filter. This attribute is often used to make a linear phase response, which gives a perfectly flat group delay [12]. This means that the waveform of the signal is preserved in the filtering process, which is important when filters are used in applications like audio where frequency dependent propagation delay cannot be tolerated [2]. The IIR filter in comparison cannot easily manage the phase response, and can only approximate linear phase characteristics.

FIR filters are most commonly used in S-D DAC's, since they easily can be designed with linear phase response, in addition to the fact that the hardware can be clocked at the input frequency of the IF [11]. The IIR has the advantage over FIR filters in that they can have a higher stopband attenuation with a lower hardware complexity. Regardless of this, they are not commonly used for S-D DAC's [11]. The S-D DAC designed in this thesis is for an audio application. Since filters with a constant group delay is desired for audio DACs [2], the linear phase FIR filter will be used in this paper.

2.4.3 Interpolation filter partitioning

When implementing an IF with a large interpolation factor I, the IF can in principle raise the sampling frequency to $f_{so} = f_{si} \cdot I$ in one step, but this will not be an efficient implementation. When the interpolation factor I is large, the passband and transition band of the filter will become very small. This will require a long FIR filter with a large number of computations, since the FIR filters increase in order with the inverse of the transition band [3]. This would also require all the digital circuitry of the FIR filter to function at this high clock rate, and thereby dissipate an unnecessarily large amount of power [11].

The sampling frequency is usually raised in multiple stages, where most of the computations is done at lower sampling frequencies [11]. This can be done by cascading interpolation filters with low interpolation factors, which together raise the sampling frequency to the desired interpolation factor I. An example of multistage filtering is shown in figure 2.7 with three IFs, each with an interpolation factor of $I_i = 2$, which cascaded together gives an interpolation factor of $I = 8$.

The first filter in a multistage IF has the most demanding requirements, because of the small transition band needed to remove the adjacent spectrum image, which can be very close to the passband [11]. For each subsequent filter, the transition band will increase since the spectral replicas will be further and further apart [2].

For an IF in a S-D DAC, the required attenuation of the spectral replicas is highest for the first filter, and is relaxed for each subsequent filter as discussed in section 2.4.1. The requirements for short transition band and high attenuation, causes the first filter to become the most computational heavy filter, but it is also the filter with the lowest clock rate. Each subsequent filter is less computational heavy as the clock rate increases. This reduces the number of computations compared to a single stage IF, and causes most of the computations to be done at a the lower clock rates, which reduces the power consumption of the digital logic.

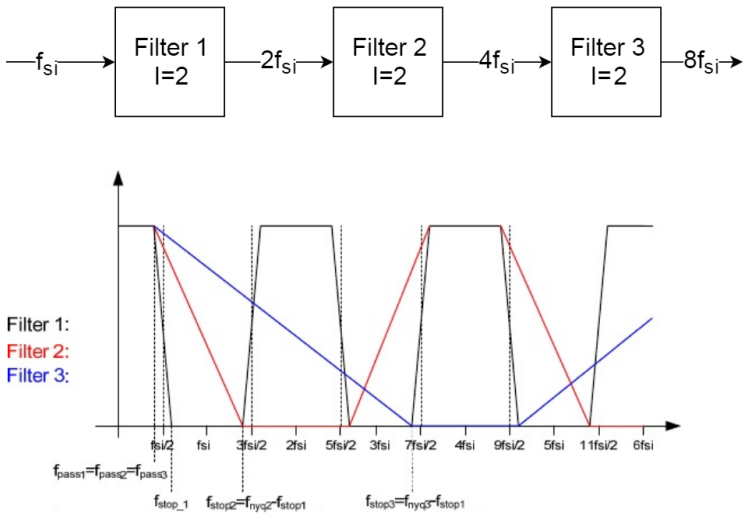

Figure 2.7: Interpolation with cascade filters, adapted from [2]

2.4.4 Interpolation filter structures

Every practical realizable digital filter can be described by a set of difference equations. These difference equations can be used to realize the filter into a filter structure. The filter structure is basically a pictorial block diagram of the difference equations represented by adders, multipliers and delays [3].

The convolution sum in equation (2.10) is the difference equation of the FIR filter, and can be directly realized into an filter structure [2]. This filter structure is called the direct-form structure. The direct-form structure is shown in figure 2.8 with a zero-stuffing block at the beginning of the filter, implying that filter is part of an IF with an oversampling factor I.

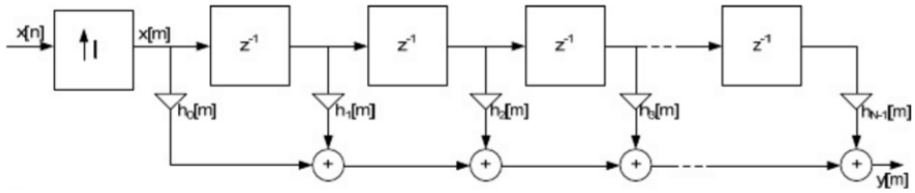

Figure 2.8: FIR filter direct-form structure, adapted from [2]

However, the direct-form structure is not an efficient implementation of an IF. The direct-form requires N multiplications for every output sample for a filter with length N, but only the I^{th} sample in the delay pipeline in figure 2.8 will be nonzero at any given time because of the zero-stuffing. This means there will be many redundant multiplications for every output sample, when the direct-form structure is used in a IF [2].

The polyphase structure is widely used to simplify the implementations of IFs, and uses the fact that only the I^{th} sample in the filter will be nonzero to its advantage [3]. The polyphase decomposition groups the impulse response of the IF into I subfilters of length M, where I is the interpolation factor of the IF [3]. The general form of the polyphase decomposition of a filter $H(z)$ in the z-domain, with I subfilters $P_k(z)$ is defined as equation (2.12) below [3]:

$$H(z) = \sum_{k=0}^{I-1} z^{-k} P_k(z^I) \tag{2.12}$$

where:

$$P_k(z) = \sum_{k=0}^{M-1} p_k[n] z^n \tag{2.13}$$

and:

$$p_k[n] = \sum_{k=0}^{I-1} h[nI + k] \tag{2.14}$$

This grouping of subfilters can be exploited in the realization of the polyphase structure. The realized direct-form polyphase structure and its transpose, with I subfilters are showed in figure 2.9. The two structures in figure 2.9 are equivalent [2].

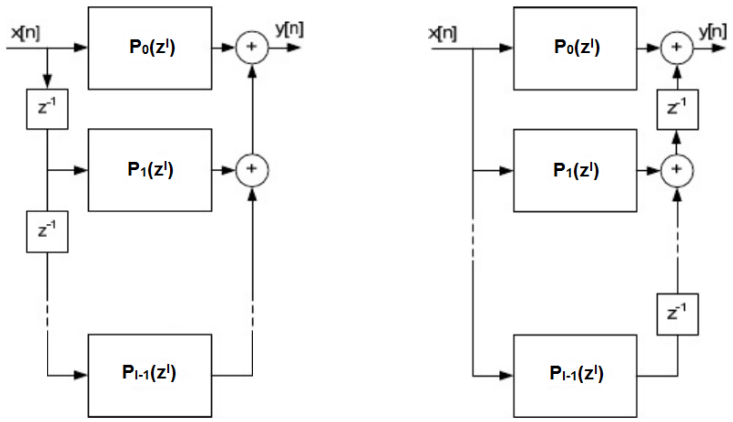

Figure 2.9: General polyphase FIR-filter structure and its transpose, adapted from [2]

The multirate identity for an upsampling system states that the filter can interchange with the zero-stuffer, if the filter is properly modified [3]. This principle is shown in figure 2.10.

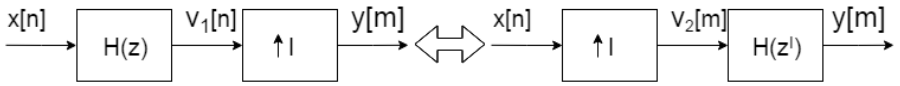

Figure 2.10: The multirate identity for an upsampling system

Applying the multirate identity on the transposed polyphase structure in figure 2.9 will change the subfilters system function from $P_k(z^I)$ to $P_k(z)$, and the subfilters will now run at the input frequency f_{si} [3]. The zero-stuffing and delay operations can now be replaced by a commutator as shown in figure 2.11. The commutator operates at the output sample rate $I \cdot f_{si}$, and start at $y_0[n]$ and sequentially picks up I samples for each input sample, by doing a full rotation counter clock wise [3].

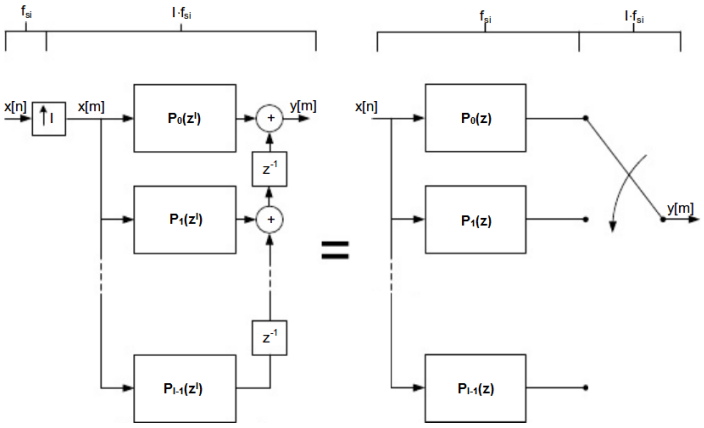

Figure 2.11: Efficient realization of the polyphase IF, adapted from [2]

The number of computations per input sample are the same as for the direct-form implementation, but the advantage of the polyphase structures is that the filter is operating at input sample rate f_{si}. This reduces the number of computations per second by I times for an IF with an interpolation factor of I, compared to the direct-form implementation [2].

2.4.5 Half-band filter

The half-band filter is a subclass of the linear-phase FIR filter, and is widely used in multirate systems. The half-band filter is characterized by its symmetric frequency response. The passband and stopband are equally wide with the center of the transition band at exactly at half the Nyquist frequency $\frac{f_{si}}{2}$ [3]. The passband and stopband ripple is also symmetric [3]. This symmetric frequency response of an ideal and a real half-band filter is shown in figure 2.12.

The impulse response of the ideal half-band filter is characterized by [3]:

$$h_{\frac{1}{2}}[n] = \frac{1}{2}\text{sinc}\left[\frac{n}{2}\right] \quad , n \in \langle -\infty, \infty \rangle \tag{2.15}$$

The ideal half-band filter cannot be realized in practice, because it has an infinite impulse response and is non causal. The half-band filter must be limited to a finite length $N = M + 1$, where M is the order of the half-band filter. The impulse response for a half-band filter with an order $M = 18$ is shown in figure 2.13.

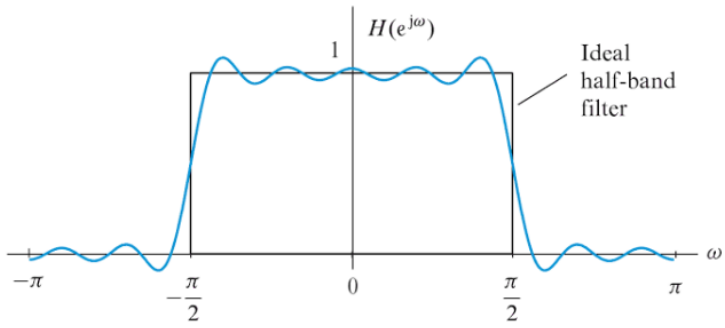

Figure 2.12: Frequency response of half-band filter, from [3].

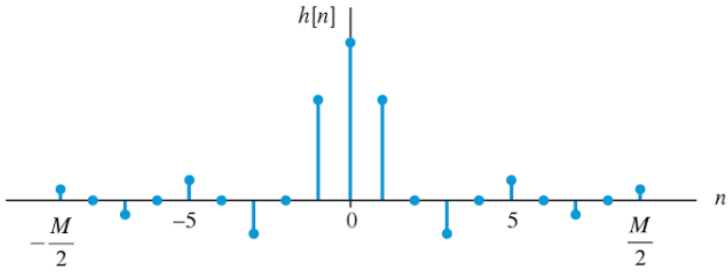

Figure 2.13: Impulse response of half-band filter with an order $M = 18$, from [3].

Since the impulse response is a sinc function all the even coefficients of the filter, except for $h_{\frac{1}{2}}[0]$, will be equal to zero. This can also be observed in the impulse response in figure 2.13. Thus, the half-band filter have about half the number of computations compared to an arbitrary FIR filter with the same length [2].

A causal half-band filter can efficiently be realized in the polyphase structure when it is used as part of an IF. Consider an IF with an interpolation factor of $I = 2$. Since the polyphase decomposition in equation (2.12) will group the impulse response into subfilters of even and odd terms, the impulse response of the half-band filter in the z-domain is equal to equation (2.16).

$$H(z) = \sum_{k=0}^{I-1} z^{-k} P_k(z^I) = P_0(z^2) + z^{-1} P_1(z^2) = h_{\frac{1}{2}}[0] + z^{-1} P_1(z^2) \tag{2.16}$$

Thus the combination of half-band filter with polyphase structure has about a quarter of the computations per second, compared to an arbitrary IF filter of same length.

2.4.6 Cascaded integrator-comb (CIC) filter

When implementing a conventional FIR filter in hardware, most of the resources will be used in the multiplications. The cascaded integrator-comb (CIC) filter is a subclass of linear phase FIR filters for decimation and interpolation, which uses no multiplication and limited storage [4]. This makes it very economical to use compared to conventional FIR filters for some applications.

As the name implies, the CIC filter consists of cascaded ideal integrator stages, and an equal number of comb stages which together produce a uniform FIR. Figure 2.14 shows the basic structure of the CIC IF.

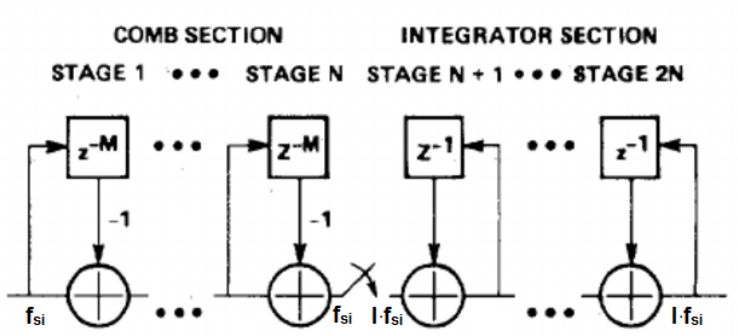

Figure 2.14: CIC IF, adapted from [4].

The N number of comb stages are operating at the lower sampling rate f_{si} of the filter, and each stage has a differential delay of M. This differential delay is in practice usually held to $M = 1$ or 2 [4]. The system response of a single comb stage is equal to equation (2.17) reference to the high sampling rate f_{so} of the CIC IF [4].

$$H_C(z) = 1 - z^{-IM} \tag{2.17}$$

The N integrators stages are operating at the output sampling rate $f_{so} = I \cdot f_{si}$ of the filter, where I is equal to interpolation factor. Each integrator stage is implemented with a system function shown in equation (2.18) [4]:

$$H_I(z) = \frac{1}{1 - z^{-1}} \tag{2.18}$$

Between the comb and integrator section is a zero-stuffer which increases the sample rate on the input of the integrator section. Combining equation (2.17) and (2.18) the system function for the complete CIC filter is produced, and is shown in equation (2.19) referenced to the high sample rate [4].

$$H(z) = H_I^N(z)H_C^N(z) = \frac{(1 - z^{-IM})^N}{(1 - z^{-1})^N} = \left[\sum_{k=0}^{IM-1} z^{-k}\right]^N \tag{2.19}$$

The frequency response of the CIC filter is given by equation (2.19) evaluated at $z = e^{j2\pi f/I}$, where f is referenced to the low sample rate of the filter. Using this, the frequency magnitude response can be showed to be equal to equation (2.20) [4].

$$H(e^{j2\pi f}) = \left| \frac{\sin(\pi M f)}{\sin(\pi f/I)} \right|^N \tag{2.20}$$

The CIC filter has a low-pass frequency response, and from equation (2.20) it is evident that the frequency response of the CIC filter is fully determined by the three integer factors M, I and N. This will limit the range of possible filter characteristics. To increase the attenuation of the spectral replicas, the differential delay M and order N of the filter can be increased at the cost of more hardware adders and increased passband drop. Another penalty is the increased gain of the filter, which is exponential with the order of the filter. For an interpolating CIC filter the net gain can be shown to be equal to equation (2.21) [13].

$$G = \frac{(MI)^N}{I} \tag{2.21}$$

Another downside with high order CIC filter is the data word-length. The CIC filters generally use full precision to remain stable, and this gives a large data word-width penalty for high order filters [13]. Given a number of bits B_{in} on the input, the output data word-width is given by equations (2.22) [4].

$$B_{max} = [N \log_2(IM) + B_{in} - 1] \tag{2.22}$$

The gain and increased data word-width may require the signal to be attenuated and/or truncated depending on the applications after the CIC filter.

The CIC filter has a passband drop, which increases with the order and differential delay of the CIC filter. This effect is generally unwanted when interpolating a signal for audio, since this can be audible. This effect can be compensated by using a compensation FIR filter in cascade with the CIC filter, which ideally is an inverted version of the CIC filter over the passband. An example of this concept is shown in figure 2.15 with the frequency response of a CIC filter, a compensation FIR filter , and the composite filter. In this example the passband is 4kHz, and the magnitude is normalized to 0 dB.

The inverse sinc filter is a FIR filter subclass which can be used to compensate for the passband drop. The passband response of the inverse sinc filter is a slight rise, similarly to the compensation filter in 2.15, and the transition- and stopband can be configured like any arbitrary FIR filter.

2.4.7 Coefficient sharing

Since linear phase FIR filters has a symmetric or anti symmetric impulse response, the coefficients on each side of the center tap will be equal or have inverted sign [3]. This can be exploited by halving the number of stored coefficients, and by using the coefficient sharing structure illustrated in figure 2.16 [2]. This implementations will halve the number of multiplications in comparison to the a direct-form structure.

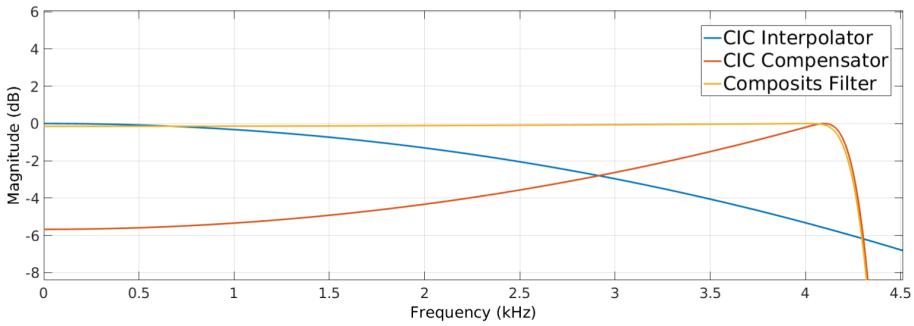

Figure 2.15: CIC filter, compensation FIR filter, and the composite filter frequency response.

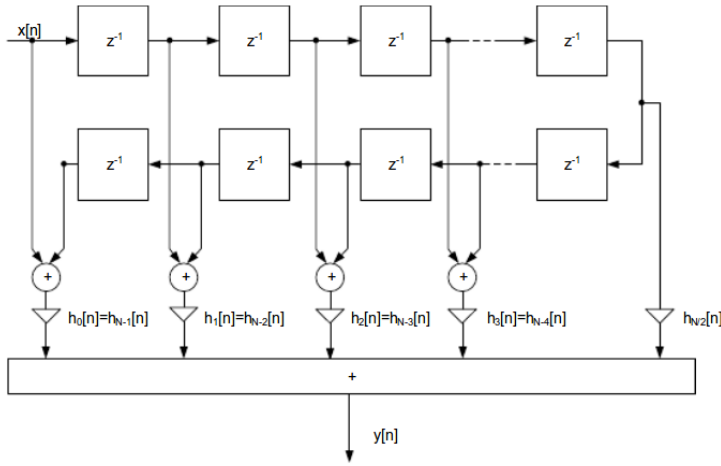

Figure 2.16: Symmetrical FIR filter with coefficient sharing, from [2].

The number of delays and additions remains the same as for a direct-form structure [2]. The coefficients sharing structure can also be used in a polyphase subfilter, provided that the subfilter has symmetric or anti symmetric coefficients.

2.4.8 Finite wordlength effects

When implementing a practical digital filter, all the data in the filter must be represented by a finite number of bits. This means that; the filters coefficients must be quantized, and a number format to represent the coefficients and data must be chosen. The finite wordlength arithmetic inside the filter can also require intermediate multiplication results to be quatized, and may cause overflow errors. The finite wordlength effects may introduce unacceptable errors in the filter, and must be taken into consideration when implementing the filter.

2.4.8.1 Fixed-point versus floating-point arithmetic

The first choice to make in the quantization process is if filter coefficients and data should be formatted in fixed-point or floating-point arithmetic. In many cases this will be a choice between the filters performance and its computational load. The floating-point arithmetic gives a high numerical precision, but is more computationally heavy then the fixed-point arithmetic [3]. When implementing a filter with low cost and resource usage, the fixed-point arithmetic must be used [3].

2.4.8.2 Quantization of filter coefficients

When the number format is chosen, the filter coefficients are quantized in this format. This is a one time operation, which causes the poles and zeros of the filter to move [2]. This happens since the quantization changes the coefficients infinite precision values, and is dependent on the quantization error. This changes the filter characteristics from its ideal values, and is noticeable in the frequency response, and/or its stability [3]. How much the filter characteristics changes depends on the filter structure, and the number of bits used to represent the coefficients [3]. If the filter characteristics of the new quantized filter violates the design specifications, the coefficients must be quantized using more bits until the specifications are satisfied. Using higher precision on the coefficients increases the length of the multiplications in the filter, and thus the filters computational load. There is a trade off between resource usage and performance, when choosing the precision of the coefficients [2].

2.4.8.3 Round-off noise

The round-off noise is caused by the quantization of intermediate multiplication results in the filter, which might need to be quantized to avoid the wordlength growing to much through the filter structure [2]. The quantization causes the filter to become a nonlinear system, which is challenging to understand an analyze theoretically [3]. The effects can be approximated by using a linear model of the quantization process to make educated design choices. However the most efficient way to analyze the effects is to simulate the filter, and examine the performance [3]. How the filter responds to the quantization of the internal variables is dependent on the filter structure [2].

A linear model of the quantization process, where the quantization is replaced by a additive white noise source on a direct-form FIR filter structure, is shown in figure 2.17. In figure 2.17 each intermediate multiplication result is quantized.

The quantization noise source $e_k[n]$ is assumed to be a wide-sense stationary white noise process with a zero mean. The quantization interval is uniformly distributed and given by $\Delta = 2^{-B}$, where B is the number of bits used in the quantization. Assuming that all the noise sources $e_k[n]$ are uncorrelated with a variance σ_e^2 expressed by equation (2.23).

$$\sigma_e^2 = \frac{\Delta^2}{12} = \frac{2^{-2B}}{12} \tag{2.23}$$

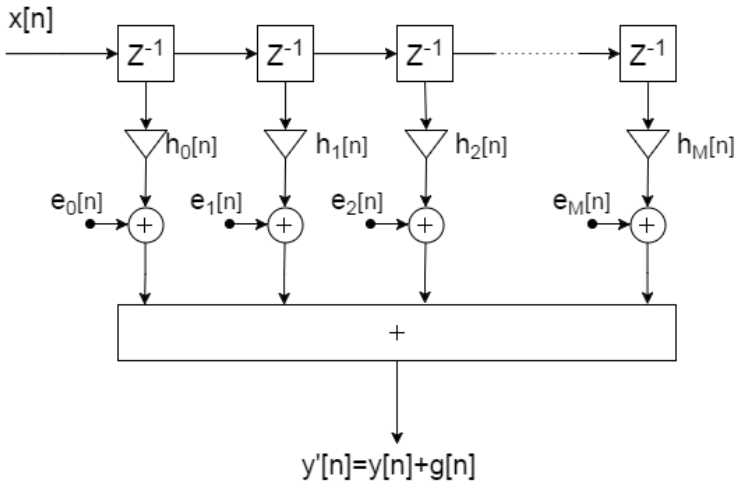

Figure 2.17: Linear noise model of qutanzation process in FIR filter, adapted from [3].

It can be shown that quantizing each intermediate multiplication result in the filter to $B+1$ bits, the total quantization noise power will be equal to equations (2.24) [3].

$$\sigma_g^2 = (M+1)\frac{\Delta^2}{12} = (\frac{M+1}{3})2^{-2(B+1)} \tag{2.24}$$

If a double length $2(B+1)$ accumulator is available, the final result can be quantized on the output of the filter. With the same assumptions as before it can be shown that this quantization noise power will be equal to equations (2.25) [3]:

$$\sigma_g^2 = \frac{\Delta^2}{12} = \frac{1}{3}2^{-2(B+1)} \tag{2.25}$$

Equation (2.25) show that the quantization noise will not increase proportionally with the number of coefficients M, as for the intermediate quantization used in equation (2.24). Thus the quantization noise will be lower for a FIR filter of arbitrary length M when using the second method. However, the computational load of the filter is increased, since all the accumulators are now double length $2(B+1)$, in comparison to the single length $B+1$ accumulators in figure 2.17.

2.4.8.4 Overflow errors

Overflow errors is large errors due to addition overflow, which causes the accumulator to wrap around or saturate. When an accumulator wrap around, a two's compliment number will change sign and thereby introduce large errors. When a accumulator saturates, the output saturate at the maximum or minimum number that the accumulator can represent. An accumulator that saturates will need more digital logic than an accumulator which wrap around. The overflow errors can be prevented by properly scaling the numbers to be added.

2.4.9 Filter implementation strategies

There are several different filter implementation strategies which can be used when implementing digital filters in hardware. The next sections will discuss the direct implementation and the multiplier-accumulator (MAC) implementation, which is two of the most common filter implementation strategies.

2.4.9.1 Direct implementation

The direct implementation, also called fully parallel implementation, is a direct implementation of the direct-form filter structure [2]. The direct implementation is shown in figure 2.18.

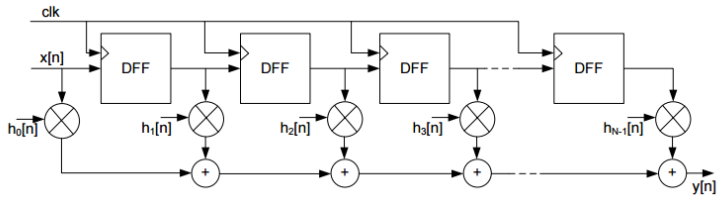

Figure 2.18: Direct implementation of FIR filter, from [2].

The advantage of this implementations is that it computes the output in one clock period, and can thus be very fast. The disadvantage of this implementations is the high amount of resource usage, caused by the large amount of delay elements, multiplications and additions. The worst case timing path through the hardware structure can also become very long for high order filters, and can cause the filter to not fulfill the timing constrains. However, for multiplier-free filters like the CIC filter this can be an efficient hardware implementation, since its the multipliers that use most of the resources [2].

2.4.9.2 MAC implementation

The MAC implementation, also called fully serial implementation, is a serialized implementation of the direct-form filter structure. In this implementation the coefficients are stored in a read-only memory (ROM), and is consecutively multiplied with the corresponding input sample [2]. The result from each multiplication is added up by a accumulator, and set to the output when all the multiplications are done. The input samples are stored in a register which inputs a new sample every time the filter computations are done, and rotates the stored input samples one step higher in the registers. In every rotation of the register, the last stored sampled in the chain is simply dismissed. Figure 2.19 show the MAC implementation of a FIR filter.

The MAC-implementation requires only one multiplier and one accumulator regardless of the filter order, but also requires two N length register to store the coefficients and the input samples for a N^{th}-order FIR filter. The resource usage is substantially reduced in comparison to the direct implementation, since its the multipliers which uses

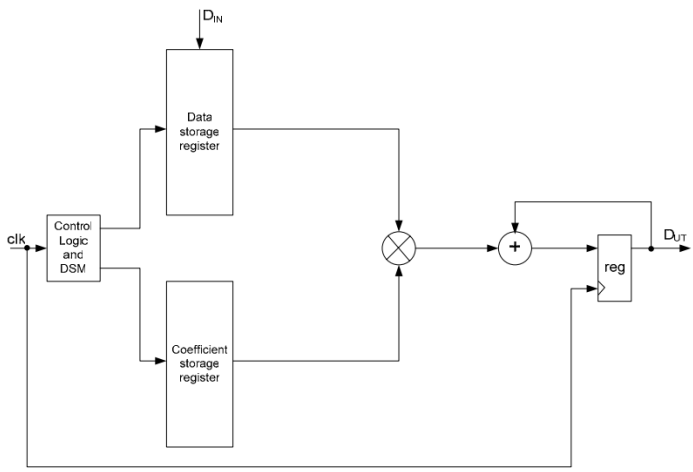

Figure 2.19: MAC implementation of FIR filter,from [2].

most of the resources in a hardware implementation [2]. The disadvantage with the MAC-implementation is that the hardware must run N times faster than the sample rate to complete all the computations before receiving a new input sample. The MAC-implementation is therefore most suited for low speed operations, and is the most dominant implementation for IF used for audio applications [2].

2.5 ZedBoard

The complete S-D DAC solution is to be implemented on a FPGA, and therefore a target FPGA platform must be chosen. For this project the ZedBoard development board was chosen. The ZedBoard is a development board which is based on the Xilinx Zynq-7000 All Programmable SoC. The Zynq-7000 combines a dual Cortex-A9 processing system (PS), and 85.000 Series-7 programmable logic (PL) cells which are coupled together [14]. The ZedBord also features 512MB double data rate type three (DDR3) memory, 256Mb quad serial peripheral interface (QSPI) flash, and a range of interfaces like an USB-JTAG and an USB-UART bridge which the Zynq-7000 can utilize [14]. Figure 2.20 show a overview of the Zynq-7000 SoC.

The Zynq architecture enables software programming on the PS and implementation of custom logic on the PL. The PS can communicate with PL cores using the ABMA AXI4 bus interface. The PS can be configured to provide clock sources and reset signals to the PL cores. The PL can also run independent of the PS by using the 100 MHz oscillator on the ZedBoard as a clock source. The dual Cortex-A9 on the PS is capable of running a range of operating system (OS) like Linux and FreeRTOS^{TM}, but can also run a a bare-metal single-threaded environment which provides basic features.

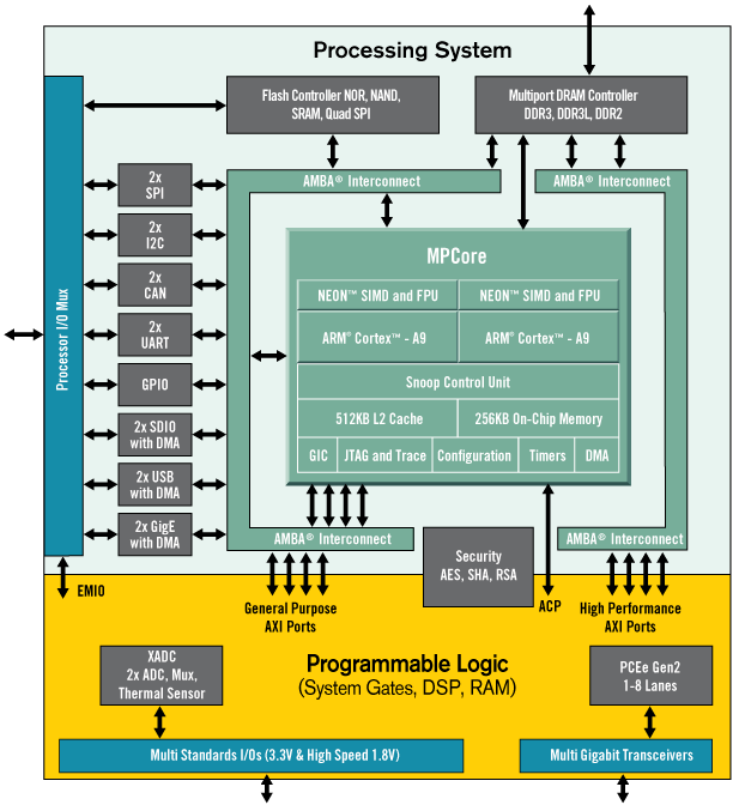

Figure 2.20: Xilinx Zynq-7000 SoC overview, from [5]

Chapter 3

Design and Implementation

3.1 Design and implementation of interpolation filter

The first task is to design and implement a IF for the S-D modulator from the project assignment in [6]. The specifications for an IF in a S-D DAC is, as mentioned in section 2.4.1, quite diffuse, and the design space is large with many parameters to consider. The IF is designed using the well known design and implementation techniques described in 2.4, and a limited time is used to explore a wide range of design configurations. The final solution is implemented in RTL code, in the Verilog language. In the following sections the design and implementation of the IF for the S-D DAC is described.

3.1.1 Specifications

The sample rate and bit depth on the input of the IF is given by the CD-DA format, which was used as the standard when designing the S-D modulator. This gives a sample rate of 44.1kHz, and a bit depth of 16 bits on the input.

The interpolation factor I of the IF is given by the OSR of the S-D modulator, which in this case is 128. The output wordlength of the IF is set by the input wordlength on the S-D modulator, and is 16 bits.

The passband ripple should not be audible, and for a typical high-end DAC it is usually ranging from 0.001dB to 0.0001dB, as discussed in section 2.4.1. Using this as a guideline, the specification for the maximum allowed passband ripple is set to the lower end of this at 0.001dB.

The specifications for the phase response of the IF is set to a linear phase response, in accordance with standard used for audio filters.

The finite wordlength arithmetic in the filter will add quantization noise to the signal, and this noise should be audible. The design goal for maximum added distortion in the baseband is set to 1dB.

The requirements for attenuation of the spectral replicas is set with a IF partitioning implementation in mind. The spectral replicas should be attenuated under the noise floor of the S-D modulator, but for the spectral replicas close to the passband where the S-D

modulators noise floor is very low, a maximum attenuation of 100dB is set. This is in the middle range of the attenuation used by a typical high-end DAC, as discussed in section 2.4.1. The general specifications for the IF is summarized in table 3.1.

Table 3.1: Summary of IF specifications

OSR	Input sample rate	Input & output wordlength	Passband ripple	Distortion
128	44.1kHz	16 bits	<0.001dB	<1dB

3.1.2 Implementation

The implementations of the IF was done by using well known theory and design methods as a starting point, and then using the DSP toolbox in Matlab for testing and finding a suitable solution. The hardware description language (HDL) coder in Matlab is used to generate Verilog code of the IF. The HDL coder has some limitations in the implementations available for the IF, so the generated Verilog code is manually optimized, making the IF much more efficient. In the attachment to this thesis, the complete Matlab script used when designing the IF filter is available.

3.1.2.1 Partitioning

Since the IF had a interpolation factor of 128, a partitioning of the IF will have a huge positive impact on the efficiency of the filter in terms of resource and power consumption. The IF is therefore partitioned in seven stages, where each stage has an interpolation factor I of 2, which together gives a total interpolation factor of 128. The input sample rate, passband, transition-band and stopband for each filter stage is determined, and is summarized in table 3.2.

Table 3.2: Input sample rate, passband, transition-band, and stopband for each IF stage.

IF #	Input sample rate	Passband	Transistion-band	Stopband
1.	44.1kHz	0-20kHz	20kHz-24.1kHz	24.1kHz-44.1kHz
2.	88.2kHz	0-22.05kHz	22.05kHz-66.1kHz	66.1kHz-88.2kHz
3.	176.4kHz	0-22.05kHz	22.05kHz-154.4kHz	154.4kHz-176.4kHz
4.	352.8kHz	0-22.05kHz	22.05kHz-330.8kHz	330.8kHz-352.8kHz
5.	705.6kHz	0-22.05kHz	22.05kHz-683.6kHz	683.6kHz-705.6kHz
6.	1.411MHz	0-22.05kHz	22.05kHz-1.389MHz	1.389MHz-1.411MHz
7.	2.822MHz	0-22.05kHz	22.05kHz-2.800MHz	2.8004MHz-2.822MHz

The passband, stopband and transition-band of the first filter is set to the typical values for the CD-DA format as discussed in section 2.4.1, and is symmetrical around the middle of the spectrum. The passband, stopband and transition-band for the rest of the filters are also symmetrical around the middle of the spectrum, but have a slightly higher stopband (and passband) to ensure that the whole frequency specter of the replicas are suppressed

to the requested amount. The reason for setting a symmetrical frequency response for the filters is that it allows for the use of half-band filters, which will be discussed in a later section.

The required stopband attenuation for each of the filter stages, in accordance with the specifications of the IF, has to be determined. This is done by plotting the power spectral density (PSD) of the noise transfer function (NTF) and signal transfer function (STF) of the S-D modulator, and examine how much attenuation is required to suppress the spectral replicas under the S-D modulators noise. The STF of the S-D modulator has a LPF characteristic, and will help suppress the spectral replicas at frequencies higher than ≈ 0.5MHz. For the filters in stage 2-6, the transition-bands of the other filters in higher stages will also help suppress the spectral replicas. These filters can therefore have their requirements relaxed, provided that the complete IF filter suppresses the spectral replicas under the noise floor of the S-D modulator. The plot of the NTF and STF is shown in figure 3.1. The stopband of each filter is marked with a data cursor, showing the amount of attenuation needed to suppress the spectral replicas under the noise floor.

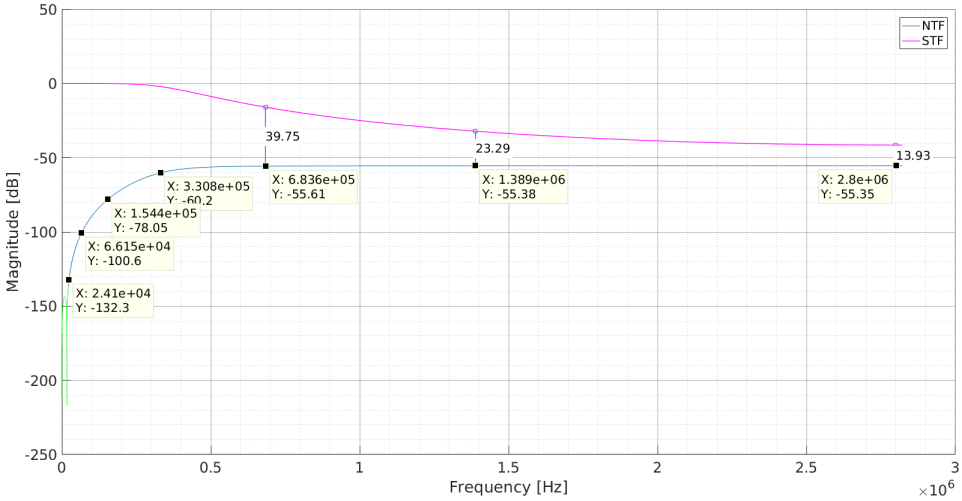

Figure 3.1: NTF and STF of S-D modulator, with the stopband edge of each filter marked.

Table 3.3 summarize the minimum attenuation requirements for each filter stage, and as can be seen here the first filter stage is set to 100dB in accordance with the specifications.

Table 3.3: Stopband attenuation summary

Filter #	1	2	3	4	5	6	7
Attenuation	100 dB	100 dB	78 dB	60 dB	40 dB	23 dB	14 dB

3.1.2.2 Filter classes

The next step is to determine the filter type, and the FIR filter was naturally chosen because of its ability to easily and precisely make linear phase filters. The seven filter stages in the IF has vastly different specifications, and this motivates the use of multiple FIR filter subclasses to make the IF more efficient. The first filter stages has the most demanding requirements with the shortest transitions-bands, and highest stopband attenuation. For these stages the half-band filter is a good choice, because of its efficient implementations in comparison to a arbitrary FIR filter of the same length. The last stages of the IF has less demanding requirements, with long transition-bands and low stopband attenuation. For these stage the CIC filter is a good choice, because of its efficient implementations with no multiplications. The use of CIC filters in the filter chain will produce an inevitable drop in the passband of the IF, because of the filter characteristics of the CIC filter. This drop in passband can not be tolerated, and is compensated using a inverse Sinc FIR filter in order to fulfill the specifications on the passband ripple. The seven filter stages of the IF is therefore roughly grouped into the following FIR filter subclasses: Half-band→Inverse Sinc→CIC.

3.1.2.3 Design and testing in Matlab

The DSP system toolbox for Matlab is used to generate and test different configurations of the three FIR subclasses, and to find the necessary stopband attenuation for each filter. The half-band and inverse sinc filters where all realized in the direct-form (D-F) FIR polyphase interpolator structure by the DSP toolbox. The final configuration of the IF, which satisfied the specifications and was deemed as an efficient and good implementation, is summarized in table 3.4

Table 3.4: Summary of the final IF configuration

Filter Stage:	1	2	3	4	5	6	7
FIR Subclass:	Half-band	Half-band	Inverse sinc	CIC	CIC	CIC	CIC
Filter Structure:	D-F FIR Polyphase	D-F FIR Polyphase	D-F FIR Polyphase	CIC	CIC	CIC	CIC
Interpolation Factor:	2	2	2	2	2	2	2
Linear Phase:	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Polyphase Length:	66	12	6	-	-	-	-
Filter Length:	131	23	12	-	-	-	-
Differential Delay:	-	-	-	1	1	1	1
Number of Sections:	-	-	-	3	2	1	1

In order to quantize the filter coefficients, a number representation is chosen for the filter. The fixed-point arithmetic is chosen for the filter, because of the low resource usage in comparison to the floating-point arithmetic. Since CIC filters does not have coefficients, only the first three filter stages needed quantization. The number of bits used to quantize each filter is found by inspecting the filter response until it is satisfactory with the least amount of bits. The number of bits used for each of the filter stages is summarized in table 3.5.

Table 3.5: Quantized coefficients

Filter Stage:	1	2	3
Bits:	22	20	16

The frequency response of the of the final IF, before and after quantization, is plotted in figure 3.2.

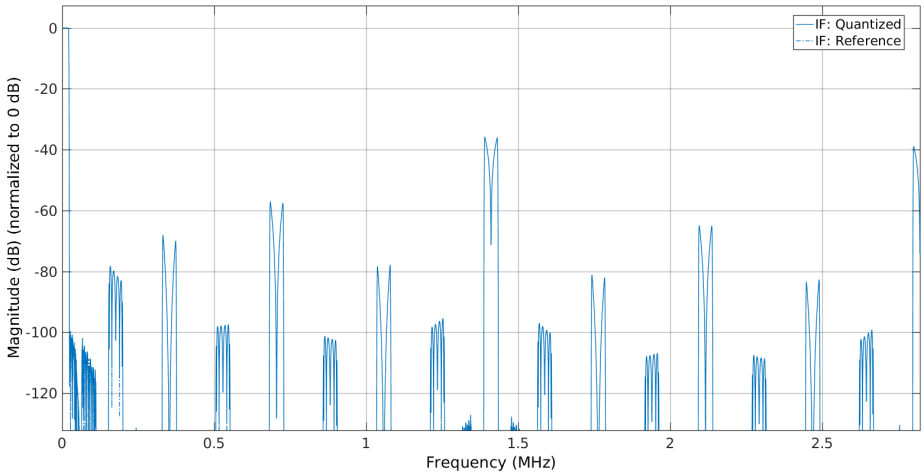

Figure 3.2: Frequency response of the final IF

The passband ripple, for the reference and quantized filter, is shown in figure 3.3, and is under 0.0007dB which satisfies the specification of the IF.

To avoid the wordlength growing to much through the cascaded IF, and to scale the wordlength to the 16 bit input of the S-D modulator, quantization of the filter stages wordlengths is implemented. To reduce the quantization noise the filter stages is implemented with double length accumulators, and only the final results of each filter stage is quantized. In order to find each wordlength a trial and error approach is used, with simulations of the IF to ensure the quantization noise do not distort the final output signal more then 1dB. The final implementations, which satisfied the specifications through simulations of several different sinusoid inputs, is summarized in table 3.6.

Table 3.6: Output wordlength of each filter stage

Filter Stage:	1	2	3	4	5	6	7
Output wordlength:	21	20	18	19	18	17	16

To avoid large errors in the case that one or more of the filters internal accumulators overflowed in the first three stages, the accumulators is implemented with saturation logic in case of overflow. This was done by setting the parameter for overflow action to saturate for each filter in the Matlab script.

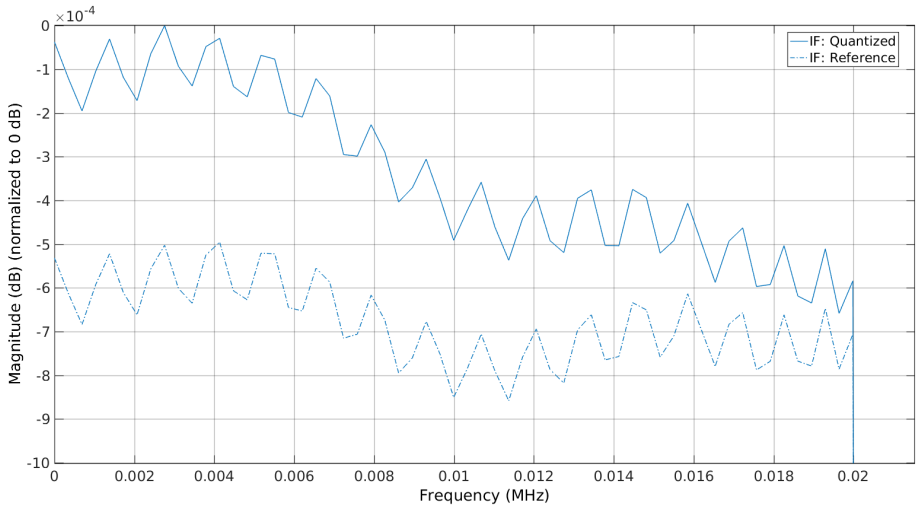

Figure 3.3: Passband ripple for final IF

3.1.2.4 Generating Verilog code

Since S-D DAC where to be implemented on a FPGA for testing, the IF had to be implemented in a HDL, and for this project the Verilog/SystemVerilog language is chosen as the HDL to use. Matlab has a HDL coder, which can generate complete Verilog code from a filter object, and this was used for to generate the Verilog code of the IF. The HDL coder can also generate test benches for the generated HDL filters, and this option was used to generate a test bench. For a IF object with polyphase structure, the implementation and optimization strategies was limited in the HDL coder. The HDL coder implements the filter in a polyphase structure, and the filter stages are effectively running on their lowest possible clock rates, because of a clock enable signal which propagates through the cascaded filter structure. This implementation is shown in the block diagram of the IF's top module in figure 3.4.

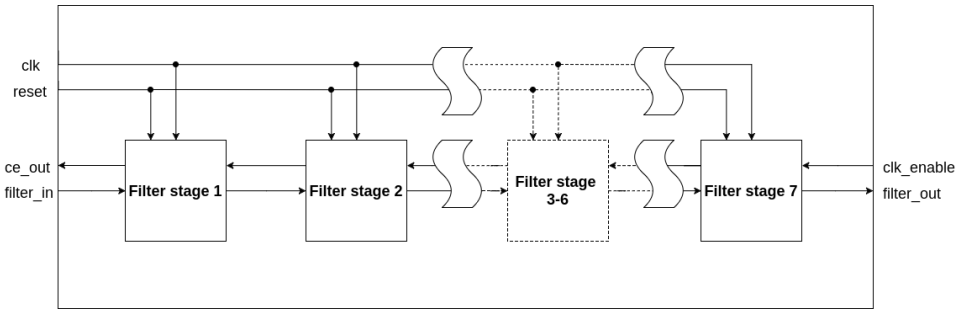

Figure 3.4: Block diagram of IF top module

Each of the filter stages in figure 3.4 is generated with the same interface, which prop-

agats the internal samples and clock enable signals through the IF. This interface is shown in figure 3.5.

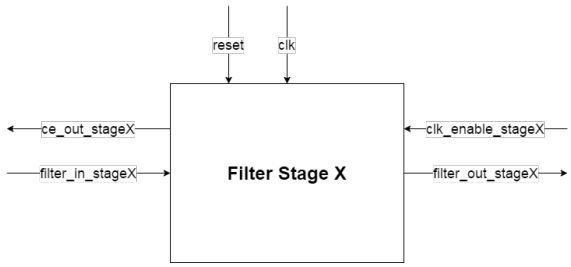

Figure 3.5: Signal interface of the filter stages

However, the HDL coder did not optimize away all the multiplications with zeros in the half-band filters, or utilize coefficient sharing for the first three filters. The implementation strategies for the filters is by default a direct/fully parallel implementation, and a MAC/serial implementation is not available. The HDL coder's implementation of the last 4 CIC filters was a standard fully parallel CIC implementation with delays and accumulators.

The generated Verilog code of the IF passed the generated test bench, and the frequency response is satisfactory when the RTL code is simulated with sine waves. The RTL code is synthesised with Vivado using the Zedboard as the target board, in order to get the utilization cost of the filter, and the results are shown in table 3.7.

Table 3.7: IF filter resource utilization on Zedboard

Resource	Estimation	Available	Utilization
Look up table (LUT)	3642	53200	6.85%
Flip flop (FF)	1789	106400	1.68%
DSP	84	220	38.18%
BUFG	1	32	3.13%

Because of the high utilization cost it is decided to try and manually optimize the RTL code. The optimization strategy is to remove the multiplications with zeros in the Half-band filters, implementing coefficient sharing for the first three filter stages, and implement the first three filters in a MAC implementation. The generated RTL code of the CIC filters was deemed to be good, and no apparent optimization is found for these filters.

3.1.2.5 RTL optimization

Using the generated Verilog code as a starting point, the first three filters is optimized. The half-band filters and the inverse sinc filter have to be optimized in different ways due to the different filters characteristics.

Half-band filters The first and second filter stage is implemented as half-band filters, which means that every even filter coefficient is zero except for the middle one, which is equal to one. Since the half-band filters is implemented in the polyphase structure, the filter is divided into two polyphase subfilters of even and odd coefficients. This means that the result from the first subfilter with even coefficients, is always equal to the delay element which is multiplied with the middle coefficient. The RTL code is therefore changed so that instead of performing any multiplication in the first subfilter, the correct delay element is just set on the output. This halved the number of multiplications per input element for each of the Half-band filters.

The second polyphase subfilter has symmetric coefficients, and this can be utilized by using the coefficients sharing implementations. The second subfilter can also be implemented in a MAC structure. With the coefficient sharing implementation, the first filter stage would need 33 clock periods for the polyphase subfilter to compute its result, and the second filter stage would need 6 clock periods with a MAC structure. Since the first filter stage is clocked 64 times faster than the output sample rate, and the second filter stage is clocked 32 times faster than the output sample rate, there is enough clock periods to complete all the computations. Therefore, the generated RTL code is change so that the second subfilter is implemented with coefficient sharing in a MAC structure.

The changes to the RTL code reduced the number of multipliers to one, and number of accumulators to two for each of the half-band filters. The number of multiplications per input sample is reduced from 132 to 33 for the first stage, and from 24 to 6 in the second stage.

Inverse sinc filter Unlike the half-band filters, all the coefficients of the inverse sinc filter is non-zero, and even though the inverse sinc filter has symmetric coefficients, the two polyphase subfilters did not have symmetric coefficients. This means that a coefficient sharing implementation for each of the polyphase subfilters is not possible, but the subfilters could share the coefficients between each other, and thereby halve the number of coefficients stored. The subfilters can also be implemented in a MAC structure, since filter stage three is clocked 16 times faster then its output sample rate, and the subfilters needs 6 clock periods to complete their computations. Therefore, the generated RTL code of the inverse sinc filter is changed so that the number of coefficients is halved, and the subfilters is implemented in a MAC structure. The changes to the RTL code reduced the number of multipliers to one, and number of accumulator to two.

Optimization results The optimized RTL code is synthesised using Vivado with the Zedboard as the target board, in order to see the utilization cost of the IF after the optimization. The results from the synthesis, and the improvements after RTL optimization is shown in table 3.8.

Besides a small increase in flip flops, the improvements in LUTs and DSP resource is substantial. Especially for the DSP resources, which went from a 40% to 3.13% utilization cost on the Zedboard. The optimized RTL code is clearly more efficient than the generated RTL code, and is therfore used as the final RTL code for the IF. In the attachment to this thesis, the final RTL code for the complete IF filter is available.

Table 3.8: IF filter resource utilization on Zedboard after RTL optimization

Resource	Estimation	Available	Utilization	Improvement
LUT	976	53200	1.83%	273.2%
FF	1912	106400	1.8%	-6.4%
DSP	5	220	2.27%	1580.0%
BUFG	1	32	3.13%	0.0%

3.1.3 Summary of interpolation filter design

The IF is designed with the specifications set in table 3.1. The IF filter is partitioned into 7 stages, where each stage has a specification set by table 3.2 and 3.3. The final configuration of the filter stages are summarized in table 3.9.

Table 3.9: Summary of the implemented IF

Filter Stage:	1	2	3	4	5	6	7
FIR Subclass:	Half-band	Half-band	Inverse sinc	CIC	CIC	CIC	CIC
Filter Structure:	D-F FIR Polyphase	D-F FIR Polyphase	D-F FIR Polyphase	CIC	CIC	CIC	CIC
Interpolation Factor:	2	2	2	2	2	2	2
Linear Phase:	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Polyphase Length:	66	12	6	-	-	-	-
Filter Length:	131	23	12	-	-	-	-
Differential Delay:	-	-	-	1	1	1	1
Number of Sections:	-	-	-	3	2	1	1
Bit depth of quantized coefficients:	22	20	16	-	-	-	-
Output wordlength:	21	20	18	19	18	17	16

The IF is implemented with fixed-point arithmetic, and the filter coefficients and the output of each filter stage is quantized. The results from quantization is summarized in table 3.9. RTL code of the filter is generated using the HDL coder in Matlab, and the generated RTL code is manually optimized for the first three filter stages. The optimization substantially reduced the utilization cost of the filter on the FPGA, and the results are summarized in table 3.8. In the attachment to this thesis, the final RTL code for the complete IF filter, which satisfies all the specifications, is available.

3.2 Design and implementation of Sigma-Delta modulator IP

The next stage in implementing a complete S-D DAC solution on a FPGA, is to make a IP of the S-D modulator from the project assignment in [6]. The S-D modulator was designed and thoroughly tested in the project assignment, and through this work a C code model of

the modulator was made, which is shown in figure 3.6.

```
#include <stdio.h>

main(){
  long u,v,x1=0,x2=0,x3=0;
  while(scanf("%ld",&u)==1){
    v = (x3+(x3<<2)) >> 21;
    if( v > 1 )
      v = 1;
    if( v <= 0 )
      v = -1;
    x2 += (x1>>2) + (x1>>4) + (x1>>6) - (v<<18) - (v<<17) - (v<<16) - (v<<14) - (x3>>10) - (x3>>11) - (x3>>13) - (x3>>14);
    x3 += (x2>>3) + (x2>>4) + (x2>>5) - (v<<17) - (v<<16) - (v<<15) - (v<<13);
    x1 += (u<<3) - (v<<18);
    printf( "%ld ",v);
  }
}
```

Figure 3.6: C code of S-D modulator from [6]

Using this C model as a foundation for implementing the S-D modulator in RTL, the main task for this implementation will consist of porting the C code into Verilog code. The S-D modulator IP will also need an external interface, able to receive data from the IF IP.

3.2.1 External interface

The generic interface of the S-D modulator is shown in table 3.10.

Table 3.10: Generic interface of the S-D modulator IP

Generic Variable	Type	Default Value	Description
DATA_WIDTH_X1	Integer	31	Defines the width of the internal *X1* varaible.
DATA_WIDTH_X2	Integer	31	Defines the width of the internal *X2* varaible.
DATA_WIDTH_X3	Integer	31	Defines the width of the internal *X3* varaible.

The signal interface of the S-D modulator is shown in table 3.11.

Table 3.11: Signal interface of the S-D modulator IP

Signal	In/Out	Description
Clock and reset:		
ck	In	Clock signal at 5644800Hz
rst	In	Synchronous set reset signal
instabiltyRst	In	Synchronous set reset signal. Asserted when S-D modulator becomes unstable
Miscellaneous signals:		
u [15:0]	In	Input samples to S-D modulator
v	Out	Output signal from S-D modulator
ck_enable	In	Enables/disables the S-D modulator

3.2.2 Design implementation

The design implementations of the S-D modulator IP, consisted of porting of the C code in figure 3.6 into SystemVerilog code. The wordlength of the internal nodes (*x1*, *x2*, *x3*) of the modulator, is by default set to 32 bits by the generic interface. This is equivalent to the *long* data type used in the C code in figure 3.6.

The external interface of the IP is shown in section 3.2.1. The interface for receiving data from the IF IP is a simple concept. The IF IP outputs a valid sample every positive clock edge when its clock enable signal is asserted. The data transfer between the two IPs is implemented by connecting the S-D modulator to the same clock enable signal, and designing the S-D modulator to read a sample ever positive clock edge when its asserted. This enable signal also functions as a start/stop signal for the S-D modulator IP.

Since the S-D modulator can potentially become unstable, it was decided to implement an extra reset signal for the S-D modular IP. This signal could be used by an external IP, which monitors the S-D modulator, to safely reset the modulator IP if it becomes unstable.

The complete RTL implementation of the S-D modulator can be found in the attachment to this thesis.

3.2.3 Testing and verification

The C model of the S-D modulator in figure 3.6, was extensively simulated and tested in the project assignment in [6]. Thus, if the RTL code produce the exact same output as the C code model, the RTL implementation could be assumed to be correct. The testing environment for this verification strategy is shown in figure 3.7.

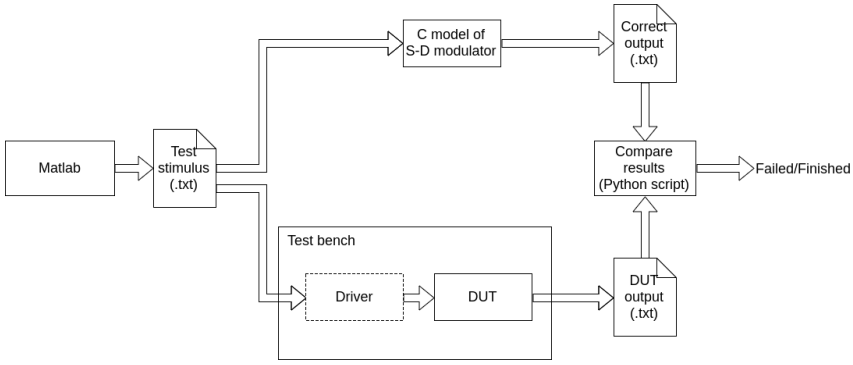

Figure 3.7: Testing environment for S-D modulator IP

Matlab is used to generate sine wave stimulus, which is written to a text file. A SystemVerilog test bench is made, which read the stimulus text file and then simulates the S-D modulator device under test (DUT) with these samples. The test bench also drive all the input signals on the DUT, and reads the output signal. The output from the S-D modulator is written to a text file. The C model of S-D modulator is also simulated with the same stimulus text file, and its output is also written to a text file. To compare these two text files, a simple Python script was made. The Python script prints out a success message if

the samples in the two text files are equal, or stops and prints out the number of the first line where they differ. In the attachment to this thesis, the scripts and test bench files used in this testing environment are available.

This verification environment is used with several different input stimulus, running the S-D modulator both in normal mode and into unstable mode. The S-D modulator IP is found to be equivalent to the C model in all the test cases, and is therefore verified to be working as expected.

3.3 Design and implementation of DAC IP

The two digital blocks of the S-D DAC implementation is designed and tested separately in the previous sections. In order to check that the IF and S-D modulator IPs functions properly together, a simple wrapper IP is made for the two IPs, called DAC. The DAC IP connects the two IPs together, and implements an external interface for the two.

3.3.1 External interface

The signal interface of the DAC IP is shown in table 3.12.

Table 3.12: Signal interface of the DAC IP

Singal	In/Out	Description
Clock and reset:		
ck	In	Clock signal at 5644800Hz
rst	In	Synchronous set reset signal
sdInternalRst	In	Synchronous set reset signal. Asserted when S-D modulator becomes unstable
Miscellaneous signals:		
dataIn [15:0]	In	Input samples to IF IP
ce_out	Out	Clock enable out signal. Asserted when IF reads data from the dataIn bus.
ck_enable	In	Enables/disables the IF and S-D modulator IPs
dataOut	Out	Output signal from S-D modulator

3.3.2 Design implementation

The DAC IP is a simple wrapper IP for the IF and S-D modulator IP. No additional logic is needed, since the S-D modulator IP is designed to be directly connected to IF IP. The block diagram in figure 3.8 show how they are connected together.

In the attachment to this thesis, the complete RTL implementation of the DAC IP is available.

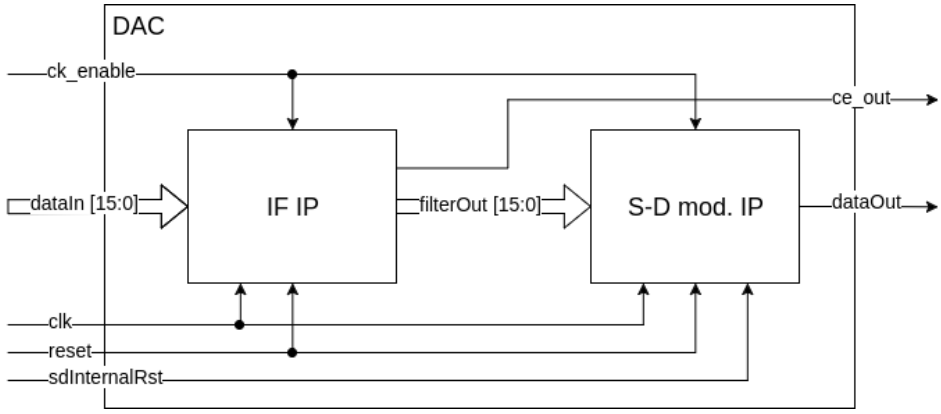

Figure 3.8: Block diagram of DAC IP

3.3.3 Testing and verification

Since the IF and S-D modulator IPs are thoroughly tested and verified separately, the verification consisted only of checking that they function properly together. The verification is done by producing input stimulus with Matlab, and making a simple SystemVerilog test bench, which drive the input signals and writes the stimulus to the DAC IP. The data out of the DAC IP is written to a text file, which can be analyzed with Matlab. The wave diagrams of the simulated DAC IP is manually checked, and the fast Fourier transform (FFT) of the output is plotted to verify the functionality. In figure 3.9 the spectrum of the DAC IP with a 3kHz sin wave as stimulus is shown. The sine wave has a bit depth of 16 bits, and an amplitude of -2.5dBFS.

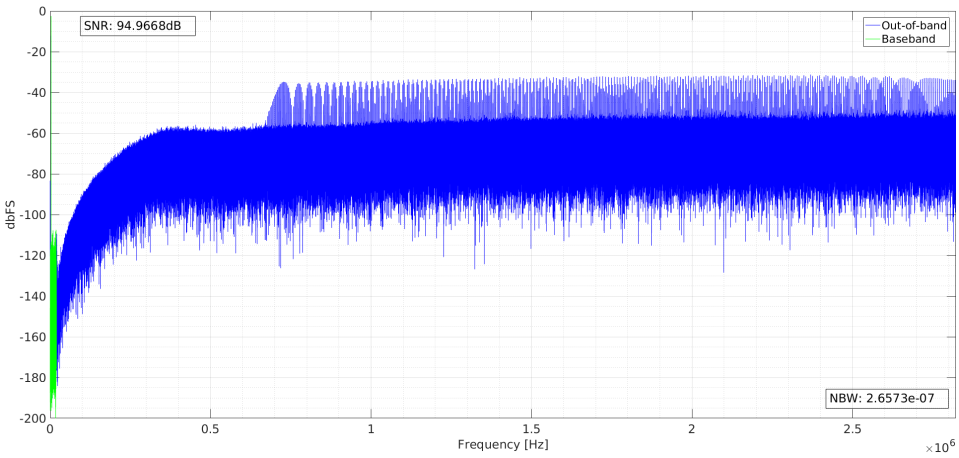

Figure 3.9: Spectrum of DAC IP output

In figure 3.10 the spectrum is zoomed in at baseband. The spectrum in figure 3.9 and

3.10 shows the expected behavior of the DAC IP, and this is also the case throughout the testing and checking of wave diagrams. The DAC IP is therefore verified to have a correct functionality.

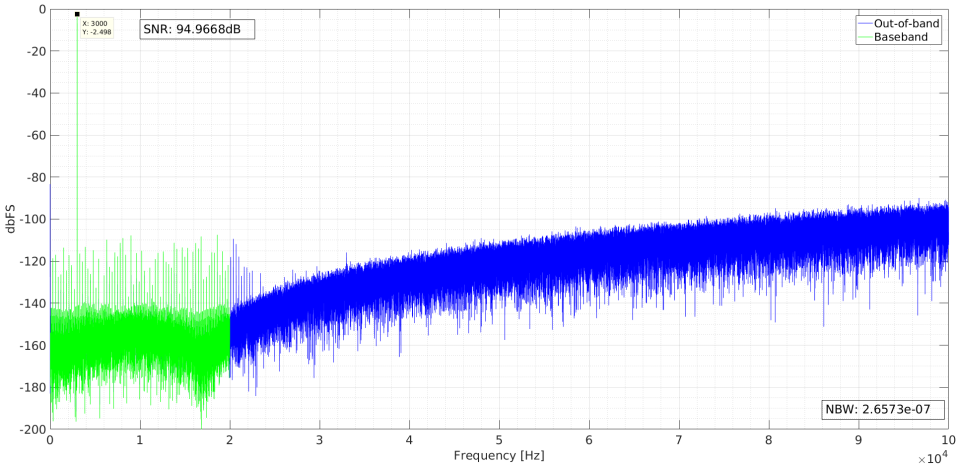

Figure 3.10: Spectrum of DAC IP output zoomed in at baseband

3.4 Design and implementation of Sigma-Delta DAC IP for FPGA

The digital parts of the S-D DAC solution is designed and verified to be working together in the DAC wrapper IP. The next step in implementing this DAC IP on the Zynq-7000 SoC, is to design an IP which can be implemented on the Zynq-7000's PL. This IP will manage the digital logic and interfaces needed to control and stream samples to the DAC IP on the PL. The next sections outline the design and implementations of this S-D DAC IP.

3.4.1 Specifications

The main aim for the implementation on the Zynq-7000 SoC, is to utilize the PS to control the S-D DAC IP with a control register on the IP, and to stream samples to the IP from the DDR3 memory or other interfaces using the PS. This requires a bus interface between the PS and PL, and control logic on the S-D DAC IP.

To communicate between the PS and PL, the ARM AMBA AXI4 bus protocols can be used. The AMBA AXI4 protocol consists of three sub interfaces; the AXI4, AXI4-Lite, and AXI4-Stream. The AXI4-Lite interface is made for simple memory mapped communication, and is a good protocol to use for reading and writing to a simple control register on the S-D DAC IP. The AXI4-Stream is made for high speed streaming of data, and supports direct memory access (DMA) transfers. This is a good protocol to use for

streaming samples from the PS to the S-D DAC IP. The AXI4-Lite and AXI4-Stream were therefore chosen as bus interfaces for the S-D DAC IP.

Since the S-D modulator can potentially become unstable, it is decided to add a stability control unit IP. This IP should monitor the output of the modulator, and reset the modulator if it becomes unstable. To handle underflow events, an underflow interrupt signal, which can be connected to the PS, should also be implemented. The underflow signal should be asserted when a underflow of samples on the AXI4-Stream interface occur, and the DAC should be stopped when this occur.

3.4.2 External interface

The external signal interface for the SD_DAC top module is determined from the specifications, and is explained in table 3.13.

Table 3.13: Singal interface

Signal	In/Out	Description
AXI4-Lite Slave bus interface:		
AXI4-Lite bus	-	See [15] for descriptions
AXI4-Stream bus interace:		
s00_axis_aclk	In	Clock signal
s00_axis_aresetn	In	Async reset signal
s00_axis_tvalid	In	Indicates that the source is ready to send data
s00_axis_tdata [31:0]	In	Frame data is transmitted across this bus
s00_axis_tready	Out	Indicates that the sink is ready to accept data
Miscellaneous signals:		
dacOut	Out	Output from S-D modulator
instabilityReset	Out	Internal instability reset from stability control unit IP
Interrupt signals:		
irqUnderflow	Out	Asserted when a underflow of samples occur

The external register interface for the SD_DAC top module is explained in table 3.14.

Table 3.14: Register interface

Address	Name	Bit	Reset Value	Type	Description
Control:					
0x0	ctrlReg	2	0	R/W	Control register for SD_DAC IP. Bit 0: Start/stop the DAC module. Bit 1: Enable stability control unit.

3.4.3 Design implementation

From the specifications in section 3.4.1, the SD_DAC IP is designed, and a simplified block diagram of the IP is shown in figure 3.11. For the sake of clarity are some signals omitted and some names changed in the block diagram, compared to the final RTL implementation.

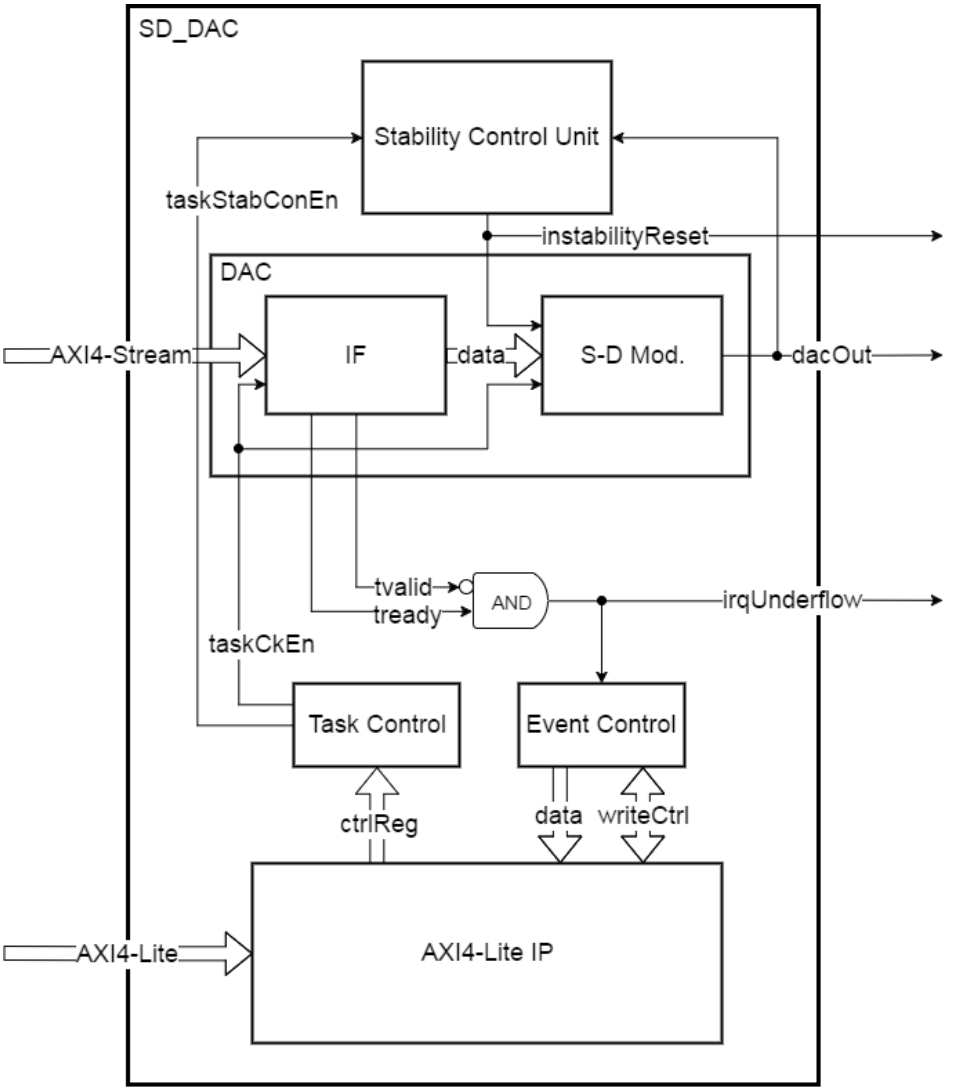

Figure 3.11: Simplified block diagram of SD_DAC IP

In the next sections a general description of the different sub IPs and control logic will be explained. In the attachment to this thesis, the complete RTL implementation of the SD_DAC module with all the sub modules are available.

3.4.3.1 AXI4-Lite IP

The AXI4-Lite IP is a generated IP from the Vivado design suite, and is manually modified for this SD_DAC IP. The generated IP manages the AXI4-Lite slave protocol from an AXI4-Lite master, and includes 4 read/write memory mapped registers of 32 bits each. One of these registers are used as a control register for the SD_DAC IP. The generated IP is modified so that the control register is outputted to the task control IP, which is managing the tasks set by the control register, as shown in the block diagram in figure 3.11. In order to make it possible for internal writes to the control register, in case of events in the SD_DAC IP, a simple handshake protocol for writing to the control register is implemented in the AXI4-Lite IP. The internal write requests is managed by the event control IP as shown in figure 3.11.

3.4.3.2 Task control IP

The task control IP manages the tasks set by the control register in the AXI4-Lite IP. The control register is inputted to the task control IP, and the task signals are set high/low if the corresponding bits in the control register is set. The task control IP only controls two tasks, the task for enabling the DAC IP, and the task enabling the stability control unit. This is shown in the block diagram in figure 3.11.

3.4.3.3 Event control IP

The event control IP manages the internal events in the SD_DAC IP, and updates the control register in the AXI4-Lite IP accordingly when an internal event occur. The internal writes are done with a simple handshake protocol, and are initiated when an internal event occurs. In the block diagram in figure 3.11, the handshake signals are marked *writeCtrl* and the write data is marked *data*. The only event in the SD_DAC is the underflow event, and it causes the event control IP to stop the DAC IP by writing to the control register.

3.4.3.4 Stability control unit IP

The stability control unit IP monitors the output of the S-D modulator IP, and asserts the *instabilityReset* signal, which resets the S-D modulator, if the modulator becomes unstable. The stability control unit counts the number of consecutive ones or zeros, and asserts the internal instability reset when the number of consecutive ones or zeros passes a set limit. The limit is set by a parameter and is by default set to 30. The number of clock periods the reset is asserted is also set by a parameter, and is by default set to 6 clock periods. The stability control unit is enabled and disabled by writing to the control register, and is by default off. The instability reset signal is directly connected to the S-D modulator IP, and is also ported out of the SD_DAC IP for debugging purposes.

3.4.3.5 DAC IP

The DAC IP consists of the IF IP and the S-D modulator IP, which are connected together, and make up the digital parts the S-D DAC. The instability reset from the stability control

unit IP, is connected to S-D modulator IP, and synchronous resets the modulator if it becomes unstable. The IF and S-D modulator IPs are started and stopped by the *taskCkEn* signal from the task control IP. The AXI4-Stream interface is connected to the IF IP. The AXI4-Stream is a simple interface, which asserts the *tvalid* signal when the data on the *tdata* bus is valid. The IF reads the *tdata* bus and asserts *tready*, but it does not check if the *tvalid* signal is asserted. An underflow occurs when the *tready* signal is asserted when *tvalid* is low, triggering the interrupt signal *irqUnderflow*. Figure 3.12 shows a timing diagram of an AXI4-Stream data transfer, and an underflow interrupt generation.

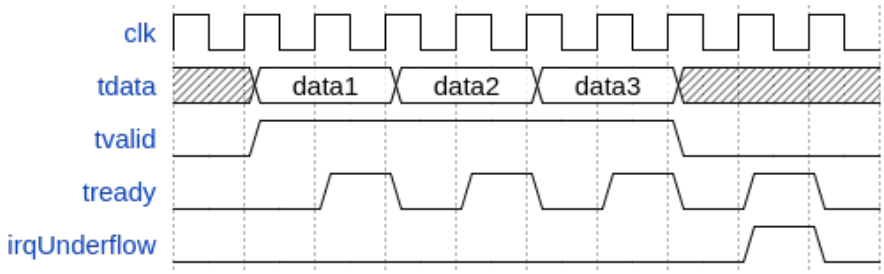

Figure 3.12: Timing diagram of AXI4-Stream transaction and interrupt generation

The underflow interrupt triggers an internal event, which in turn stops the DAC IP, and can alert the PS that an underflow has occurred. This could also be used by the PS as a finish signal when all the samples were streamed to the DAC. The block diagram in figure 3.11 shows how the DAC IP, and the interrupt is connected.

3.4.4 Verification

The DAC IP is verified in the previous sections, and the AXI4-Lite IP is, except for minor modifications, entirely generated by Vivado. Thus, the most complex parts of the SD_DAC, the DAC IP and the AXI4-Lite IP, have been tested and verified. The RTL verification of the complete SD_DAC is therefore limited. Some very basic test benches is made for the other sub modules for functional testing. The functional testing of the complete SD_DAC IP will be done when the IP is implemented on the Zynq-7000, with the use of debugging cores.

3.5 Complete FPGA implementation

The final stage in implementing a complete solution for a S-D DAC on the FPGA, is to integrate the SD_DAC IP on the FPGA. The ZedBoard development board is the target platform for this project. The ZedBoard is based around a Xilinx Zynq-7000 All Programmable SoC. Since the Zynq-7000 is a Xilinx product, the development platform for this project is the Vivado Design Suite for the PL design, and the Xilinx Software Development Kit for the PS design. The next sections examines the complete design and implementation of the S-D DAC on the ZedBoard.

3.5.1 Specifications

The main aim for the design is to implement the SD_DAC IP from section 3.4, on the PL, and use the PS to control and stream samples to the SD_DAC. The output of the SD_DAC IP should be connected to a general-purpose input/output (GPIO) pin on ZedBoard, which can be used for measurements and for connecting it to a audio system. This requires a framework around the SD_DAC on the PL connecting it to the PS, and software on the PS to run it. The framework must consist of AXI4 interfaces in order to be compatible with the PS.

The SD_DAC IP must run at a clock frequency of $f_c = 5.644800$MHz to produce the correct frequency spectrum. The AXI4 interfaces from the PS will run at a higher clock frequency, which means there will be two asynchronous clock domains on the PL, requiring the implementations of logic for safe clock domain crossing.

The software implementation on the PS must control the SD_DAC IP, and stream samples to the SD_DAC IP fast enough to avoid underflow.

3.5.2 Programmable logic design

The IP integrator in Vivado is used to design the complete PL implementation. The IP integrator uses a block design interface, which makes it easy to integrate IPs from the Vivado IP catalog and custom made IPs. The block diagram from the IP integrator, of the complete PL implementation, is shown in figure 3.13. The two main IP blocks in figure 3.13 are the *processing_system7_0* and *SD_DAC_0*. The *SD_DAC_0* IP block in figure 3.13 is the designed SD_DAC IP from section 3.4, and the *processing_system7_0* IP is the Zynq-7000's PS core. The other IPs in figure 3.13 are for the bus interfaces between the two main blocks, for clock and reset generation and for debugging.

3.5.2.1 Clock and reset

The PL design in figure 3.13 contains two clock domains, and two IP's for safely asserting the reset signal in the two domains.

One clock domain is for the SD_DAC IP, and runs at a clock frequency of 5.645MHz. The clock is generated by the *clk_wiz_0* IP in figure 3.13, connected to the external 100MHz oscillator on the ZedBoard through the external *sys_clock* connection. The SD_DAC IP should run at a clock frequency of 5.644800MHz, but 5.645MHz is closest frequency the clock generator can produce. The error is less than 0.001dB, and will have a minor affect on the frequency spectrum.

The other clock domain is for the AXI4 bus systems connected to the PS core, and runs at a clock frequency of 50MHz. The clock is generated by the PS core, and is set 50MHz to avoid the bus systems becoming bottlenecks for the data streaming. The AXI4 bus systems are clocked almost 9 times faster then the SD_DAC IP, and should run fast enough to avoid underflow problems.

The two IPs *proc_sys_reset_0* and *rst_ps7_0_100M* in figure 3.13, controls the reset signals for the two clock domains. The IPs are running separately in the different clock domains, both connected to the reset signal from the PS core. The IPs assert the reset signals at power *on*, or when the reset signal from the PS core is asserted.

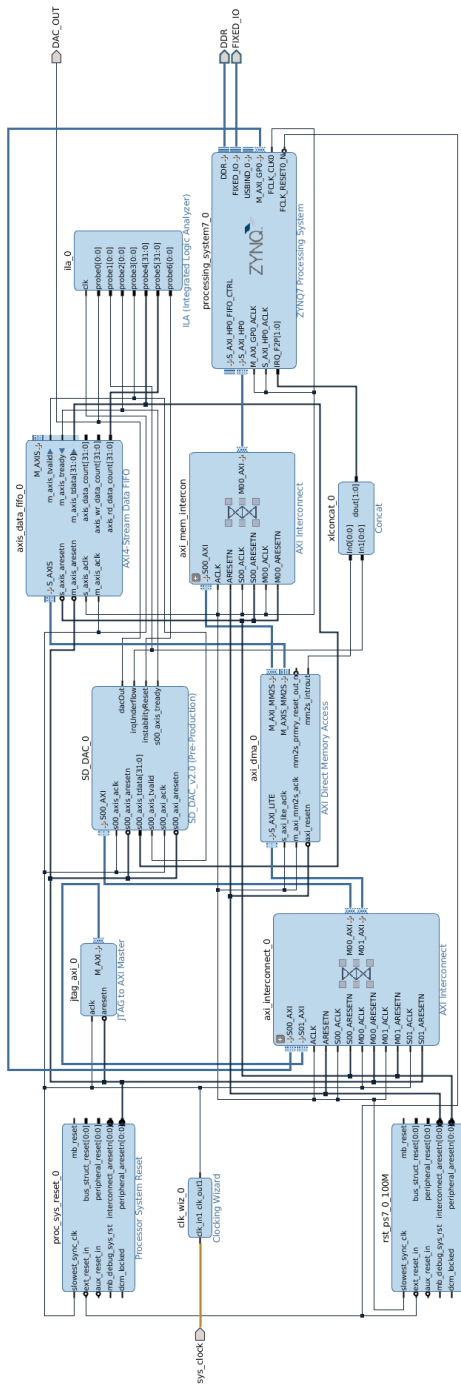

Figure 3.13: Block diagram of the complete PL implementation

3.5.2.2 Processing system core configuration

The Zynq-7000 PS IP in figure 3.13 has two external connections called *DDR* and *FIXED_IO*. The *DDR* is for connecting the Zynq-7000 PS to the 512MB DDR3 memory on the ZedBoard. The *FIXED_IO* is for connecting I/O peripherals in the PS to the ZedBoard. The PS is configured with the UART, USB and Ethernet I/O peripherals to enable communication with a personal computer (PC) for data streaming and debugging. To enable bus communication with the PL a AXI4 master port, and a high performance AXI slave port is enabled on the PS. The other PS configurations are either disabled or set to default.

3.5.2.3 Data streaming

One of the main goals for the design is to enable streaming of data from the PS to the S-D DAC IP. This is solved using a DMA IP in the PL, which can be configured by the PS to start DMA transfers from the DDR3 memory to the S-D DAC IP. The DMA IP is connected to the PS through the high performance AXI slave interface on the PS, and outputs the data on an AXI4-Stream bus. The high performance AXI slave interface from the PS cannot be directly connected to the DMA IP, and therefore it goes through an AXI4 interconnect IP, before connecting to the DMA IP. Since the S-D DAC IP is in an asynchronous clock domain, the AXI4-Stream bus from the DMA IP cannot be directly connected to the AXI4-Stream interface on the S-D DAC IP. To ensure a safe clock domain crossing an AXI4-Stream data first in first out (FIFO) IP is used, which can handle asynchronous clock domain crossing of data. The output of the AXI4-Stream data FIFO is connected to the AXI4-Stream interface on the S-D DAC IP, completing the bus system for the data streaming on the PL.

The DMA IP can be configured to assert an interrupt signal when the DMA transfer is finished, and this signal is connected to the PS core along with the underflow interrupt signal from the S-D DAC IP. The final implementation is shown in figure 3.13.

3.5.2.4 AXI4-Lite bus system

The second goal for the design is to control the S-D DAC IP on the PL from the PS through a control register. The S-D DAC IP is designed with an AXI4-Lite slave interface and a control register for this purpose. The DMA IP also has an AXI4-Lite slave interface for its control and status registers, and should also be connected to AXI4-Lite bus system. Since the S-D DAC IP is in an asynchronous clock domain, the AXI4-Lite bus also needs a safe clock domain crossing between the PS core and S-D DAC IP.

The *axi_interconnect_0* IP in figure 3.13, takes care of both safe clock domain crossings and all the interconnects of the AXI4-Lite bus system. Thus, the AXI4 master port from the PS core, and the AXI4-Lite slave interfaces from the S-D DAC IP and DMA IP are connected to the interconnect. The base addresses for the different IP's are generated by Vivado, completing the AXI4 bus system.

3.5.2.5 Debugging cores

The PL design in figure 3.13 contains two debugging IPs for testing and verification of the design. The two IP's are called *jtag_axi_0* and *ila_0*.

The *jtag_axi_0* IP enables reading and writing to the AXI4 bus system on the PL through the joint test action group (JTAG) interface on the ZedBoard. The read and write operations are done using the commando line in Vivado when the design is running on the ZedBoard.

The *ila_0* IP is an integrated logic analyzer (ILA) core, which can monitor any signal on the PL when the design is running on the ZedBoard. The probed signals are shown in a wave diagram in Vivado, and is connected to the PC through the JTAG interface on the ZedBoard. The ILA core is used to monitor signals from the S-D DAC IP and the AXI4-Stream data FIFO IP.

3.5.2.6 DAC output

The output signal from the DAC is connected to a GPIO pin on the ZedBoard through the external connection point *DAC_OUT* in figure 3.13. The digital pad from PL is configured with the LVTTL I/O stander, a drive of 24mA and a fast slew rate, providing a maximum current drive. The digital pad is connected to the *XADC-GIO2* GPIO pin on the ZedBoard.

3.5.2.7 Synthesis and implementation

The synthesis and implementation of the complete PL design were successfully completed without critical warnings, hence all timing constrains and Vivado design rules are fulfilled. The post implementation utilization of Zynq-7000 is shown in table 3.15.

Table 3.15: Post implementation utilization

Resource	Estimation	Available	Utilization
LUT	5597	53200	10.520677%
LUTRAM	648	17400	3.724138%
FF	9364	106400	8.800752%
BRAM	6.5	140	4.642857%
DSP	5	220	2.2727273%
IO	2	200	1.0%
BUFG	4	32	12.5%
MMCM	1	4	25.0%

Table 3.15 shows that the total utilization is well below the maximum limit of the Zynq-7000's PL.

3.5.3 Processing system design

The generated bit stream of the PL design in section 3.5.2 is exported to the Xilinx software development kit (SDK), where the PS design is done. The design goal for the PS design is to control and stream samples to the S-D DAC IP using DMA. This will only require a single-thread process and some basic features from the PS, so the standalone or bare-metal OS is sufficient for this task. The standalone OS will initialize the ZedBoard, and start executing a main function which can be programmed in the SDK. The control and

stream functionality of the PS is programmed in this main function in the programming language C. In the next sections the S-D DAC driver and main function are explained. In the attachment to this thesis, the complete C code of the driver and main function are available.

3.5.3.1 S-D DAC driver

A simple driver for the S-D DAC IP is written in C. The driver consists of functions which starts and stops the S-D DAC, and configures the stability control unit. These functions input the base address for the S-D DAC IP, and set the correct bits in the control register depending on the action. They also read back the register, and checks if the bits were actually set. The driver functions return a success flag if the bits were successfully set, or a failure flag if the bits were not set.

3.5.3.2 Main function

The goal for the design of the main function is to use the S-D DAC driver to control the S-D DAC IP, and stream samples to the S-D DAC using interrupt based DMA transfers. Using interrupt based DMA transfer instead of polling, will free up the processor for other operations during the DMA transfer. The DMA IP raises an interrupt signal when the transfer is finished or if an error occurs. Thus, an interrupt handler function is implemented, handling the interrupt events from the DMA IP. When designing the PL in section 3.5.2, the PS was configured with an UART-USB bridge. This can be used to print info and debugging messages to a console on a host PC connected with an USB cable, and this is done throughout the code. The main function is designed using example code for an interrupt based DMA transfer from the SDK as a starting point. The flow chart for the pseudo code of the main function and interrupt handler, are shown in figure 3.14.

The main function starts by printing a start message to the console on the host PC. Next the DMA IP is configured with interrupt, and the PS is configured to start the interrupt handler function when the DMA IP raise the interrupt signal. If the configurations are successful, the S-D DAC IP is configured with the stability control unit and started. If successful, the processor initiate a DMA transfer of the data samples in the DDR3 memory to the S-D DAC IP in the PL. The data samples are of a 3kHz sine wave, and are allocated in a header file as a C array. The data samples are loaded into the DDR3 memory alongside the main function when the code is launched on the ZedBoard. The main function then continues into a loop, checking if a DMA error flag or DMA finished flag is raised by the interrupt handler. If no flags are raised it goes into a **NOP**, or no operation process, where no actions are done, before continuing the loop. The **NOP** process represents a time slot where the processor is available for other operations while waiting for the DMA transfer to finish. This time slot is not utilized in this version of the main functions, but can be utilized in a later version for other operations. If the DMA transfer successfully finished, the loop initiates a new DMA transfer of the same data samples, giving a continuous sine wave on the output of the S-D DAC. If an error occurs in any stages of the main function, an error message is printed to the console on the host PC, and the main function is terminated.

The interrupt handler function in the flowchart in 3.14 is initiated by the PS when the interrupt signal from the DMA IP is raised. The interrupt handler function start by

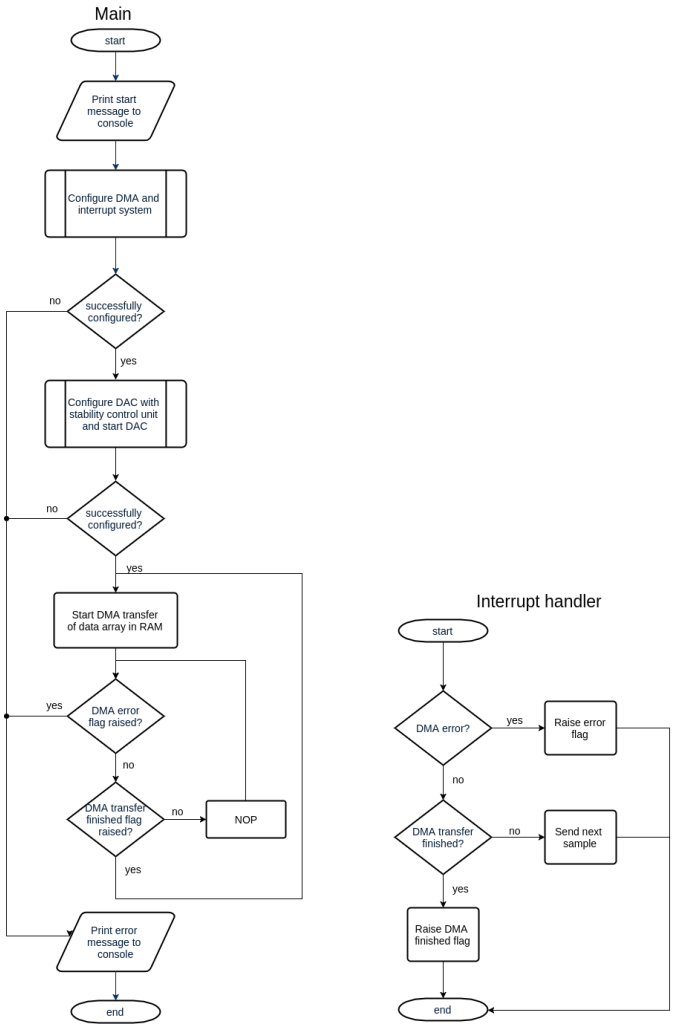

Figure 3.14: Flowchart of main function and interrupt handler

checking if a DMA error has occurred. If an error has occurred, it raises a DMA error flag and terminates the interrupt handler. If no error occurred it checks if the DMA transfer is finished. The data samples in the DDR3 memory, are divided into smaller sections which are separately transferred with DMA. If not all the sections are transferred, the interrupt handler initiate the DMA transfer of the next section and terminates. If all the sections are transferred, the interrupt handler raises the DMA finished flag and terminates.

The main function continuously stream the data samples of the sine wave to the S-D DAC IP until an error occurs. This produces a continuously playing sine wave on the output of the S-D DAC.

3.5.4 Testing and verification

The testing and verification are done using the debugging cores in the PL design. In the Vivado design suite, the probed signals on the ILA IP are shown in a timing diagram. Using this timing diagram, the functionality of the PL design is checked as the designed PS code is running. The JTAG IP is used to read and write to the S-D DAC IP's control register from the command line in the Vivado design suite, and checking the response of the S-D DAC IP. This verified the functionality of the control register. The data samples from the DDR3 memory were also verified to be successfully transferred to the S-D DAC IP. The complete FPGA implementation is working as expected, and the output of the S-D DAC IP can therefore be measured. The measurements of the S-D DAC output is shown and discussed in a later chapter.

3.6 Design and implementation of PWM test IP

In order to compare a PWM DAC to the designed S-D DAC, a PWM test IP is designed and implemented in RTL code. Due to time limitations only the simplest PWM scheme, the single sided PWM scheme with no predistortion, is chosen. Thus, the test IP will generate severe harmonic distortion in the signal baseband, and have poor audio performance. However, this modulation scheme is easy to implement on a microcontroller, and is therefore in some cases used for audio application. Comparing this PWM scheme to the S-D DAC IP is therefore relevant for the use on a microcontroller. The PWM IP can give an estimation of the power consumption and audio performance, and this can be compared to the S-D DAC IP.

3.6.1 Specification

The specification for the PWM IP is to implement a single-sided PWM scheme with no predistortion in RTL code. The PWM IP should be able to run samples with different bit depths, and the bit depth should generically set by a parameter in the RTL code.

3.6.2 External interface

The generic interface of the PWM IP is shown in table 3.16.

Table 3.16: Generic interface PWM IP

Generic Variable	Type	Default Value	Description
DATA_WIDTH	Integer	8	Defines the width of the data in, and the internal counter.

The signal interface of the PWM IP is shown in table 3.17.

Table 3.17: Singal interface PWM IP

Signal	In/Out	Description
Clock and reset signal:		
ck	In	Clock signal
arst	In	Async reset signal
Miscellaneous signals:		
dataIn [DATA_WIDTH-1:0]	In	Data samples in
ce_out	Out	Clock enable out
dataOut	Out	Modulated PWM signal out

3.6.3 Design implementation

The PWM IP is implemented by comparing the input data sample to a sawtooth signal from an internal binary counter. This is essentially the same implementation method as described in section 2.2. The binary counter runs continuously when the reset signal is low. The *ce_out* singal is asserted when the data samples on the *dataIn* bus is read. The PWM IP is designed so the wordlength of the data in signal and the internal binary counter is generically set by the *DATAWIDTH* parameter. In the attachment to this thesis, the RTL code of the PWM test IP is available.

3.6.4 Testing and verification

A simple testbench for the PWM IP is made, testing the PWM with sine waves of different bit depths. The functionality of the PWM IP is verified by manually studying the timing diagram from the simulation.

In order to check the audio performance of the PWM IP, the IP was simulated using a full scale 3kHz sine wave with a bit depth of 8 bits and a sample rate of 44.1kHz. The ideal output spectrum of the PWM module is computed in Matlab, and is shown in figure 3.15.

As anticipated, the spectrum in figure 3.15 shows severe harmonic distortion. The THD is -19.4414dB, heavily impacting the SNR of only 19.4465dB. This is far from the theoretical achievable SNR, which is calculated in (3.1).

$$\text{SNR} = 6.02 \cdot 8 + 1.76\text{dB} = 49.92\text{dB} \tag{3.1}$$

If the THD is excluded, the in band noise is equal to -48.9281dB, shown in figure 3.15. This is close to the theoretical limit for a 8 bit sample, and demonstrates that the harmonic distortion of the PWM is limiting the SNR. The ideal single-sided PWM modulation with no predistortion, cannot compete against the S-D DAC IP in audio performance.

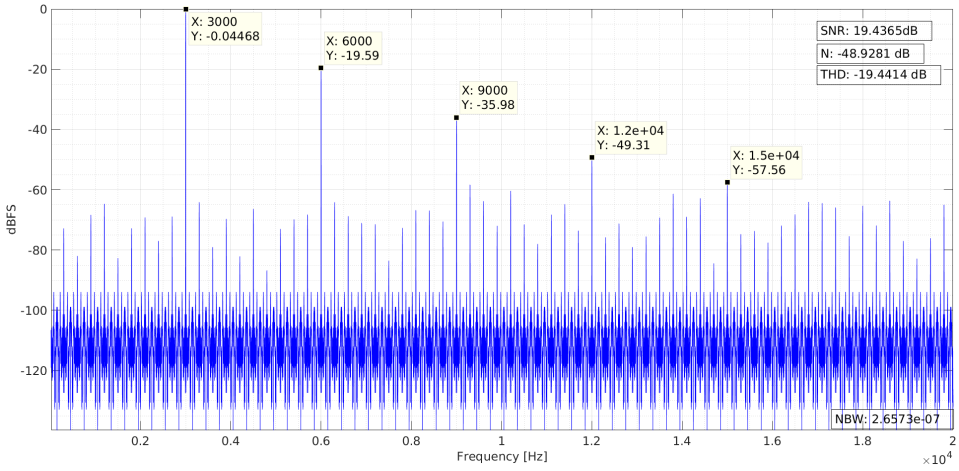

Figure 3.15: Ideal spectrum of the PWM IP with a 3kHz sine wave

Chapter 4

Results

4.1 Sigma-Delta DAC measurements

After implementing the S-D DAC on the ZedBoard, the next step is to measure its audio performance. The strategy for testing the S-D DAC is to measure THD and THD+N for a range of frequencies, amplitudes, and bit depths. The THD performance is especially interesting since nonlinearities produced by the 1-bit DAC, which in this case is the digital I/O pad on the Zynq-7000, will appear as harmonic distortion. The S-D DAC is also hooked up to a audio system and tested with audio samples. To measure the audio performance, a spectrum analyzer and an oscilloscope are used. The spectrum analyzer is a R&S© FSV Signal and Spectrum Analyzer [16], and the oscilloscope is a R&S© RTM2000 Digital Oscilloscope [17]. Figure 4.1 shows the measurement setup.

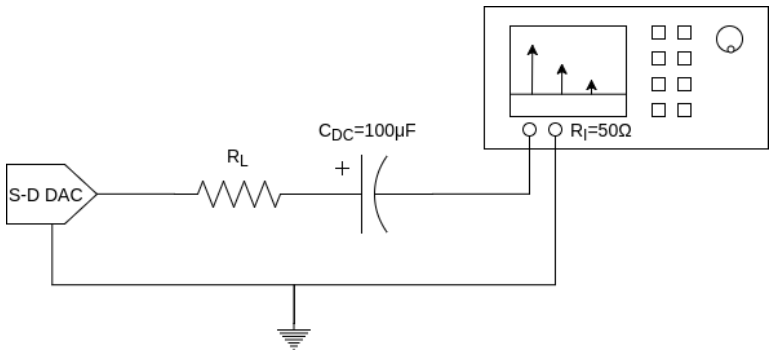

Figure 4.1: Measurement setup

The GPIO pin from the ZedBoard, connected to the output of the S-D DAC on the PL, is connected to a load resistor R_L. The R_L is used to limit the input power to the spectrum analyzer, and to find the value of the output load where the digital pad has the best THD performance. The electrolytic capacitor $C_{DC} = 100\mu$F, in series with the DAC output, is a DC block. Since the digital pad is switching between 0V and its full scale

value $V_{FS} = 1.8$V, the output signal contains a DC component. This DC component is removed by the DC block, because the spectrum analyzer dose not tolerate DC components on its input. The input impedance R_I of the spectrum analyzer is 50Ω, the same input impedance used on the oscilloscope. The measurement setup in figure 4.1 is valid for both the oscilloscope and spectrum analyzer measurements. The next section goes through all the measurements of the S-D DAC implementation on the ZedBoard.

4.1.1 THD performance of the digital pad

The THD performance of the digital pad versus the output load is measured, to find the output load with the best performance, which will be used for the succeeding measurements. The measurement is performed by running a 3kHz sine wave continuously on the S-D DAC, while testing a range of output loads. The sine wave has a bit depth of 16 and an amplitude of -2.5dBFS. The THD is calculated by the spectrum analyzer, using its embedded THD measurement function. The results are plotted in figure 4.2, each dot representing a measurement.

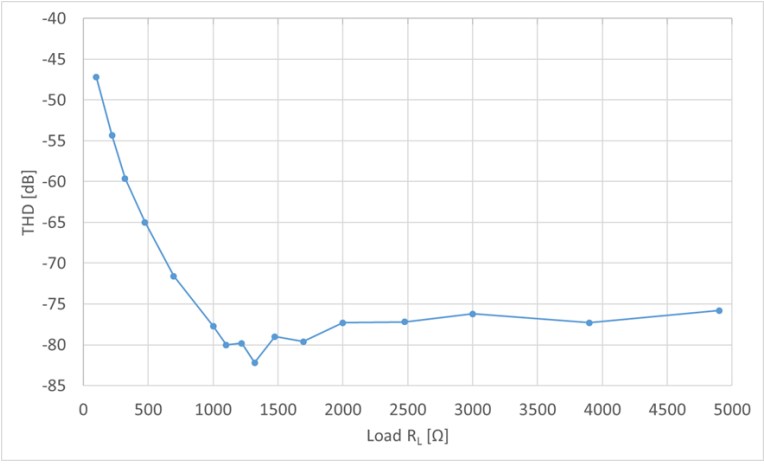

Figure 4.2: THD versus output load

Figure 4.2 shows a sharp decline in the THD from the start at 100Ω to 1100Ω. The THD is reduced from -47.2dB to -80dB between the two points. The THD then abruptly level out, before slowly rising again from 1700Ω to 5000Ω. The best measured performance is a THD of -82.2dB at an output load of 1320Ω. This is equivalent to an ENOB of 13.4 bits. Thus, the digital pad limits the overall possible performance of the S-D DAC to about a ENOB of 13 bits.

In order to further examine the pad's performance, the GPIO pin is connected to the oscilloscope to see the time domain response. The S-D DAC is running the same sine wave as before, and the best performing output load of $R_L = 1320\Omega$ is connected. Figure 4.3 shows approximately 6 clock periods of the output in the time domain with four transitions between high and low.

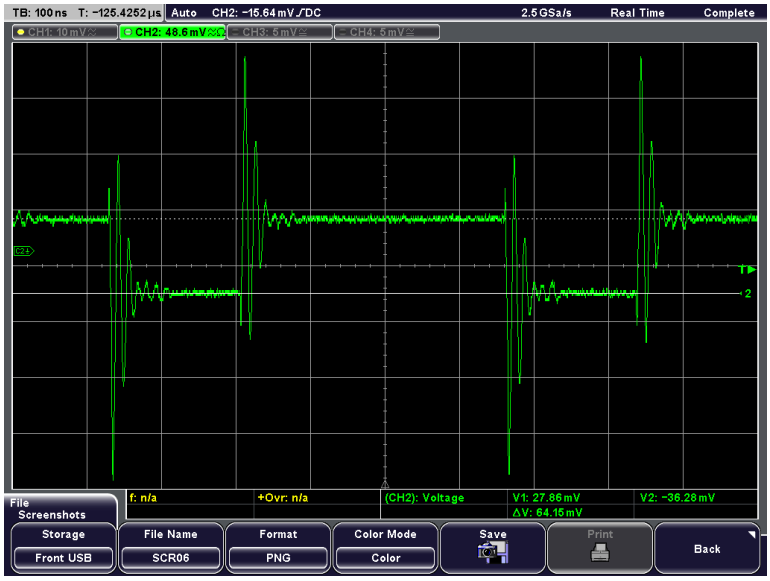

Figure 4.3: Snapshot of DAC output in the time domain

Figure 4.3 shows severe over- and undershoot during the transitions between low and high, while also displaying a settling time with a duration which is a significant part of the clock period. Table 4.1 shows a summary of the voltage levels, and the worst over- and undershoot in figure 4.3.

Table 4.1: Measurements of DAC output in time domain

Measured:	V_{High}	V_{Low}	ΔV	Overshoot	Undershoot
Result:	27.66mV	-36.28mV	64.15mV	141.3mV	160.1mV

Figure 4.4 shows the settling time of a transition from high to low. The settling time is measured to be about 79.9ns ,approximately 45% of the clock period.

The glitching from the pad when transitioning between high and low is severe. During a transition the pad is both under- and overshooting, which means the net glitch impulse area is reduced. This can reduce the error effect of the glitching, as discussed in section 2.1.9. The settling time is almost half of that of the clock period, enhancing the error effect. The error caused by the glitching will also produce ISI, since only transitions between high and low symbols generate glitching. This can clearly be seen in figure 4.3, where 2 consecutive high symbols in the middle of the screen produce no glitching. The effects from these errors are evident from the THD measurements. Thus the digital pad on the Zynq-7000 is not suited for producing high end audio at the 5.6MHz switching rate.

The harmonic distortion from the pad limits the performance of the S-D DAC. The ENOB is limited to about 13 bits, thus the full potential of the S-D DAC cannot be utilized using the digital pad on the Zynq-7000.

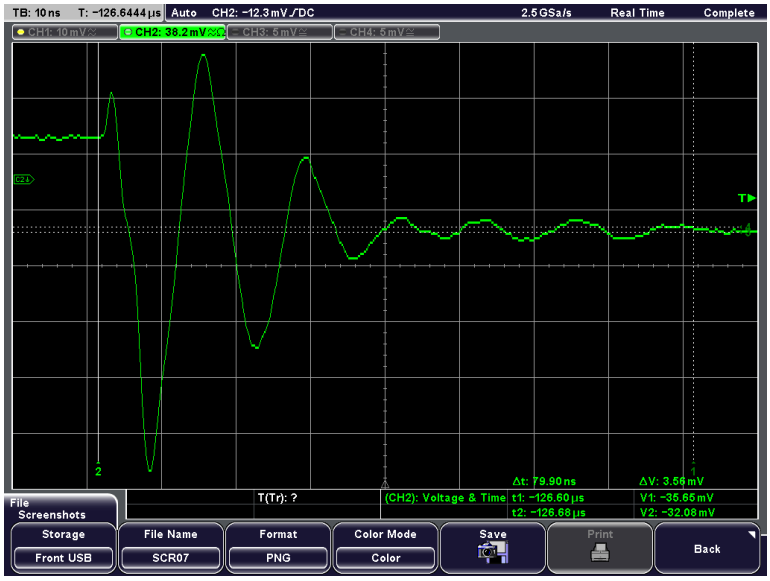

Figure 4.4: Meassurement of DAC settling time

4.1.2 Sigma-Delta DAC performance

The digital pad on the Zynq-7000 limits the maximum performance of the S-D DAC, but it can still be tested for different bit depths, amplitudes and frequencies. All the following measurement are done with an output load of $R_L = 1320\Omega$, and all sine waves have a sample rate of 44.1kHz.

4.1.2.1 Sigma-Delta DAC spectrum

In figure 4.5, the measured output spectrum from 20Hz to 2.8224MHz is shown. This is equivalent to the digital spectrum of the S-D DAC.

The S-D DAC runs a 3kHz sine wave with a bit depth of 16 bits continuously, similar as for the RTL simulation of the S-D DAC in figure 3.9. By comparing the measured spectrum to the spectrum from RTL simulations in figure 3.9, it is clear that they are close to identical. Thus, the digital parts of the S-D DAC implementation on the Zynq-7000 is working as expected.

Figure 4.6 shows the same measured output spectrum zoomed in at the baseband, from 20Hz to 20kHz. The 3kHz fundamental is marked with M1, and the 2nd and 3rd harmonic are marked with respectively D2 and D3. The X- and Y-values of the D2 and D3 markers are with reference to the fundamental M1. The 2nd harmonic is -82.71dB under the fundamental, and is limiting the overall THD performance. This harmonic distortion do not show up in the RTL simulations of the S-D DAC IP, and is likely due to the switching characteristics of the digital pad and ISI. In order to investigate the error produced by ISI, the transition density on the output of the S-D modulator is plotted, alongside the input samples to the S-D modulator in figure 4.7. The results in figure 4.7 are from a

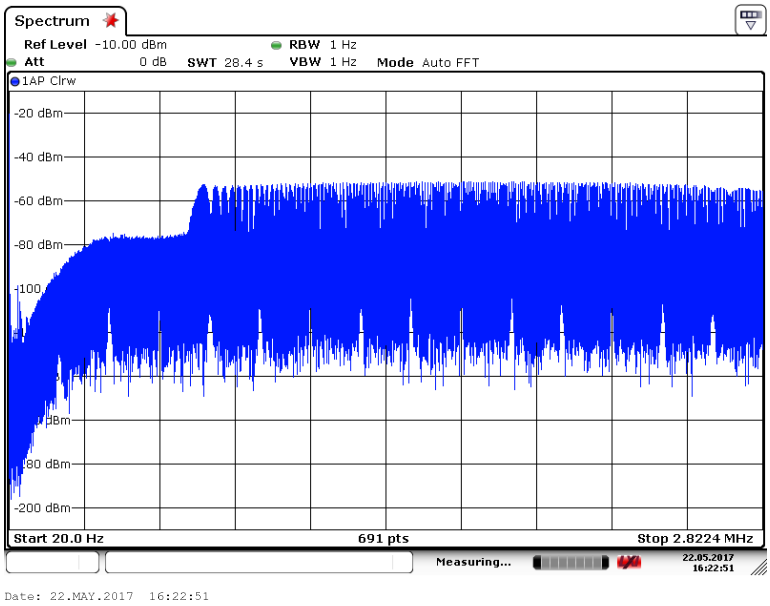

Figure 4.5: Measured spectrum of S-D DAC

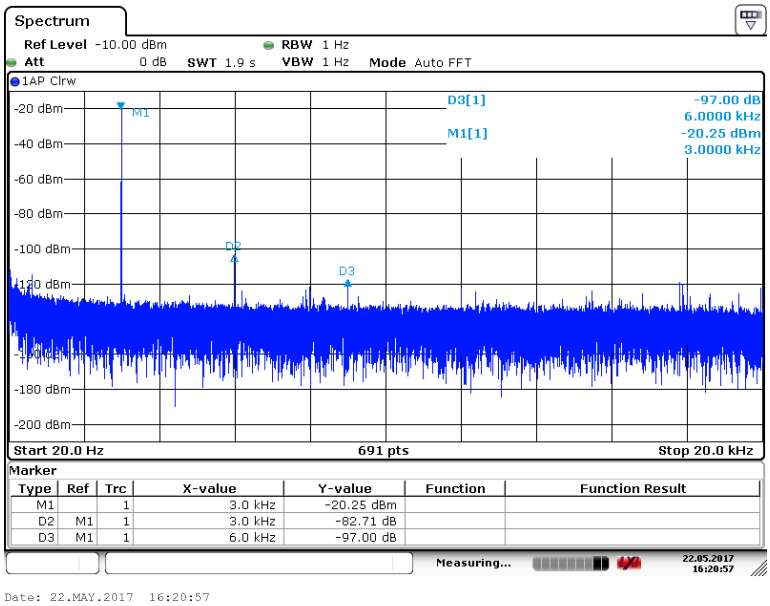

Figure 4.6: Measured baseband spectrum of S-D DAC

RTL simulation of the S-D DAC IP. The input is a 3kHz sine wave with a amplitude of -2.5dBFS. The transition density is the number of transitions on the output of the S-D modulator in the preceding 100 samples.

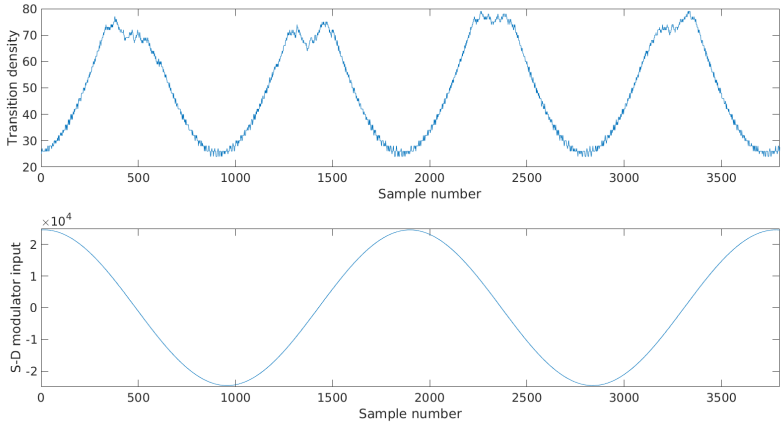

Figure 4.7: Transition density and input of S-D modulator

Figure 4.7 shows that the ISI error should have a strong 2nd harmonic content, and this is what the measured spectrum in figure 4.6 clearly shows. Thus affirming that the harmonic distortion is a product of the digital pad's switching characteristics combined with ISI.

Comparing the basband spectrum of RTL simulation in figure 3.10 to the measured spectrum in figure 4.6, reveals some interesting differences. The noise floor in the simulation is about 140dB under the fundamental, 20dB more than the noise floor in the measured spectrum. Due to the fact that the noise floor of the spectrum analyzer cannot go any lower, limiting the in band noise power N to about -76.5dB. This is equivalent to an ENOB of 12.4 bits. The current measurement setup will not be able to correctly measure the in band quantization noise power for bit depths higher than 11-12 bits. At bit depths higher than 12 bits, the THD is likely to be dominating the total noise power, so the setup can still be used to get a insight into the performance at the higher bit depths. As a result the measurement setup is not changed.

4.1.2.2 Noise performance versus bit depth

To test the performance of the S-D DAC implementation on different bit depths, the S-D DAC is measured while running sine waves of bit depths from 8 to 16 bits. All the sine waves have a frequency of 3kHz with an amplitude of -2.5dBFS. The measurement of the in band THD, N and THD+N are done with the spectrum analyzer, and the results are shown in figure 4.8.

The results show that both the noise N and THD+N are close to a linear reduction from 8 to 12 bits. From 13 bits and beyond it flattens out, which is excepted since the noise floor of the spectrum analyzer is limiting the noise measurements. The THD has an insignificant

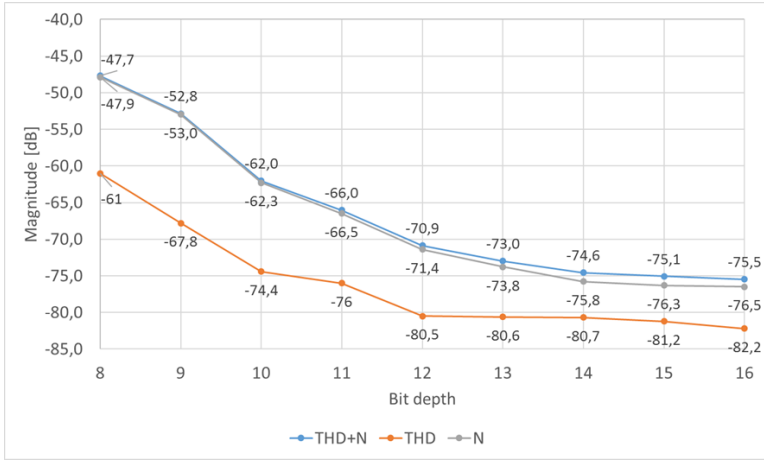

Figure 4.8: THD, N and THD+N versus bitdepth

affect on the THD+N from 8 to 10 bits, and the plot shows that it has a linear reduction in this range. At 11 bits, the reduction of THD flattens out. From 12 to 16 bits the THD stabilizes at a value of about -80.5dB. The equivalent ENOB for the THD and THD+N measurements in figure 4.8 is plotted in figure 4.9, along side the ideal ENOB line.

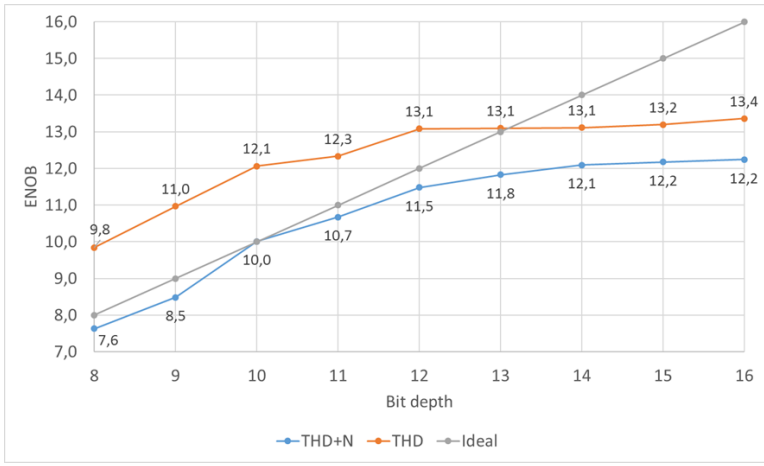

Figure 4.9: ENOB versus bitdepth

The results in figure 4.9 show that the THD+N of the S-D DAC implementation almost follow the ideal line up to a bit depth of 12 bits. At this point the noise floor of the spectrum analyzer is limiting the measurement, and the curve flattens out until reaching a max ENOB of 12.2 bits. The measurements are however, not limited too much by the analyzer, since the ENOB for the THD reaches its maximum value at a bit depth of 12 bits. Thus, the THD performance limits the total noise performance to about -80.5dB or

an ENOB of 13.1 bits, for bit depths higher than 12 bits.

4.1.2.3 Noise performance versus amplitude

In order to test the S-D DAC implementations response to different input amplitudes, a range of amplitudes from the maximum stable input at -2.5dBFS to -20dBFS are tested on the DAC. A 3kHz sine wave with a bit depth of 16 bits, is used for all the measurements. The results from THD measurements are plotted in figure 4.10.

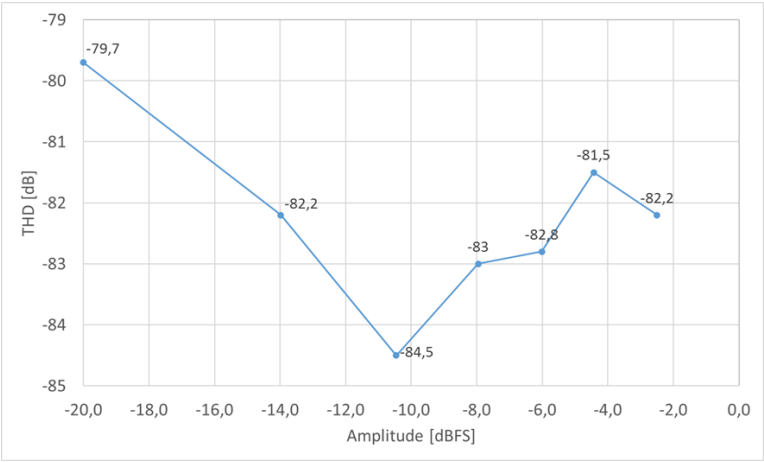

Figure 4.10: THD versus amplitude

The results in figure 4.10 show that the THD is varies for the different amplitudes around the average THD of -82.3dB. In these measurements the the biggest deviation from the average is 2.6dB at an amplitude of -20dBFS.

4.1.2.4 Noise performance versus frequency

In order to test the response of the S-D DAC implementation for different frequencies, the THD performance are tested with a range of frequencies from 100Hz to 7kHz. All the sine waves used in the measurements has an amplitude of -2.5dBFS, and a bit depth of 16 bits. The results are plotted in figure 4.11.

Figure 4.11 shows a severe harmonic distortion at 100kHz, rapidly reduced as the frequency increases until reaching the 1kHz mark. The THD slowly reduce until 3kHz, stabilizing around -83dB. The worst measured THD is -56.3dB at 100Hz, equivalent to an ENOB of 9.1 bits.

Why the S-D DAC implementations has an increasing harmonic distortion at the lower frequencies is hard to say. The RTL simulations of the S-D DAC does not have this harmonic distortion at the lower frequencies, suggesting that the harmonic distortion is not produced by the digital implementation. The distortion when the digital pad is transitioning is the same for all the frequencies, but might be reinforced by ISI or other nonlinear

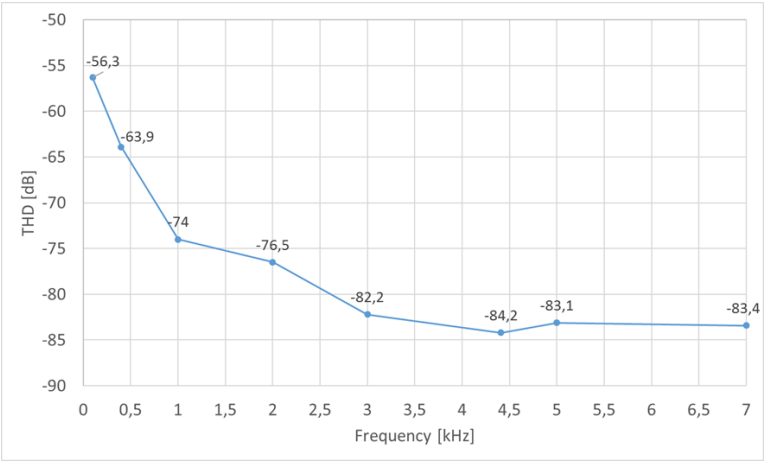

Figure 4.11: THD versus frequency

effects. The distortion from ISI is dependent on the bit pattern from the S-D modulator, and this will differ for the various frequencies. This can be the explanation for the harmonic distortion at the lower frequencies.

4.1.3 Audio testing

Since the digital pad and ISI limit the performance of the S-D DAC implementation and due to time limitations, no scientific psychoacoustic testing is done. The implementation is still tested with audio samples, to test if any clear distortions or noise can be heard. Audio files in the free lossless audio codec (FLAC) format are converted to C arrays using Matlab, and loaded into the ZedBoard's DDR3 memory. The GPIO pin is connected to an auxiliary port (AUX) input on an audio system, with an output load of $R_L = 1320\Omega$ and a 100μF DC-block capacitor in series. The audio samples were successfully played on the audio system from the S-D DAC, and no apparent distortion or noise is detected when listening to the sound. Even though this is not a scientific test, it still contributes to the verification that the S-D DAC implementation is working as expected.

4.2 Area and power estimation

An important aspect to consider when implementing digital IPs in an application-specific integrated circuit (ASIC), is the area and power usage. The total power consumption is determined by the dynamic and static power consumption of the digital logic. The static power dissipation is primarily dependent on the complementary metal-oxide-semiconductor (CMOS) process technology used, and is caused by leakage currents in the CMOS gates. The dynamic power dissipation is dependent on the switching activity of the digital logic, the CMOS process technology and the supply voltage. The switching activity is the most important aspect of the power dissipation to consider at the IP design level. The switching

activity is determined by an activity factor, and the clock frequency of the digital logic.

The DAC IP from section 3.3 consists of the digital blocks of the S-D DAC, and this IP is running at a fixed clock frequency of 5.6MHz. It can input samples with a bit depth of up 16 bits, with a fixed sampling rate of 44.1kHz. The PWM test IP from section 3.6 can be configured for a range of bit depths. Its clock frequency depend on the bit depth and sampling rate of the input samples. The PWM IP is a small IP in comparison to the DAC IP, but must run at a higher clock frequency. The power estimation will give an indication of how the area versus the clock frequencies affect the total power consumption.

In order to compare the DAC IP and PWM IP, the area and power of the two IPs are estimated using the Synopsys SpyGlass Power tool. The synthesis uses TSMC's low power 55-nm libraries with a supply voltage of 1.1V, and a typical operating temperature of 25°Celsius.

4.2.1 Area results

4.2.1.1 Sigma-Delta DAC

The area result from the SpyGlass synthesis is shown in table 4.2, and listed for the different sub IPs. The indent of the IP names in the *module* column indicates the hierarchy of the IPs. The total area of an IP includes the area of all the sub IPs in the hierarchy.

Table 4.2: DAC IP area results

Module	Total Area	Number of Combinational Instances	Total Number of Registers
dac	36132.300 μm^2	8382	2050
casfilt	33029.600 μm^2	7783	1956
casfilt_stage1	16216.800 μm^2	4113	1109
casfilt_stage2	7061.040 μm^2	1700	302
casfilt_stage3	4481.120 μm^2	1103	183
casfilt_stage4	2094.960 μm^2	345	137
casfilt_stage5	1493.520 μm^2	247	100
casfilt_stage6	790.720 μm^2	133	56
casfilt_stage7	891.520 μm^2	142	69
sdMod	3102.680 μm^2	599	94

Figure 4.12 shows a pie chart of how the total area of the *dac* module is utilized by the sub modules. The *casfilt* IP is omitted in the pie chart as it is a wrapper IP for the filter stages, and does not add any additional logic.

The area results show that the first filter stage clearly is the largest module, using almost 45% of the total area. The area usage decrease for each subsequent filter stage until reaching the last two filters, the last filter being a fraction bigger than the previous. This result is as anticipated since the first filter is the largest with 131 taps, while the next two filters have 12 and 6 taps respectively. The CIC filters in stage 4-7 use substantially less area than the first three filters, because they are multiplier-free filters. The advantage of using CIC filters are clear from the pie chart, where the last 4 stages only use a total

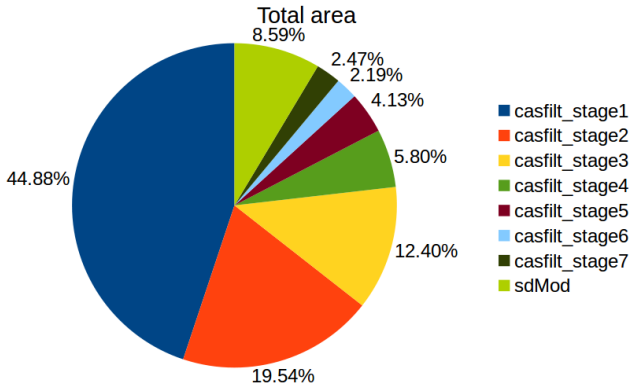

Figure 4.12: Total area usage of DAC IP

of 14.59% of the total area. The S-D modulator module is using 8.59% of the total area, which seems reasonable considering its computations are mostly additions and bit shift operations.

4.2.1.2 PWM DAC

The PWM IP is synthesised in SpyGlass with a bit depth range from 8 to 14 bits. The area results from the synthesis for each bit depth are listed in table 4.3.

Table 4.3: PWM IP area result

Module	Bit depth	Total Area	Number of Combinational Instances	Total Number of Registers
pwm	8	234.080 μm^2	42	18
pwm	9	261.800 μm^2	49	20
pwm	10	288.400 μm^2	53	22
pwm	11	315.000 μm^2	61	24
pwm	12	343.000 μm^2	64	26
pwm	13	370.720 μm^2	70	28
pwm	14	398.160 μm^2	76	30

The total area of the PWM module versus its bit depth is plotted in figure 4.13. The area of the PWM module is on average about 115 time smaller than the DAC module. This is as expected, as the PWM module is basically a binary counter and some simple logic. The increase in area with respect to the bit depth is almost linear. This seems reasonable since the length of the internal registers and the binary counter are increasing linearly with the bit depth.

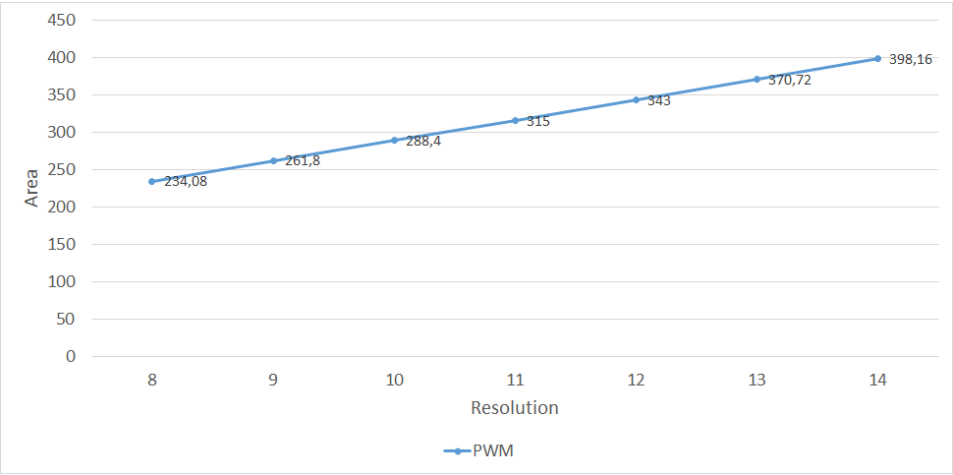

Figure 4.13: Area usage versus bit depth for PWM DAC

4.2.2 Power results

The Synopsys SpyGlass Power tool uses the synthesized RTL code of the two IPs, and estimates power consumption based on a switching activity file. The switching activity file is generated by simulating the RTL code of the two IPs with a typical use case, logging the activity to a FSDB file. The FSDB file is loaded into the SpyGlass tool, which runs a power estimation based on this switching activity.

The use case simulated and logged for the DAC IP and PWM IP, is a sine wave of 4.41kHz. The sine wave has a sample rate of 44.1kHz, and a varying bit depth. The DAC IP is simulated using a bit depth from 8 to 16 bits, and the PWM is simulated using a bit depth from 8 to 14 bits. Using these switching activity files, the power is estimated.

4.2.2.1 Sigma-Delta DAC power consumption

The estimated power consumption of the DAC IP with 16 bit samples, is shown in table 4.4. The *total internal* column represents the internal power consumption in the standard cells, and the *total switching* column represents the power consumption due to switching on the output of the standard cells. The *total leakage* column represents the total static power dissipation of the IP.

Table 4.4: DAC IP power estimation with 16 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	72.658 uW	115.859 nW	61.620 uW	10.922 uW

The power estimations shows that the total leakage is insignificant compared to the dynamic power consumption of the IP. In figure 4.14 the total power of 72.658μW is presented as a pie chart, dived into the power consumption of the submodules.

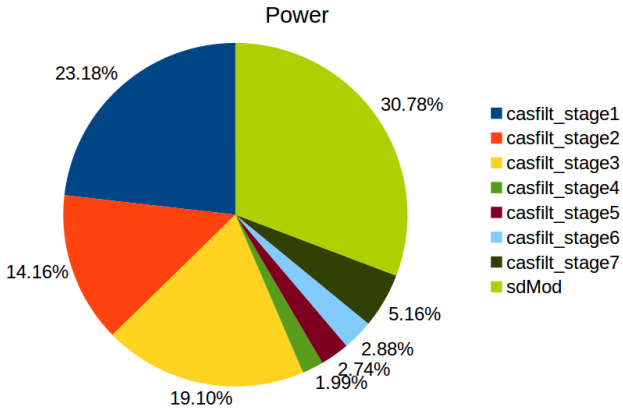

Figure 4.14: S-D DAC power consumption with 16 bit samples

The pie chart shows that the S-D modulator use almost one third of the total power, despite only using 8.59% of the total area. The reason for this inconsistency between the area and power consumption, is the high clock rate and switching activity of the S-D modulator. The modulator is running at a clock rate of 5.6MHz, and conduct a computation every clock period. In comparison, the IF stages have a much lower switching activity due to clock gating and its design implementation. Therefore the IF use two thirds of the power consumption, despite using 91.41% of the total area. Since even a small reduction in area of the S-D modulator could have a big impact on the total power consumption of the DAC, an attempt to optimize the implementations of the S-D modulator could be sensible regarding power efficiency.

The largest contribution to the total power consumption is the IF, using over two thirds of the total power. In figure 4.15 the IF's power consumption is presented as a pie chart, dived into the seven filter stages.

The first filter stage uses over one third of the IF's power, and is the single largest contributor to the total power consumption. This is reasonable considering it is almost half of the total area of the IF, but also has the lowest clock rate. The third filter stage is the second largest contributor to the total power consumption, with 27.6% of the total power. This is large considering it uses only 13.57% of the total area of the IF. The CIC filters in stage 4 to 7 are using only 18.46% of the total power, even though they are running on the highest clock rates. This shows how economical these filters are in terms of both area and power, in comparison to conventional FIR filters. A suggestion for reducing the power consumption of the IF, is to implement the third filter stage as a CIC filter, and moving inverse sinc filter to the second stage. A CIC filter will give a high word length and gain penalty on its output, but could reduce the power consumption considerably.

4.2.2.2 Power consumption comparison

The power consumption of the PWM IP is estimated for bit depths from 8 to 14 bits, with a fixed sample rate of 44.1kHz. This is done in order to compare the two IPs when running

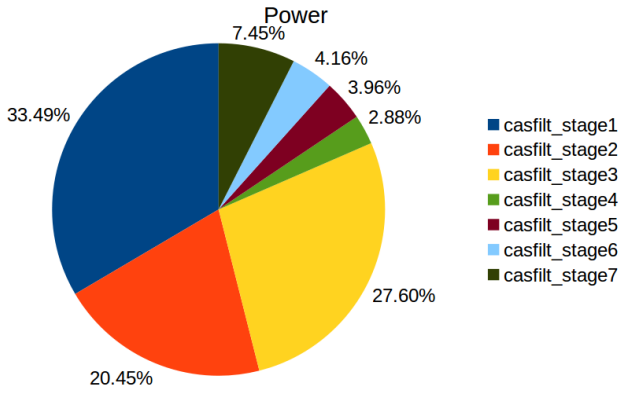

Figure 4.15: IF power usage with 16 bit samples

the same samples. This increases the clock frequency for the PWM IP for every extra bit in the bit depth. The clock frequencies for the different bit depths are summarized in table 4.5.

Table 4.5: Bit depth versus clock rate for PWM IP

Bit depth:	8	9	10	11	12	13	14
Clock rate [MHz]:	11.3	22.6	45.2	90.3	180.1	361.3	722.5

The clock rate of the PWM IP starts at 11.3MHz and doubles for every extra bit. Thus, the clock rates quickly become too high to realize for a practical PWM DAC. In a practical implementation the sample rate would be decreased for the higher bit depths.

The power consumption of the DAC IP is estimated for bit depths from 8 to 16 bits. The implementation of the DAC IP is fixed for all the bit depths, but is still estimated to see if the bit depth has any impact on the power consumption. In figure 4.16 the total power consumption of the DAC IP and PWM IP are plotted versus the bit depth of the input samples.

The power consumption of the DAC IP is reasonably constant over all the bit depths. This is reasonable considering its implementation and clock rate is fixed for all the estimations. The power consumption for the PWM IP starts at 1.754μW, and a little more than doubles for every extra bit in the bit depth. This also seems reasonable since the binary counter in the PWM IP is doubling its switching activity for every extra bit.

The PWM DAC has considerable less power consumption than the S-D DAC for the lower bit depths, and does not surpass the DAC IP before a bit depth of 13 bits. At this point, the clock rate of the PWM IP is 361.3MHz, and is not realizable in a practical implementation. At a bit depth of 8 bits were it is realizable in practice, the PWM IP uses 40 times less power then the S-D DAC.

The results from the power estimations clearly show that the PWM IP uses considerably less power for the lower bit depths compared to the DAC IP. The PWM IP has a

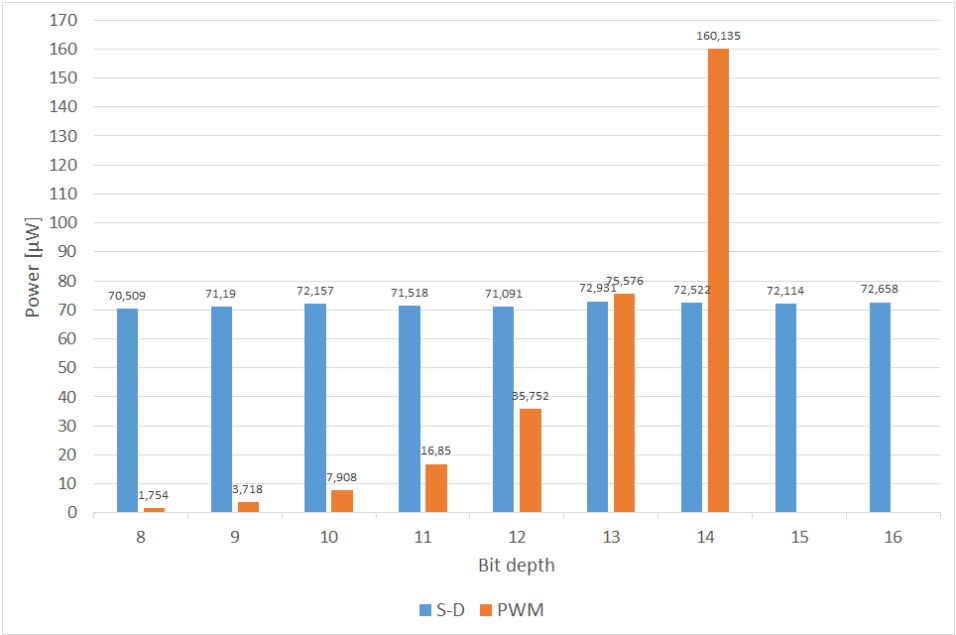

Figure 4.16: Power consumption versus bit depth for DAC and PWM IP

higher power consumption for bit depths higher than 12 bits, but is not realizable at these bit depths with a sampling rate of 44.1kHz. This means that for high end audio it is not possible to use the PWM DAC IP. The detailed results from all the power estimations can be found in appendix B.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The main goal for this thesis is to implement a complete solution for a S-D DAC on a FPGA for an audio application, and compare the solution to a PWM solution.

An IF filter for the S-D DAC was designed and implemented in RTL code. The S-D modulator from [6] was implemented in RTL code, and connected to the IF in the DAC IP. A S-D DAC IP for the FPGA implementation was designed, including the DAC IP, AXI4-bus interfaces, a control register and a stability control unit for the S-D modulator. The ZedBoard development board, which uses a Zynq-7000 SoC, was the target platform for the design. The S-D DAC IP was implemented on the PL of the Zynq-7000, along with a framework on the PL for controlling and streaming samples to the S-D DAC IP. A program was made for the PS on the Zynq-7000, which streamed samples from the DDR3 memory on the ZedBoard to the S-D DAC IP. This concluded the implementation of the S-D DAC.

The output from the S-D DAC was connected to a GPIO pin, and the output was measured using a spectrum analyzer. The results show that the performance of the S-D DAC is limited by the digital pad on Zynq-7000, used as a 1-bit DAC. The THD versus the output load on the digital pad was tested, and the best performing output load was found to be a load of 1320Ω. Even with this output load, the switching characteristics of the digital pad had serve over- and undershoot, with a settling time approximately 45% of a clock period. At 3kHz with a bit depth of 16 bits, the measured THD is -82.2dB, equivalent to an ENOB of 13.4 bits. The THD did not stay constant over the baseband, and increased for the lower frequencies. The worst measured THD is -56.3dB at 100Hz, equivalent to an ENOB of 9.1 bits. The reason for this distortion was determined to be the poor switching characteristics of the digital pad combined with ISI, and/or other nonlinear effects.

The audio performance of the S-D DAC implementation is still much better compared to an ideal single sided PWM scheme, which was simulated and found to be limited to a THD of -19.4dB. The S-D DAC implementation were also connected to an audio system, and tested with audio samples. The audio was successfully played, and no apparent distortion or noise was heard. No scientific psychoacoustic testing was done, but the test

verified that the implementation worked as expected.

The power consumption of the S-D DAC IP was estimated, and compared a single sided PWM scheme when running at a sample rate of 44.1kHz. The S-D DAC IP used around 71μW of power, when running samples from 8 to 16 bits. The PWM DAC IP used 40 times less power then the S-D DAC IP when running 8 bit samples. The power consumption for the PWM IP doubles for every extra bit in the samples, and surpasses the S-D DAC IP at 13 bits. At these high bit depths, the clock frequency of the PWM is so fast that its not realizable. Thus, the PWM DAC IP uses considerably less power than the S-D DAC IP in scenarios where it is realizable.

The S-D DAC is more generic considering the range of bit depths it can run at a 44.1kHz sample rate, but the PWM DAC uses much less power when it can be used. The S-D DAC also uses considerably more area than the PWM DAC IP, but has a much better audio performance. In scenarios where both schemes can be used, the choice will be between good audio quality or low power consumption.

The main goal of this thesis is achieved with a fully working S-D DAC solution on a FPGA, capable of playing high end audio samples. The audio performance, area and power consumption of the S-D DAC solution is compared to a PWM DAC solution, fulfilling the second goal of this thesis.

5.2 Future work

The results from this work clearly show that audio performance of the S-D DAC implementation is limited by the digital pad and ISI. To improve the audio performance, methods for reducing ISI could be explored. The distortion from the digital pad could also be noise shaped with a S-D modulator, by using an ADC as a feedback from the digital pad. Another options is to implement the output from the S-D DAC IP on a differential digital pad, or design a 1-bit DAC with better performance.

The power consumption of the S-D DAC could be reduced by exploring more of the design space of the IF. The power estimations shows that CIC filters are very economical in both area and power, in comparison to the conventional FIR filters. One possible improvement is implementing a CIC filter in the third filter stage.

The power estimation also shows that a small reduction in the digital logic of the S-D modulator could have a significant improvement on the overall power consumption of the S-D DAC. Thus, exploring possibilities for optimization of the S-D modulator, could be beneficial for the total power consumption.

References

[1] E. Analog Devices Inc., *Data Conversion Handbook*. Elsevier Science, 2004.

[2] I. Løkken, "Interpolation filters for oversampled audio DACs." https://www.scribd.com/document/20203269/Interpolation-filters-for-oversampled-audio-DACs, January 2006. [Online; accessed May 23, 2017].

[3] D. G. Manolakis and V. K. Ingle, *Applied digital signal processing: theory and practice*. Cambridge University Press, 2011.

[4] E. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE transactions on acoustics, speech, and signal processing*, vol. 29, no. 2, pp. 155–162, 1981.

[5] Xilinx Inc, All programmable, "Zynq-7000." https://www.xilinx.com/content/dam/xilinx/imgs/block-diagrams/zynq-mp-core-dual.png, 2017. [Online; accessed May 19, 2017].

[6] H. T. Jøsok, "Sigma-Delta DAC for audio," project assignment, NTNU, December 2016.

[7] S. Franco, *Design with Operational Amplifiers and Analog Integrated Circuits*. McGraw-Hill international editions, WCB/McGraw-Hill, 1998.

[8] L. Gaddy and H. Kawai, "Dynamic Performance Testing of Digital Audio D/A Converters," *Application Bulletin AB-104.—Burr-Brown Corporation.—1998,—8 pp*, vol. 18, 1997.

[9] A. Sanyal and N. Sun, "An enhanced ISI shaping technique for multi-bit $\Delta\Sigma$ DACs," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pp. 2341–2344, IEEE, 2014.

[10] I. Løkken, "Pcm-pwm analysis brief." https://www.scribd.com/doc/20488456/PCM-to-PWM-Analysis-Brief. [Online; accessed May 20, 2017].

[11] R. Schreier and G. Temes, *Understanding Delta-Sigma Data Converters*. Wiley, 2004.

[12] F. Taylor, *Digital Filters: Principles and Applications with MATLAB*. IEEE Series on Digital & Mobile Communication, Wiley, 2011.

[13] R. Lyons, "Understanding cascaded integrator-comb filters." `http://www.embedded.com/design/configurable-systems/4006446/Understanding-cascaded-integrator-comb-filters`, March 2005. [Online; accessed May 5, 2017].

[14] Avnet, Phoenix, Arizona, *ZedBoard (Zynq™ Evaluation and Development) Hardware User's Guide*, 2.2 ed., January 2014.

[15] Xilinx, San Jose, California, *AXI Reference Guide*, 13.4 ed., January 2012.

[16] "R&S© FSV Signal and Spectrum Analyzer." `https://www.rohde-schwarz.com/us/product/fsv-productstartpage_63493-10098.html`. [Online; accessed May 23, 2017].

[17] "R&S© RTM2000 Digital Oscilloscope." `https://www.rohde-schwarz.com/us/product/rtm2000-productstartpage_63493-38936.html`. [Online; accessed May 23, 2017].

Appendices

Appendix A

Attachments

Attached to this thesis are the scripts, HDL code and C code used in the thesis. The Vivado and SDK projects of the S-D DAC implementation are also attached. The directory structure in the digital attachment is presented below:

```
Attachments
├── IP_Repo
│   ├── DAC
│   ├── IF
│   ├── PWM
│   ├── SD_DAC
│   └── SDmod
├── Matlab_Script
├── PS_Design
│   ├── Driver
│   └── Main
├── SDK_Project
└── Vivado_Project
```

The directories contain the following files:

- DAC
  - **dac.sv** -RTL code of the DAC IP from section 3.3.
  - **tb_dac.sv** -Test bench for the DAC IP.
- IF
  - **casfilt.v** -RTL code of the top level IF IP.
  - **casfilt_stage1.v** -RTL code of filter stage 1.
  - **casfilt_stage2.v** -RTL code of filter stage 2.
  - **casfilt_stage3.v** -RTL code of filter stage 3.

  - **casfilt_stage4.v** -RTL code of filter stage 4.
  - **casfilt_stage5.v** -RTL code of filter stage 5.
  - **casfilt_stage6.v** -RTL code of filter stage 6.
  - **casfilt_stage7.v** -RTL code of filter stage 7.
  - **casfilt_tb.v** -Test bench for the IF, generated by Matlab's HDL coder.
- PWM
  - **pwm.sv** -RTL code of the PWM test IP from section 3.6.
  - **tb_pwm.sv** -Test bench for the PWM test IP.
- SD_DAC
  - **RegEventControl.sv** -RTL code of the Event control IP from section 3.4.
  - **RegTasks.sv** -RTL code of the Task control IP from section 3.4.
  - **SD_DAC_v2_0.v** -RTL code of the SD_DAC IP from section 3.4.
  - **SD_DAC_v2_0_S00_AXI.v** -RTL code of the AXI4-Lite IP from section 3.4.
  - **StabilityControlUnit.sv** -RTL code of the Stability control unit IP from section 3.4.
- SDmod
  - **compOutput.py** -Python script used for comparing the simulation output from the C code and RTL code of the S-D modulator.
  - **DSmod3OSR128FinalRev.c** -C code model of S-D modulator from the previous work in [6].
  - **SD_mod.sv** -RTL code of S-D modulator IP from section 3.2.
- Matlab_Script
  - **makeIF.m** -Matlab script used when designing the IF in section 3.1.
- Driver
  - **SD_DAC.c** -C code of the S-D DAC driver.
  - **SD_DAC.h** -Header file of the S-D DAC driver.
- Main
  - **DMAtest1.c** -C code of the main program running on the PS.
  - **DMAtest1.h** -Header file of the main program running on the PS.
  - **sinus.h** -Header file containing a C array of a 3kHz sine wave, which is used in the main program.

- SDK_Project
  - **DAC-FPGA.sdk** -A directory containing the SDK project of the complete S-D DAC implementation on the ZedBoard. The project is ready to be flashed on the ZedBoard, and will play a 3kHz sine wave continuously on the *XADC-GIO2* GPIO pin.
- Vivado_Project
  - **DAC-FPGA** -A directory containing the Vivado project of the complete PL design from section 3.5.2.

Appendix B

Detailed Power Estimation Results

The power consumption of the DAC IP and PWM IP are estimated using the Synopsys SpyGlass Power tool. The IPs are synthesized with TSMC's low power 55-nm libraries. The power consumption is estimated on the netlists using a supply voltage of 1.1V, and a typical operating temperature of 25°Celsius. The Synopsys SpyGlass Power tool estimates the power consumption based on a switching activity file. The switching activity file is generated by simulating the RTL code of the two IPs with a typical use case, logging the activity to a FSDB file. The FSDB file is loaded into the SpyGlass tool, which runs a power estimation based on this switching activity. The use case simulated and logged for the DAC IP and PWM IP, is a sine wave of 4.41kHz. The sine wave has a sample rate of 44.1kHz, and a varying bit depth.

B.1 PWM DAC

The results from the power estimations of the PWM IP is shown in table B.1.The power consumption of the PWM IP is estimated with bit depths from 8 to 14 bits.

Table B.1: PWM DAC IP power estimations

Module	Bit depth	Total Power	Total Leakage	Total Internal	Total Switching
pwm	8	1.754 uW	736.971 pW	1.608 uW	145.350 nW
pwm	9	3.718 uW	823.527 pW	3.481 uW	235.407 nW
pwm	10	7.908 uW	906.400 pW	7.491 uW	416.261 nW
pwm	11	16.850 uW	991.009 pW	16.065 uW	784.585 nW
pwm	12	35.752 uW	1.075 nW	34.245 uW	1.506 uW
pwm	13	75.576 uW	1.534 nW	72.647 uW	2.928 uW
pwm	14	160.135 uW	1.622 nW	154.245 uW	5.888 uW

B.2 Sigma-Delta DAC

The power estimation results from the SpyGlass Power tool are shown in tables B.2 to B.10. The power consumption of the DAC IP is estimated when running samples with 8 to 16 bits. In the tables, the results are listed for the different sub IPs. The indent of the IP names in the *module* column indicates the hierarchy of the IPs. The total power consumption of an IP includes the power consumption of all the sub IPs in the hierarchy.

Table B.2: S-D DAC IP power estimation with 8 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	70.509 uW	114.722 nW	59.738 uW	10.657 uW
casfilt	48.004 uW	105.102 nW	41.061 uW	6.838 uW
casfilt_stage1	14.523 uW	46.584 nW	12.065 uW	2.411 uW
casfilt_stage2	10.693 uW	22.770 nW	9.340 uW	1.330 uW
casfilt_stage3	13.612 uW	14.292 nW	11.690 uW	1.907 uW
casfilt_stage4	1.443 uW	8.708 nW	1.259 uW	175.070 nW
casfilt_stage5	1.948 uW	6.061 nW	1.700 uW	241.675 nW
casfilt_stage6	2.066 uW	3.194 nW	1.797 uW	266.645 nW
casfilt_stage7	3.719 uW	3.492 nW	3.210 uW	505.880 nW
sdMod	22.505 uW	9.620 nW	18.677 uW	3.819 uW

Table B.3: S-D DAC IP power estimation with 9 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	71.190 uW	115.763 nW	60.347 uW	10.727 uW
casfilt	48.713 uW	106.142 nW	41.694 uW	6.913 uW
casfilt_stage1	15.318 uW	47.386 nW	12.770 uW	2.500 uW
casfilt_stage2	10.406 uW	22.901 nW	9.083 uW	1.301 uW
casfilt_stage3	13.845 uW	14.308 nW	11.902 uW	1.928 uW
casfilt_stage4	1.420 uW	8.751 nW	1.239 uW	172.506 nW
casfilt_stage5	1.941 uW	6.078 nW	1.695 uW	240.727 nW
casfilt_stage6	2.065 uW	3.221 nW	1.796 uW	266.005 nW
casfilt_stage7	3.717 uW	3.496 nW	3.209 uW	505.105 nW
sdMod	22.476 uW	9.621 nW	18.653 uW	3.814 uW

Table B.4: S-D DAC IP power estimation with 10 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	72.157 uW	115.310 nW	61.193 uW	10.848 uW
casfilt	49.730 uW	105.689 nW	42.582 uW	7.043 uW
casfilt_stage1	15.840 uW	46.957 nW	13.227 uW	2.567 uW
casfilt_stage2	10.472 uW	22.884 nW	9.140 uW	1.310 uW
casfilt_stage3	14.168 uW	14.332 nW	12.186 uW	1.968 uW
casfilt_stage4	1.431 uW	8.739 nW	1.249 uW	173.831 nW
casfilt_stage5	1.968 uW	6.063 nW	1.718 uW	243.989 nW
casfilt_stage6	2.097 uW	3.216 nW	1.823 uW	270.286 nW
casfilt_stage7	3.754 uW	3.498 nW	3.240 uW	510.707 nW
sdMod	22.427 uW	9.621 nW	18.611 uW	3.806 uW

Table B.5: S-D DAC IP power estimation with 11 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	71.518 uW	114.978 nW	60.635 uW	10.768 uW
casfilt	49.069 uW	105.357 nW	42.005 uW	6.959 uW
casfilt_stage1	15.662 uW	46.714 nW	13.069 uW	2.546 uW
casfilt_stage2	10.563 uW	22.814 nW	9.223 uW	1.318 uW
casfilt_stage3	13.642 uW	14.313 nW	11.725 uW	1.902 uW
casfilt_stage4	1.431 uW	8.746 nW	1.248 uW	174.091 nW
casfilt_stage5	1.965 uW	6.049 nW	1.716 uW	243.509 nW
casfilt_stage6	2.077 uW	3.235 nW	1.806 uW	267.803 nW
casfilt_stage7	3.730 uW	3.486 nW	3.219 uW	507.124 nW
sdMod	22.449 uW	9.620 nW	18.630 uW	3.809 uW

Table B.6: S-D DAC IP power estimation with 12 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	71.091 uW	115.528 nW	60.242 uW	10.733 uW
casfilt	48.730 uW	105.907 nW	41.687 uW	6.937 uW
casfilt_stage1	15.315 uW	47.208 nW	12.750 uW	2.518 uW
casfilt_stage2	10.502 uW	22.879 nW	9.167 uW	1.312 uW
casfilt_stage3	13.711 uW	14.312 nW	11.782 uW	1.915 uW
casfilt_stage4	1.437 uW	8.770 nW	1.254 uW	174.824 nW
casfilt_stage5	1.967 uW	6.045 nW	1.717 uW	243.794 nW
casfilt_stage6	2.072 uW	3.207 nW	1.801 uW	267.219 nW
casfilt_stage7	3.726 uW	3.486 nW	3.216 uW	506.886 nW
sdMod	22.361 uW	9.621 nW	18.555 uW	3.796 uW

Table B.7: S-D DAC IP power estimation with 13 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	72.931 uW	115.770 nW	61.879 uW	10.936 uW
casfilt	50.533 uW	106.149 nW	43.291 uW	7.136 uW
casfilt_stage1	16.683 uW	47.485 nW	13.966 uW	2.670 uW
casfilt_stage2	10.759 uW	22.877 nW	9.397 uW	1.339 uW
casfilt_stage3	13.871 uW	14.330 nW	11.925 uW	1.931 uW
casfilt_stage4	1.436 uW	8.726 nW	1.253 uW	174.469 nW
casfilt_stage5	1.974 uW	6.029 nW	1.723 uW	244.773 nW
casfilt_stage6	2.079 uW	3.202 nW	1.808 uW	268.316 nW
casfilt_stage7	3.732 uW	3.499 nW	3.221 uW	507.689 nW
sdMod	22.398 uW	9.621 nW	18.588 uW	3.801 uW

Table B.8: S-D DAC IP power estimation with 14 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	72.522 uW	116.060 nW	61.504 uW	10.901 uW
casfilt	50.113 uW	106.439 nW	42.913 uW	7.093 uW
casfilt_stage1	16.217 uW	47.756 nW	13.546 uW	2.623 uW
casfilt_stage2	10.699 uW	22.862 nW	9.341 uW	1.335 uW
casfilt_stage3	13.946 uW	14.320 nW	11.996 uW	1.936 uW
casfilt_stage4	1.444 uW	8.737 nW	1.260 uW	175.589 nW
casfilt_stage5	1.986 uW	6.051 nW	1.733 uW	246.598 nW
casfilt_stage6	2.084 uW	3.219 nW	1.812 uW	268.709 nW
casfilt_stage7	3.738 uW	3.494 nW	3.226 uW	508.560 nW
sdMod	22.409 uW	9.621 nW	18.591 uW	3.808 uW

Table B.9: S-D DAC IP power estimation with 15 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	72.114 uW	115.855 nW	61.136 uW	10.862 uW
casfilt	49.754 uW	106.234 nW	42.584 uW	7.064 uW
casfilt_stage1	15.816 uW	47.524 nW	13.187 uW	2.581 uW
casfilt_stage2	10.585 uW	22.893 nW	9.240 uW	1.323 uW
casfilt_stage3	14.070 uW	14.325 nW	12.100 uW	1.956 uW
casfilt_stage4	1.445 uW	8.731 nW	1.260 uW	175.878 nW
casfilt_stage5	1.991 uW	6.049 nW	1.738 uW	246.810 nW
casfilt_stage6	2.096 uW	3.221 nW	1.822 uW	270.298 nW
casfilt_stage7	3.752 uW	3.491 nW	3.238 uW	510.392 nW
sdMod	22.360 uW	9.621 nW	18.552 uW	3.798 uW

Table B.10: S-D DAC IP power estimation with 16 bit samples

Module	Total Power	Total Leakage	Total Internal	Total Switching
dac	72.658 uW	115.859 nW	61.620 uW	10.922 uW
casfilt	50.297 uW	106.238 nW	43.068 uW	7.122 uW
casfilt_stage1	16.843 uW	47.459 nW	14.095 uW	2.700 uW
casfilt_stage2	10.287 uW	22.915 nW	8.976 uW	1.288 uW
casfilt_stage3	13.881 uW	14.326 nW	11.936 uW	1.930 uW
casfilt_stage4	1.449 uW	8.762 nW	1.263 uW	176.380 nW
casfilt_stage5	1.994 uW	6.048 nW	1.741 uW	247.413 nW
casfilt_stage6	2.094 uW	3.226 nW	1.821 uW	270.178 nW
casfilt_stage7	3.749 uW	3.500 nW	3.236 uW	509.979 nW
sdMod	22.361 uW	9.621 nW	18.552 uW	3.799 uW