NTNU – Trondheim
Norwegian University of
Science and Technology

FPGA Filter Design and Measurements with Emphasis on a Filter with Steep Transition Bands

Andre Firing

Master of Science in Electronics
Submission date: May 2013
Supervisor: Bjørn B. Larsen, IET
Co-supervisor: Stig Rooth, Kongsberg Norspace AS
Tor Audun Ramstad, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Design of Digital Systems

Master's Thesis

FPGA Filter Design and Measurements with Emphasis on a Filter with Steep Transition Bands

Andre Firing

Supervisors
Bjørn B. Larsen, IET
Tor A. Ramstad, IET
Stig Rooth, Kongsberg Norspace

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Electronics and Telecommunications

Abstract

The objective of this thesis was to design and implement a digital bandpass filter with emphasis on steep transition bands. The goal of the design was to create a FPGA based digital alternative to an existing analog filter used in the Galileo Satellite Search and Rescue Transponder. This filter has a passband of 100 kHz centered at 70.98 MHz. The transition bands of the filter are approximately 50 kHz on both sides of the passband, ending in a stopband attenuation of at least 80 dB. The phase of the filter response is linear.

The proposed filter, called *Masterfilter*, was designed by applying multistage and multirate filtering techniques, including undersampling, signal mixing, decimation and interpolation. The filter was designed with a sampling frequency of 30 MHz, and the input signal was undersampled down to 10.98 MHz. The output of the Masterfilter was centered at 3.48 MHz, with a bandwidth of 100 kHz. Matlab was used to simulate the behavior and determine intermediate filter characteristics, while VHDL was used to implement the filter in hardware. To make the design fit the performance and spacial limitations of a space qualified FPGA, digital design optimization techniques such as pipelining and resource sharing were implemented. The filter was designed to be as general as possible, meaning that no vendor specific components were used in the design.

The digital filter was tested using several approaches, including impulse responses and tone testing of both internal filters and the complete filter system. The transfer function of the Masterfilter was verified using Modelsim simulations. In addition to these simulations, a version of the filter was implemented in an Altera Cyclone II FPGA, where the functionality of the filter, and the actual frequency response was determined. These results provided a transfer function for the complete filter system.

It was concluded that the filter did fulfill the requirements given, and that it would be a sufficient digital alternative to the existing analog filter in the Search and Rescue Transponder. An infinite attenuation, where the output transmitted a constant zero was measured at 39 kHz below the passband, at at 30 kHz above the passband. Some issues with timing were encountered when using certain optimization techniques. Therefore, it would be recommended that a device specific clocking logic should be used in an FPGA implementation.

Sammendrag (Norwegian)

Bakgrunnen for denne masteroppgaven var design og implementering av et digitalt båndpassfilter med hovedvekt på minimale transisjonsbånd. Målet med oppgaven var å utvikle et FPGA-basert digitalt filter tilsvarende et analogt båndpassfilter, plassert i dagens Galileo Satellite Search and Rescue Transponder. Det digitale filteret skulle designes og implementeres for bruk på en romfartskvalifisert FPGA. Det eksisterende analoge filteret har en båndbredde på 100 kHz sentrert ved 70.98 MHz, med transisjonsbånd på 50 kHz. Dempingen i stoppbåndet skal være minst 80 dB, og faseresponsen skal være lineær.

Det foreslåtte digitale filteret, her kalt *Masterfilter*, ble utformet ved å benytte flertrinns filterteknikker, som blant annet undersampling, miksing, desimering og interpolering. Sampling frekvensen til filteret ble satt til 30 MHz, og inngangssignalet ble undersamplet til 10.98 MHz. Utgangssignalet av filteret ble sentrert om 3.48 MHz, med en båndbredde på 100 kHz. Matlab ble benyttet for å lage prototyper av interne filtere i designfasen, og det endelige filteret ble implementert i VHDL. For å oppfylle kravene til ytelse og plassbruk i en romfartskvalifisert FPGA ble digitale optimaliseringsteknikker benyttet. Dette omfattet blant annet pipelining og resource sharing. For å gjøre filteret så generisk som mulig, ble ingen leverandør-spesifike komponenter benyttet.

Filteret ble grundig testet ved blant annet test av pulsrespons og tonetesting av både interne komponenter og det samlede filtersystemet. Filterets overføringsfunksjon ble bestemt ved hjelp av simuleringer i Modelsim. I tillegg til disse simuleringene ble det også implementert en versjon av filteret i en Altera Cyclone II FPGA. Her ble funksjonalitet og frekvensrespons testet og kartlagt.

På grunnlag av de testresultatene som ble funnet i denne oppgaven ble det konkludert med at det foreslåtte digitale filteret ville være et tilstrekkelig alternativ til det eksisterende analoge filteret i dagens Search and Rescue Transponder. En uendelig demping ble målt for frekvenser 39 kHz under passbåndet, og 30 kHz over passbåndet. Det ble funnet noen utfordringer med timing i den fysiske implementeringen av filteret, og det anbefales at FPGA-spesifik klokkelogikk blir benyttet i en eventuell implementasjon.

Preface

This paper was written as a master's thesis in design of digital systems at NTNU. The focus of my master's degree is FPGA design, and digital signal processing was a relatively new field for me before this thesis. This thesis was performed over the course of 20 weeks, without any related preceeding specialization project.

The assignement was given by Kongsberg Norspace, under the supervision of Stig Rooth. The initial problem description was quite open, and further limitations were introduced throughout the semester. Also, it was decided that the focus of the thesis should mainly be FPGA based design of the filter, as this was most relevant considering previous experience and interests.

I would like to thank my thesis supervisors, Associate Professor Bjørn B. Larsen and Professor Tor A. Ramstad for helpful guidance, and Kongsberg Norspace for an interesting and challenging assignment. I would also like to thank Stig Rooth, Maxime Adadja and Espen Flo Eriksen for guidance and insight during this semester.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC	Analog to Digital Converter
C-Cells	Combinatorial Cells (Actel)
CORDIC	Coordinate Rotation Digital Computer
DAC	Digital to Analog Converter
dBFS	Decibels relative to full scale
DSP	Digital Signal Processing
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
$\mathbf{F_s}$	Sampling Frequency
GPIO	General Purpose Input/Output
HDL	Hardware Description Language
IIR	Infinite Impulse Response
IP	Intellectual Property
LO	Local Oscillator
LTI	Linear Time-Invariant
LUT	Look-Up Table
MSB	Most Significant Bit
NCO	Numerically Controlled Oscillator
PLL	Phase-Locked Loop
QML	Qualified Manufactering Line
RAM	Random Access Memory
R-Cells	Register Cells (Actel)

RHA	Radiation Hardened Assurance
RMS	Root-Mean-Square
RTL	Register Transfer Level
SART	Search and Rescue Transponder
SAW	Surface Acoustic Wave
SEL	Single Event Latch-Up
SEU	Single Event Upset
SNR	Signal-to-Noise Ratio
SQNR	Signal-to-Quantization-Noise Ratio
TID	Total Ionizing Dose
TMR	Triple Module Redundancy
VHDL	Very-High-Speed Integrated Circuit Hardware Description Language

Chapter 1

Introduction

1.1 Motivation

One of the most important components in any signal processing system is frequency sensitive filters. Filters are applied to exclude frequency components from a signal, and they are generally classified according to their frequency responses. A field where digital signal processing, and hence filtering is widely applied is in communication systems. Communication systems can be found in cellphones, broadcasting and even in space.

The concept of space communication systems brings forth new challenges in the filter design, such as reliable performance in extreme conditions. This is where digital signal processing has an advantage over traditional analog systems. As long as the system clock is held at a constant frequency, the behavior of the system will also remain stable, in contrast to analog systems which will drift significantly during changing temperatures. In satellite systems, analog components will have to be heated to a constant temperature to achieve reliable performance, which means bringing a heavy and energy costly heat source on board the satellite.

One application of space communication is the use of emergency transponders, which receives distress signals from one location on earth, filters the signal and transmits it back to a rescue central. This functionality can be found in some modern satellites, among them the new Galileo Satellites. One of the filters used in the Galileo Satellite Search and Rescue Transponder is a very narrow bandpass filter centered at 70.98 MHz, with a bandwidth of 100 kHz. The transition bands of the filter are approximately 50 kHz at both sides of the passband, with an attenuation exceeding 80 dB in the stopbands. To design a digital *finite impulse response* (FIR) version of this filter directly with the *Parks-McClellan* filter design algorithm [1] would require an extreme amount of filter stages, which would not be possible to implement in any of the current space qualified *Field Programmable Gate Arrays* (FPGA). To design a realizable digital filter with these specifications, advanced filter techniques and digital design methodology will have to be applied. However, the gain of reducing the area which will be climate controlled on the satellite will free up the weight- and power budget for other potential applications, which in turn makes the design efforts well worth while.

1.2 Problem Description

The object of this thesis is to design an FPGA based digital filter suitable to replace the analog filter which is used in the current Galileo Satellite Search and Rescue Transponder. This filter has a very high quality factor, with a bandwidth of 100 kHz, transition bands of 50 kHz and a center frequency of 70.98 MHz. Since the filter is to be placed in a sensitive communication system, the filter will need to be strictly stable and have a linear phase. The filter is a second stage filter in the receiver chain, preceded by a 1.8 MHz bandpass *surface acoustic wave* (SAW) filter and a mixer. The input of this filter will therefore be a bandpass signal, with a bandwidth of 1.8 MHz centered at 70.98 MHz, and a sufficiently attenuated stopband. From preceding stages in the transponder chain, one can also expect to find undesired signal components at 741 MHz and 670 MHz. These components should be removed with this digital filter. Furthermore, the following specification should be fulfilled by the digital filter:

- $F_{S,FPGA} = 30\,\text{MHz} \pm 5\,\text{MHz}$
- The FPGA should use the same clock frequency for receiving and transmitting data externally.
- The design should fit into a space qualified FPGA, preferably an Actel RTAX2000S.
- The bandwidth of the passband should be approximately 100 kHz.
- The attenuation in the stopbands should be at least 80 dB, 50 kHz apart from the passband.
- The filter response should have a linear phase.

The filter should be designed and simulated with Matlab, and implemented in hardware using VHDL. The implemented filter should be verified by simulations of the VHDL code, and testing on an FPGA. The simulated performance of the filter should be characterized by its transfer function.

1.3 Report Structure

The report will be structured as follows:

- Chapters 2 and 3 will handle the necessary signal processing and digital design theory needed for the design and testing.
- Chapter 4 will introduce the existing analog filter used in the Search and Rescue Transponder unit.
- In chapter 5, the tools used in the thesis are described.
- Chapter 6 describes the designed filter, and the reasons behind the design choices.
- In chapters 7 and 8 the physical performance estimations and simulation results are described.
- In chapters 9, 10 and 11 the results are discussed and conclusions are stated. Suggestions of further improvements are also described.

Chapter 2

Signal Processing

2.1 Digital Sampling

One of the fundamental prerequisites of a digital signal-processing system is that the input of the system should be a digital signal. In contrast to analog signals, the digital signal consists of several quantized steps which represents the magnitude and frequency information of the signal.

2.1.1 Signal Resolution

The granularity of the digital signal, or how similar the digital signal is to its original analog signal is dependent upon, among other parameters, how many bits which are used to represent the amplitude steps. These steps are called *quantization steps*, represented by [1]

$$\Delta = \frac{x_{max} - x_{min}}{2^b - 1} \tag{2.1.1}$$

where x is the input signal and b represents the number of bits which will represent the signal. Δ defines the *quantizer step size*, which are determined by the *Analog to Digital Converter* (ADC).

Equation 2.1.1 suggest a significant increase in signal resolution as a result of increasing the bit width. This can be illustrated using a signal with an analog amplitude of 2 V. If a 2 bit digital signal is to represent this signal, there would only be four steps to represent this 2 V range. However, if a 16 bit digital signal is used to model this analog signal, 2^{16} steps would be available. If the quantization is *linear* or *uniform*, meaning that each quantization step is of equal size, the granularity of the signal would be

$$\frac{2\,\mathrm{V}}{2^{16} - 1} = 3.05 \cdot 10^{-5}\,\mathrm{V} = 30.5\,\mu\mathrm{V} \tag{2.1.2}$$

2.1.2 Quantization and Noise

Since the number of quantization steps used to represent the signal cannot be infinite, a *quantization error* results from the analog to digital conversion, which is represented by

[1]

$$-\frac{\Delta}{2} \leq e_q(n) \leq \frac{\Delta}{2} \tag{2.1.3}$$

A graphical illustration of the quantization error resulting from an ADC can be seen in figure 2.1. Equation 2.1.3 only applies if the analog signal is located within the domain of the ADC. If the signal exceeds the ADC range, the quantization error will exceed the limits from equation 2.1.3. The signal is then clipped, which results in *overload noise* [1].

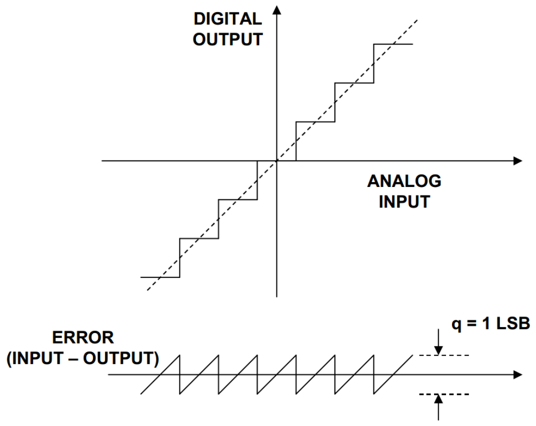


Figure 2.1: ADC Quantization and Noise [2]

The noise resulting from the quantization error is often analyzed together with the signal, as *signal-to-quantization-noise power, SQNR* [1]. The SQNR can be expressed as

$$SQNR = 10log_{10}\frac{P_x}{P_n} \tag{2.1.4}$$

where P_x is the signal power, $E[x^2(n)]$, and P_n is the power of the quantization noise [1]

$$P_n = \sigma_e^2 = \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} e^2 p(e)de = \frac{1}{\Delta}\int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} e^2 de = \frac{\Delta^2}{12} \tag{2.1.5}$$

or, expressed as the root-mean-square (rms) quantization noise [2]

$$P_{n,rms} = \frac{\Delta}{\sqrt{12}} \tag{2.1.6}$$

Assuming that the quantization noise is uncorrelated to the input signal and that $P_{n,rms}$ remains approximately as described in equation 2.1.6, a theoretical output *signal-to-noise ratio* (SNR) can be calculated for a given input at the ADC. For a rail-to-rail sinewave expressed as

$$v(t) = \frac{\Delta \cdot 2^N}{2} sin(2\pi f t) \tag{2.1.7}$$

where N is the number of bits used to quantize the signal, the RMS value would be [2]

$$v_{rms} = \frac{\Delta \cdot 2^N}{2\sqrt{2}} \tag{2.1.8}$$

The SNR for an ideal ADC would then be expressed as [2]

$$SNR = 20 \log_{10} \frac{v_{rms}}{P_{n,rms}} \tag{2.1.9}$$

$$SNR = 20 \log_{10} \frac{\Delta \cdot 2^N / 2\sqrt{2}}{\Delta / \sqrt{12}} \tag{2.1.10}$$

which results in the commonly used formula for calculating the SNR over the *dc* to $\frac{f_s}{2}$ bandwidth for quantized signals,

$$SNR = 6.02\,\mathrm{N} + 1.76\,\mathrm{dB} \tag{2.1.11}$$

Examining equation 2.1.11, it is apparent that the SNR will improve by approximately 6 dB for each bit added to quantize the signal. The constant 1.76 of equation 2.1.11 originates from the rail-to-rail sinewave at the input, and it will vary for other input signals.

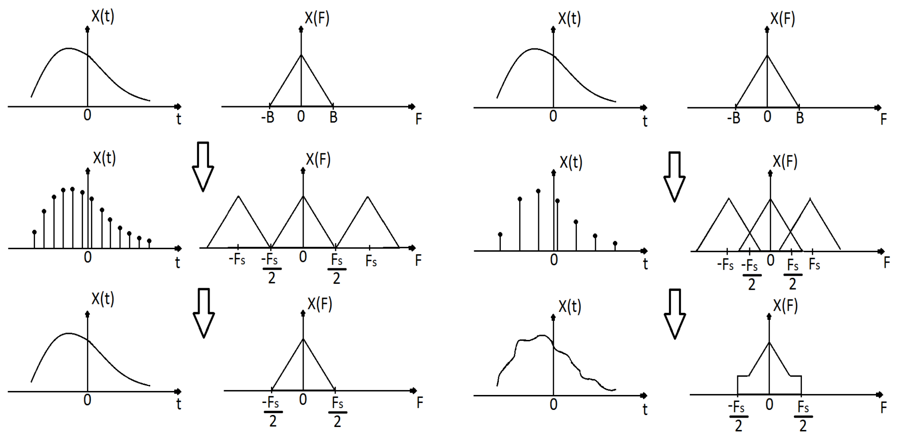
2.1.3 The Sampling Theorem

In addition to the voltage amplitude steps being sampled, it is also essential to know how often to sample these signals. The *Sampling Theorem* states that *"A bandlimited continuous-time signal, with the highest frequency (bandwidth) B hertz, can be uniquely recovered from its samples provided that the sampling rate $F_s \geq 2B$ samples per second"* [1]. For lowpass signals, this implies that the sampling frequency should be twice the frequency of the highest frequency component of the signal. The sampling rate at which $F_s = 2B$ is called the *Nyquist rate* [1].

When an analog signal is sampled, the frequency components are repeated every kF_s frequency, where k is an integer [1]. If the original signal is sampled with a lower frequency than $2B$, the resulting signal will contain interfering frequency components. This phenomena is called *aliasing*, which means that the frequency components are overlapping, and the resulting reconstructed signal will not be equal to the original signal. Aliasing is irreparable and should be avoided [3]. Figure 2.2 describes the process of sampling and reconstructing the original signal, with and without aliasing. Figure 2.2a shows a signal which is sampled with a sampling frequency that satisfies the sampling theorem, and figure 2.2b shows a signal sampled at a lower sampling rate than $2B$.

2.1.4 Sampling of Bandpass Signals

If one is to sample a signal with bandwidth $2B$ and center frequency F_c, where $B << F_c$, the traditional interpretation of the sampling theorem would suggest that the sampling frequency should be at least $2(F_c + B)$. This way the signal is treated as a lowpass signal, which assures that there will be no aliasing at all [4]. However, what the sampling theorem really states is that the sampling frequency needs to be twice the signal bandwidth, making the frequency at which the signal is centered irrelevant for the ability to preserve the signal



(a) $F_S > 2B$ (b) $F_S < 2B$

Figure 2.2: Impact of Aliasing [1]

information after sampling. This fact can be exploited to reduce the sampling frequency of bandpass signals significantly, as these signals can be *downsampled* with a sampling frequency of $2B$ or higher. However, the reduced sampling frequency brings forth new issues with mirroring, aliasing and noise.

The theory of bandpass sampling states that when sampling a signal at a rate of f_s, all frequency components in the analog domain will be mirrored down to the baseband, which is located at *dc* to $\frac{f_s}{2}$ Hz [5]. To better understand the effects of mirroring, the term *band position* is introduced. This term refers to the fractional number of signal bandwidths which the lowest frequency of a bandpass signal is located from the baseband. A special case of band position is called *Integer Band Positioning*[5]. This approach first defines an integer m to be a multiple of the bandwidth, $F_H = mB$ [1], where F_H is the highest frequency component of the signal. This integer m is the band position, and the original signal will be aliased and reflected for each band position, 0 to $m-1$. This is illustrated for a bandpass signal with several possible sampling frequencies in figure 2.3. These bands are also known as *Nyquist zones*. Notice that if the band of the original signal is placed within an even numbered Nyquist zone, the sampled signal at the baseband will be the inverse of the original signal [1]. The theoretical lowest sampling frequency is defined as $F_s = 2B$, but it should be noted that this only applies if there are no signal components at $F_c \pm B/2$ [5].

Another limitation when using the theoretical minimum sampling rate is that any engineering imperfection, such as clock instability will cause aliasing due to the low margin of error in the $\frac{F_S}{B}$ ratio [5]. These considerations leads to the fact that one does not have to use $2B$ as a goal in it self for the optimal sampling frequency.

To counter the sensitivity to engineering imperfections, a sampling scheme called *Arbitrary Band Positioning* is introduced. This line of thought states that the band positioning does not have to be strictly set by the signal bandwidth, leaving room for zero-magnitude frequency components at both sides of the signal. When selecting a sampling frequency which yields non-aliasing regions on both sides of the $\frac{F_H}{B}$ rate, this extra bandwidth is

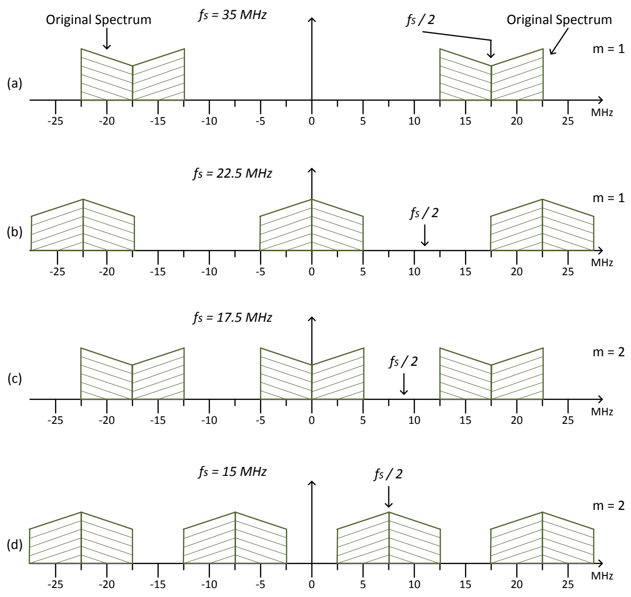


Figure 2.3: Illustration of Bandpass Sampling [4]

called the *guard band* [5]. This region safeguards against aliasing due to engineering imperfections, which is important as virtually all reference clocks are somewhat unstable. Arbitrary band positioning leads to the following two limitations of the sampling frequency F_s [1]

$$2F_H \leq kF_s \tag{2.1.12}$$

$$(k-1)F_s \leq 2F_L \tag{2.1.13}$$

These equations can be solved for k to show that $k_{max} \leq \frac{F_H}{B}$ [1]. k_{max} is the maximum number of Nyquist zones that can fit in the range of dc to F_H. This leaves the minimum sampling rate to avoid aliasing effects as

$$F_{s,min} = \frac{2F_H}{k_{max}} \tag{2.1.14}$$

Equation 2.1.14 implies that the sampling frequency can be selected from a range of frequencies, limited by [1]

$$\frac{2F_H}{k} \leq F_s \leq \frac{2F_L}{k-1} \tag{2.1.15}$$

and

$$1 \leq k \leq \lfloor \frac{F_H}{B} \rfloor \tag{2.1.16}$$

These limitations can be seen graphically in figure 2.4. It should be noted that even though guard bands are introduced and the limitations in figure 2.4 are upheld, clock instability when undersampling is critical.

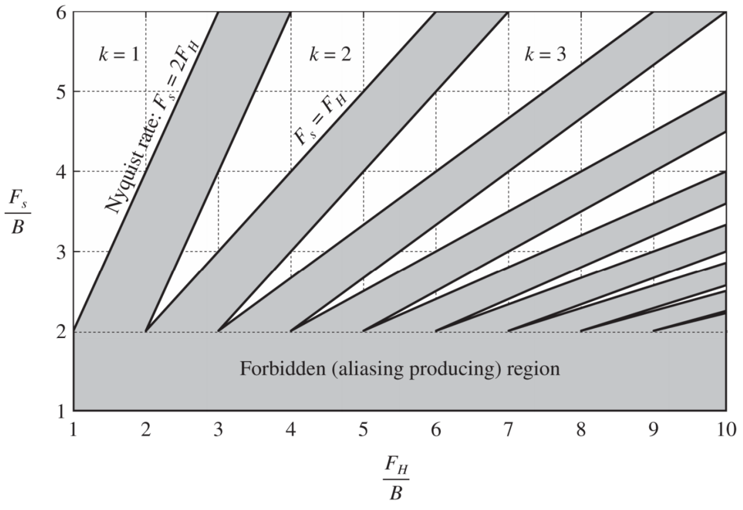


Figure 2.4: Allowed and Forbidden Sampling Frequencies [1]

2.1.5 Noise in Bandpass Sampled Signals

One of the fundamental requirements of bandpass sampling is that the sampled signal is, in fact, a bandpass signal and that the rest of the band is sufficiently attenuated. The reason for this is that all the wideband noise will also be combined into all of the Nyquist zones [5]. This will result in a degraded SNR in the baseband. The sampled SNR can be calculated as [5]

$$SNR \approx \frac{S}{N_p + (n-1)N_o} \tag{2.1.17}$$

where S is the spectral power density, N_p is the in-band noise power density, N_o is the out-of-band noise power density and n is the integer band position. For systems with several Nyquist zones and where the noise power density is equally distributed over the whole spectrum, the SNR degradation can be roughly expressed as [5]

$$D_{SNR} = 10\log(n)\,\mathrm{dB} \tag{2.1.18}$$

This degradation of the SNR can consequently be reduced by increasing the sampling frequency, reducing the number of Nyquist zones between the original signal and the baseband.

2.2 Filtering

One of the most fundamental components of any signal processing system is called a filter. A filter is a circuit which alters the attributes of a signal in the time- or frequency domain [3]. There are many types of filters, but in this thesis the focus will be set on digital, linear time-invariant (LTI) filters. LTI filters are generally divided into two types: *Finite Impulse Response* (FIR) filters and *Infinite Impulse Response* (IIR) filters [1]. Although IIR filters are generally known to have lower sidelobes in the stopband than equally sized FIR filters, the thesis' problem description states that the filter should always be stable, and the phase of the filter should always be linear. This is always the case for FIR filters, while IIR filter have the possibility of being unstable as a result of their feedback-based structure [1].

2.2.1 FIR Filters

An LTI FIR filter can be expressed mathematically as a convolution sum given by [3]

$$y[n] = x[n] * f[n] = \sum_{k=0}^{L-1} f[k]x[n-k] \tag{2.2.1}$$

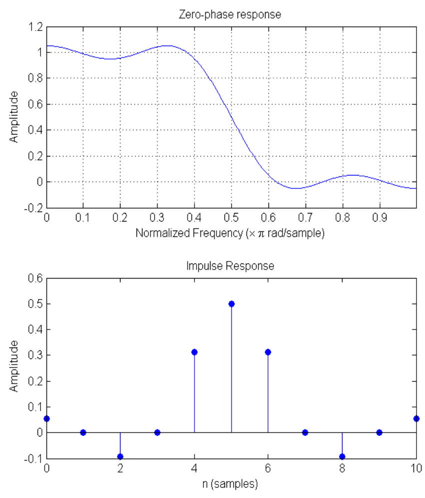
or, expressed in the z-domain

$$Y(z) = F(z)X(z) \tag{2.2.2}$$

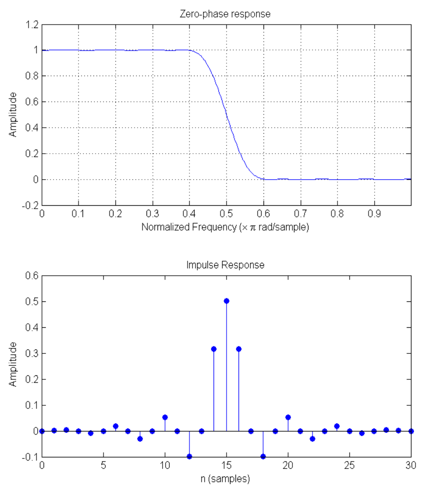
where

$$F(z) = \sum_{k=0}^{L-1} f[k]z^{-k} \tag{2.2.3}$$

L is the number of *stages* of the filter, and $f[k]$ is a set of constants which are called the filter coefficients, or *tap weights*. As the number of filter stages grows, the complexity of the filter also grows. On the other hand, when the number of stages grows, the attenuation and steepness of the filter also improves. This effect can be seen for a lowpass filter in figure 2.5.



(a) Filter With 10 Tap Weights



(b) Filter With 30 Tap Weights

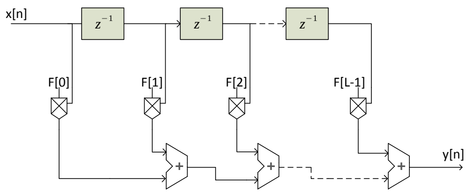
Figure 2.5: Impact of Tap Weights in an FIR Filter

For extremely steep FIR filters, the number of coefficients can grow to the magnitude of 10^4 and higher, which imposes issues with area when realizing the filter. FIR filters tend to use more tap wights than IIR filters, which makes them more expensive to implement. On the other hand, FIR filters have the advantage of being [6]

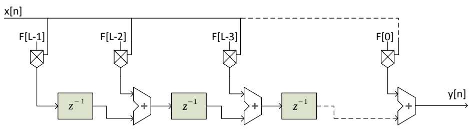
- Robust
- Inherently stable
- Easy to design for linear phase
- Common in digital applications

The FIR filter representation from equation 2.2.3 can be expressed as a circuit where $x[n]$ is the input and $y[n]$ is the output, as seen in figure 2.6a. Alternatively, one can alter the *direct form* FIR filter in figure 2.6a into the *transposed* structure in figure 2.6b [3]. This transposed FIR filter has an advantage of a significantly shorter critical path, which results in an increase in performance and throughput [3].

Another way of reducing the size of the FIR filter is to exploit the symmetry of the tap weights. As seen in figure 2.5, the tap weights are identical at both sides of the center tap weight (often described as the 0^{th} sample [3]). This means that the tap weights are equal in pairs, and only half of the multiplications needs to be performed in order to produce the filter response. Coefficients can also be antisymmetric, and with both odd and even number of filter stages [3]. An FIR filter which exploits the symmetric properties of the tap weights can be seen in figure 2.7.



(a) Direct Form FIR filter



(b) Transposed FIR filter

Figure 2.6: FIR Filter Structure [3]

An additional benefit of symmetric and antisymmetric filter coefficients is that they have a guaranteed linear-phase response. Linear-phase response means that the *group delay* is constant for every input frequency [3], which is quite important in for example communication systems, as signals might otherwise be distorted. The group delay of an N-tap FIR filter can be expressed as [4]

$$G = \frac{D}{2 \cdot f_s} \text{ s} \tag{2.2.4}$$

where D is the number of time-delay elements in the delay line of the filter, and f_s is the sampling frequency. Ideally, the output of a linear-phase filter would simply be a time delayed and amplitude-scaled version of the input [1].

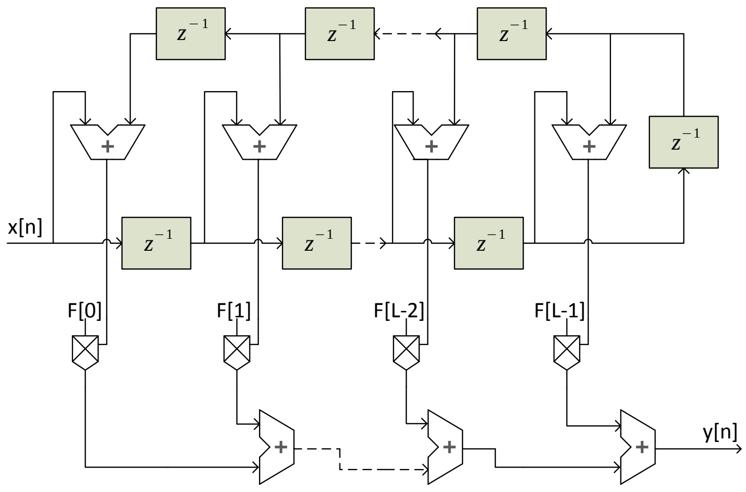


Figure 2.7: Symmetric FIR Filter Structure [3]

2.3 Signal Mixing

Another important operation in signal processing of bandpass signals is the process of shifting the signal in the frequency domain. Frequency shifting is important in several settings. In a receiver it is very common to shift a bandpass signal down to a lower frequency, which allows for less demanding processing of the information contained in the signal. In a digital filter, this down-conversion, followed by a reduction of the sampling frequency is common to reduce the filter size. This is because the filter size is dependent upon the steepness of the filter as a function of the sampling frequency. This will be explained in section 2.4.

Frequency conversion can be achieved in both the analog and the digital domain with a component called a *mixer*. This three-port device receives a signal and multiplies it with a nonlinear or time-varying element and, ideally, outputs a signal consisting of the sum and difference of the two signals [7]. The multiplicand signal, which often originates in a *local oscillator* (LO), is in many real applications a sine wave with frequency f_{LO}. The mixing process is illustrated in figure 2.8.

In most cases, only one of the two frequency components, $f_{RF} + f_{LO}$ or $f_{RF} - f_{LO}$ are wanted, and the other component will have to be filtered out. In reality, more frequency components than just $f_{RF} \pm f_{LO}$ might be generated as unwanted harmonic signals [7].

2.4 Decimation

As mentioned in section 2.2.1, the number of tap weights needed to realize a digital FIR filter heavily depends upon how wide the transition band is, with respect to the sampling frequency. A direct form symmetric FIR filter would for instance need significantly more tap weights to obtain a transition band of 5 kHz at a 100 MHz sampling frequency, than if

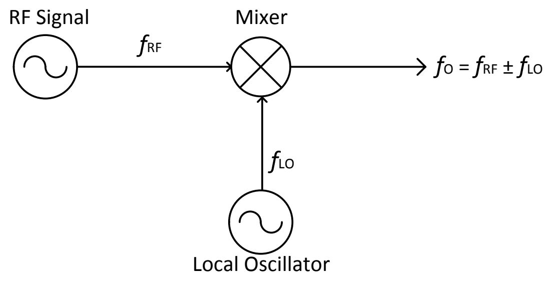


Figure 2.8: Signal Mixer

the sampling frequency was set to 100 kHz. In fact, the size of the filter at the 100 MHz sampling frequency would probably not be feasible in any realistic system. However, in the case of bandpass signals, where the bandwidth of the signal carrying the actual information is much lower than the signal center frequency, there are techniques for reducing the number of tap weights.

Decimation, or *downsampling* is a multirate signal processing technique for reducing the sampling frequency of a signal by a factor D. The general formula for sample rate conversion can be expressed as [1]

$$y(mT_y) = \sum_{n=-\infty}^{\infty} x(nT_x) g(mT_y - nT_x) \tag{2.4.1}$$

where $g(t) = sinc(\pi t/T_x)$, and T_x is the original sampling period. This equation can be specialized for decimation of a factor D as [1]

$$y(mT_y) = y(mDT_x) = \sum_{k=-\infty}^{\infty} x(kT_x) g((mD-k)T_x) \tag{2.4.2}$$

A simple illustration of decimation can be seen in figure 2.9, where a lowpass signal is decimated by a factor $D = 4$. Here one can observe that even though only the D^{th} sample is kept, the curvature of the signal and hence the frequency is maintained after downsampling. This is due to the fact that sampling theorem is still satisfied, as previously explained in section 2.1.3.

However, successful signal decimation is more than just arbitrarily reducing the sampling rate of the signal. If one assumes a signal with nonzero frequency components at all frequencies $|F| \leq F_s/2$, a sample rate reduction of D would cause folded frequency components below $F_s/2D$ [1]. This will cause aliasing when reconstructing the signal. To avoid this distortion, the bandwidth of the original signal will first have to be reduced to the resulting bandwidth after decimation. In other words, if the original signal bandwidth is $F_s/2$, a filter will be needed to reduce this bandwidth to $F_s/2D$ by suppressing all other frequency components [1]. If the relevant bandpass signal is placed in the lower band, where $|F| \leq F_s/2D$, a lowpass filter should be used, with the ideal frequency response [1]

$$H_D(\omega) = \begin{cases} 1, & |\omega| \leq \pi/D \\ 0, & \text{otherwise} \end{cases} \tag{2.4.3}$$

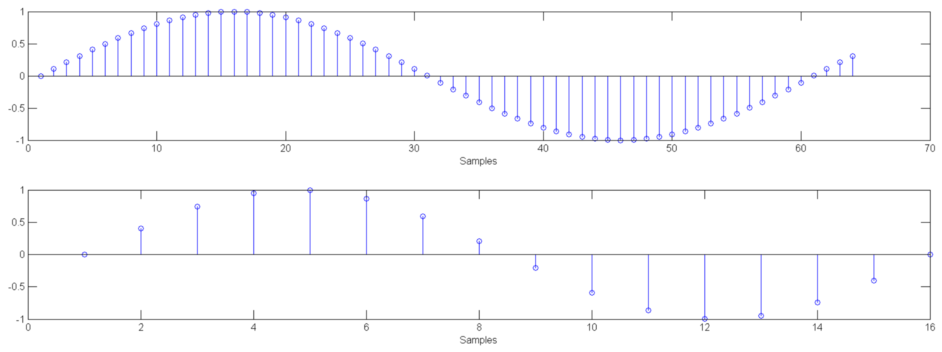


Figure 2.9: Decimation by $D = 4$

Another case of decimation is when the relevant bandpass signal is placed in the $|F| > F_s/2D$ region. As explained in section 2.1.4, when sampling at a frequency F_s, all frequencies $F \geq F_s/2$ will be folded down into the baseband. The folding attributes can be exploited in decimation to both scale down the sampling frequency and to reduce the frequency at which the signal is centered in one operation. This can be illustrated by decimating a signal with a factor of $D = 2$. Equation 2.4.2 suggests that frequencies of $|F| \geq F_s/4$ would be folded down into the band $|F| \leq F_s/4$. To avoid having the folded-down frequency components distorted by the frequencies which already occupies the band $|F| \leq F_s/4$, a highpass filter with the following ideal frequency response should be applied before the downsampling process

$$H_D(\omega) = \begin{cases} 0, & |\omega| \leq \pi/2 \\ 1, & \text{otherwise} \end{cases} \tag{2.4.4}$$

This filter, along with the downsampling will ideally leave only the $|F| \geq F_{s,original}/4$ frequency components in the baseband. However, as these signal components will be folded about $F_{s,original}/4$, the signal components will now be inverted. This means that the frequency component at the original $F_{s,original}/2 - 1$ Hz would now be placed at 1 Hz, and the component at the original $F_{s,original}/4 + 1$ Hz would be placed at the new $F_{s,new}/2 - 1$ Hz. These folded frequency placements behave similarly as the undersampled signals discussed in section 2.1.4. The process of highpass decimation is illustrated in figure 2.10.

In the digital domain, the downsampling process of decimation can be achieved efficiently by only using every D^{th} sample of the signal. The digital filter needed, in both cases of where the relevant signal is placed, should be an FIR filter to maintain the linear phase response. There are no absolute requirements of the attenuation of the filter used in the decimation process, but the filter should be designed for the specific application it is meant for, and how much aliasing this application can withstand [8].

2.5 Interpolation

The goal of signal interpolation is to increase the sampling frequency of the signal, while maintaining the signal information. Although interpolation resembles signal decimation,

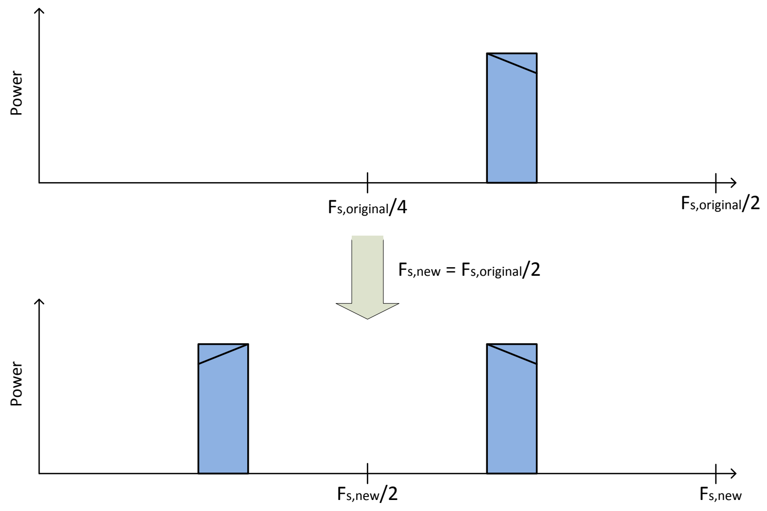


Figure 2.10: Decimation with a Highpass Filter

there are some vital differences. The first step of the interpolation process is to increase the sampling frequency, which is called *upscaling* or *upsampling* [9]. This can be done by placing the original samples at every N^{th} position, and inserting zeros in between each sample [9]. This process is illustrated in figure 2.11. In the frequency domain, the signal will be mirrored at every Nyquist interval. An illustration of this phenomena when upsampling by a factor of $N = 6$ can be seen in figure 2.12.

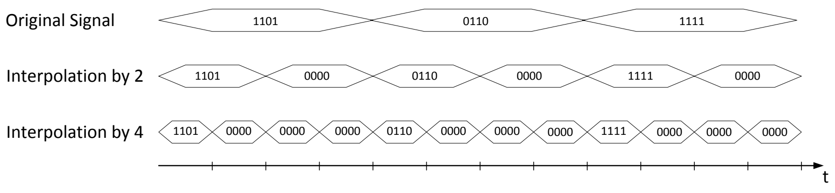


Figure 2.11: Illustration of Interpolation

The next step of the interpolation process is to filter the upscaled signal. Depending on the upscaling factor, there will be several mirrored signals within the bandwidth which needs to be removed. As with the decimation process, one can choose if the desired signal lies in the upper or lower part of the new bandwidth with the type of filter used in the filtering stage.

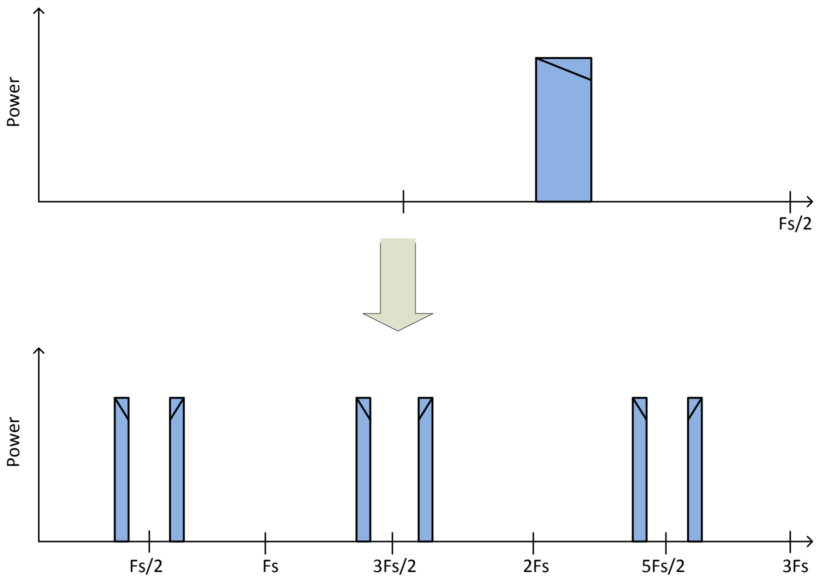


Figure 2.12: Frequency Placement After Upsampling by $N = 6$

Chapter 3

Digital Design

When working with digital signal processing, one of the first decisions to make is whether to use fixed-point representation, or floating-point representation of numbers. In a general sense, floating-point representation has a much higher accuracy than fixed-point representation [3]. This, however, comes at the cost of both performance and circuit complexity.

3.1 Fixed-Point Representation

Any integer can be represented as a fixed-point number through the formula for unsigned integer conversion [3]

$$X = \sum_{n=0}^{N-1} x_n 2^n \tag{3.1.1}$$

where N is the number of bits and x_n is a boolean that is either present or not for the n^{th} symbol. This formula will not apply to negative numbers, as there is no way to determine the sign of this binary number.

There are two common ways of representing signed numbers as fixed-point numbers, with *Signed-Magnitude* and with *Two's Complement*. Signed-magnitude representation uses equation 3.1.1 to convert the absolute value of the integer, and adds another *most significant bit* (MSB) which represents the sign of the number. The benefit of this representation is a simplified prevention of overflows [3]. On the other hand, this representation makes it more expensive to perform additions, since the process will have to be split up depending on which number is largest [3].

When using two's complement to represent signed numbers, the conversion from integer is done with the following equation

$$X = \begin{cases} \sum_{n=0}^{N-2} x_n 2^n & X \geq 0 \\ -2^{N-1} + 1 + \sum_{n=0}^{N-2} x_n 2^n & X < 0 \end{cases} \tag{3.1.2}$$

This representation has the advantage of being able to add several signed numbers within the N-bit range without having to handle overflow, as long as the final output does not result in overflow [3]. It should be mentioned that there are other, more task-optimized ways of representing digital numbers, although two's complement is the most common in digital signal processing (DSP) applications [3].

3.1.1 Overflow

An issue when using fixed-point representation for hardware filter design is the concept of *dynamic range overflow*, which determines how large numbers the filter will need to handle as a worst case. The worst case dynamic range growth can be expressed as [3]

$$G \leq \log_2 \sum_{k=0}^{L-1} |f[k]| \tag{3.1.3}$$

where $f[k]$ is the k^{th} coefficient and L is the filter order. The total number of bits needed to represent the filter is then the sum of the input bit width and this growth factor. If this condition is upheld, no overflow will occur, regardless of which number representation is used. However, equation 3.1.3 is quite pessimistic, as there is only one single input combination which achieves this maximum growth in output size.

The maximum allowed input X_{max} for a given filter internal bit width in the can also be derived from the dynamic range growth, as seen in equation 3.1.4.

$$X_{max} = \pm \frac{2^{N-1}}{\sum_{k=0}^{L-1} |f[k]|} \tag{3.1.4}$$

To avoid the possibility of overflow, X_{max} should be less than the maximum possible input, determined by the bit width.

In larger filter systems, where several filters are cascaded together to form a steep filter, there will have to be some scaling of the bit width. The reason for this is that the bit width would grow to an unfeasible size if not handled. For example, if a filter system with five stages, an input precision of eight bit and coefficients of eight bit would to be connected together without scaling, the output of the filter would sum up to

$$W_{input} + 5 \cdot W_{coefficient} = 48\,\text{bit} \tag{3.1.5}$$

To counter this problem, a scaling unit could be placed at each filter output. One way of achieving this in a fixed point system is to divide the signal by a constant factor, here called the *scalingfactor*. This will not cause distortion, but it will reduce the signal amplitude, which in turn allows for reducing the bit width of the signal.

3.2 Digital Signal Processing in an FPGA

The general signal processing techniques and challenges relevant for this thesis were presented in chapter 2. However, as this thesis is focused upon *digital* signal processing and the techniques presented in the previous chapter are directed at general signal processing, some additional challenges will surface when working with an FPGA.

3.2.1 Digital Mixing

As explained in section 2.3, the mixing process consists of multiplying the original signal with a time-varying signal, often a sine wave. In a digital system, this sine wave will have to be sampled in the same manner as the input signal, and stored in registers. This suggests that the same issues as the ones described in section 2.1 also applies to this signal.

Of course, this multiplicand signal could be sampled with a much better resolution than the input signal at compile time to reduce the quantization errors, but to store this signal in the FPGA would require a vast memory. One way of reducing this error is to use CORDIC-based algorithms [3]. The *Coordinate Rotation Digital Computer* (CORDIC) algorithm is a very common algorithm used in applications such as pocket calculators and DSP-systems. Its main function is to calculate efficient ways of minimizing the quantization error when representing a time-varying function [3]. The CORDIC-architecture is well suited for implementation in an FPGA, but in the case of signal mixing with a predefined signal, the multiplicand signal could simply be calculated in the design process with the help of for example CORDIC-based algorithms in Matlab.

Another, more hands-on method of reducing the size and error of the multiplicand signal is to carefully choose the mixing factor and sampling frequency. If a sine wave is used to mix the input signal, one can exploit the fact that a sine wave is periodical. This would suggest that if for example the LO frequency is set to 15 MHz and the sampling frequency is set to 30 MHz, the LO would only need two samples per period, 1 and -1 to perfectly represent the signal. Consequently, the logic needed to represent this mixer would be very efficient in hardware, as a multiplication with 1 could be replaced by wires, and a multiplication by -1 for two's complement signed numbers consists of an inversion and an addition with 1.

3.2.2 FPGA Clocking

One of the fundamental differences between computer simulations and testing in an actual FPGA is the intricate timing in the physical system. Many digital systems employ a clock signal to coordinate the movement of data through the system [10]. This is called a synchronous design, and it is strictly necessary in for example digital FIR filters since all the time delay elements consists of clocked D-Flip Flops.

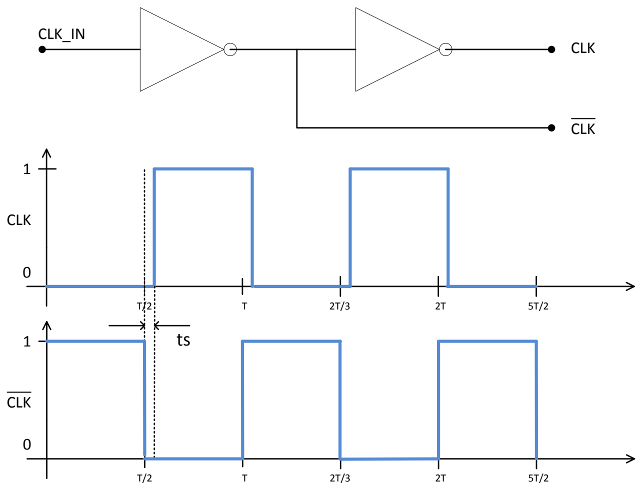


Figure 3.1: Clock Skew [10]

In a multistage FIR filter, stable timing is critical. An important issue with clocking in large systems is called *Clock Skew*. Clock Skew is where a clock signal is out of phase with the system reference [10]. This is a product of either long clocking paths, or from the clock signal propagating through logic with delay. This could for example be an inverter, which is common to use when synchronizing dataflow in cascaded logic. An example of clock skew in a circuit which generates an inverted clock signal can be seen in figure 3.1.

Clock skew can be reduced in several ways. Skew added as a result of long transmission paths within the FPGA can to some extent be reduced in the routing process. Timing-critical components can be grouped together, close to the clock source, and components which are communicating with each other can be placed close. Some modern synthesizer-tools have this option [3]. Also, many FPGAs have dedicated clock routing with low latency, which can be used to reduce clock skew. For example, Actels Antifuse FPGAs, including the RTAX2000S, lets the designer assign clock sources to a dedicated clock network which supports high fanout and minimal skew [11].

Clock skew due to delay in logic or clock generation can also be reduced by adding a *phase-locked loop* (PLL). The PLL is a component which detects differences in the signal phase between the clock source and the feedback from a given point in the delay line, and compensates for this difference at the source [10]. Many modern FPGAs have PLLs built in, and their outputs can be properly routed to dedicated clock hardware.

3.2.3 FPGA and Space Qualification

One of the challenges facing FPGA-based designs for space applications is the limitations of the FPGA-technology. First off, the main difference between common FPGAs and space qualified FPGAs is the ability withstand significant radiation. The radiation affecting satellites is much stronger compared to the radiation experienced on earth. This means that random upsets, such as bit-inversion[1] in memory is prone to happen in digital electronics [12]. There are two common ways of countering this phenomena. The first method is to introduce error correction into the circuit, which checks each stored bit with majority voting at a given interval. This method costs both area, performance and energy. The other method of countering radiation is to build physically larger and more resilient FPGAs, which in turn means physically larger gates. This results in longer delays and lower performance, as well as reduced exploitable logical area.

Space-application FPGAs are often classified as either *radiation-tolerant* or *radiation-hard* [13]. Since space is a relatively new domain for FPGAs, the qualified devices with flight-heritage are sparse. Historically, Actel have been the sole supplier of radiation-hard FPGAs due to the high production cost and small market, but as the FPGA technology has developed and grown more popular, also major manufacturers such as Xilinx have started to develop suitable FPGAs. To be classified as a space-quality FPGA, the FPGA manufacturers needs to be classified with a qualified manufacturing line (QML) Class V with *radiation hardness assurance* (RHA) [13]. This implies, among other specifications, that the single event upset (SEU) limit should be above $37\,MeVcm^2/mg$, and that the total ionizing dose (TID) should be tested for 300 krad for radiation-hard FPGAs and 100 krad for radiation-tolerant FPGAs. Actel's radiation-hard FPGAs are also equipped with *Triple Module Redundancy* (TMR), which means that each register cell in the FPGA has two

[1]Bit-inversion is when a single bit is altered to the opposite value, hence a logic 1 becomes a logic 0, and a logic 0 becomes a logic 1.

other, equal master-slave clones [14]. These three cells acts as a single register cell to the user, but they can correct SEU-errors by using majority voting between themselves. If an error is detected in one of the cells, the majority value will be restored to all of the three cells.

Another important factor when choosing an FPGA for a space application is whether or not the FPGA has any flight heritage. Due to the fact that space hardware has a limited possibility of maintenance, the unit needs to work perfectly, all the time. Since failure is not an option, the safest bet for investors is usually the FPGA which already has proven it self stable in the field. A short list of radiation hardened, space qualified FPGAs which might be suitable for a large digital filter has been summarized in table 3.1 [14], [15]. In this table, the total ionizing dose is measured in *krad*, the single-event latch-up (SEL) immunity is given as $MeVcm^2/mg4$ and the SEU is given as probability of upset/device/day in geostationary orbit.

FPGA	Size[1]	RAM [kB]	DSP Blocks	TID	SEL	SEU	TMR
RTAX2000S	250000	288	No	300	117	10^{-10}	Yes
RTAX4000S	500000	540	No	300	117	10^{-10}	Yes
Virtex4QV	200448	6048	96	300	100	1.5^{-6}	No

Table 3.1: Space Qualified FPGAs

Note that the Xilinx Virtex4QV FPGA has less area than the Actel FPGAs. However, the Virtex FPGA has built in DSP-blocks, which can compensate for the lacking area. It should also be pointed out that the architecture of the Virtex FPGA is fundamentally different from the Actel FPGAs, as the Virtex FPGA uses a Look-up Table (LUT) architecture, rather than a *sea of modules* structure[14], [15].

3.2.4 FPGA Architecture

How the FPGA architecture is designed is highly dependent upon the manufacturer and what purpose the FPGA has. As explained in the previous section, space qualified FPGAs tend to be physically large to reduce the impact of radiation. Aside from this, there are several ways of organizing the synthesized logic inside the FPGA. In the Altera Axcelerator FPGA series, the modules are divided into *combinatiorial cells* (C-cells) and *register cells* (R-cells). These two building blocks can be used as a measurement of how much area the design will require of the FPGA. A *Register Transfer Level* (RTL) illustration of the different cells can be seen in figure 3.2.

3.3 Optimizing for Performance and Area

As mentioned in section 3.2.3, space qualified FPGAs are slow compared to the commercial FPGAs. In order to fulfill requirements in both area and performance, advanced synthesis and programming techniques must be utilized. In a multirate FIR filter several clock domains are used to act as the different sampling frequencies. This means that the

[1]The sizes of the FPGAs are given as equivalent ASIC gates to the LUTs or C-/R-Cells

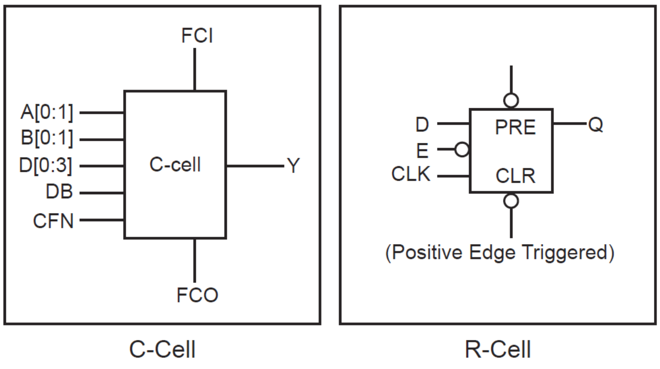


Figure 3.2: C-Cells and R-Cells in an RTAX2000S [14]

optimization focus will differ in each individual clock domain, since the slowest sampling frequency filters will not need to be compliant for the highest frequency domain. However, the lowest frequency-domain filter might be significantly larger than the highest frequency-domain filter. Consequently, the components with the highest clock frequency requirements should be optimized for performance, while the components with the lowest clock frequencies should be optimized for area, as long as they still satisfy the performance requirements.

3.3.1 Pipelining

A common performance optimization in sequential digital logic is based on trading area for performance. One way of achieving this is by dividing cascaded components into smaller sections separated with flip-flops, which allows for a faster system clock [10]. This technique is called *pipelining*. The frequency limit of a given cascaded datapath can be expressed as [10]

$$f < \frac{1}{t_{ff} + t_d + t_{su} + t_s} \tag{3.3.1}$$

where t_{ff} is the delay of the flip-flop, t_d is the delay through the combinatorial logic, t_{su} is the setup time and t_s is possible clock skew time. When using equation 3.3.1 for designs which contains slow and complex combinatorial logic, the factor t_d will be significantly dominant. In order to increase the throughput in such systems, pipelining can be used to divide the critical paths. To calculate the new frequency limit, equation 3.3.1 will now have to be used for the new, smaller datapaths separately. In the ideal case where a pipeline stage is added at $t = t_d/2$, and the remaining factors in equation 3.3.1 remains approximately constant, a frequency increase of up to 50% could be possible. An illustration of pipelining can be seen in figure 3.3.

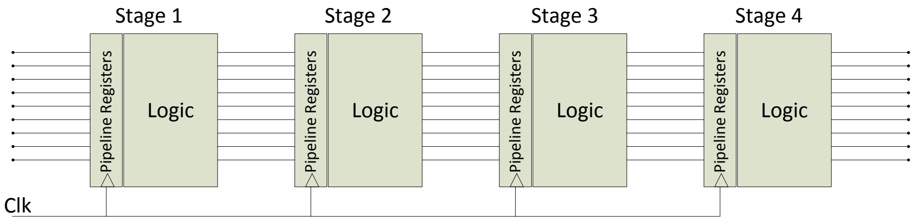


Figure 3.3: Pipelining [10]

3.3.2 Resource Sharing

One of the fundamental properties of the symmetric FIR filter is that coefficients are used twice[1] in the filter. However, in an FPGA with excessive potential performance this optimization can be extended to other logic blocks. By examining the coefficients of the filter one might see patterns which can be exploited to reduce area. The concept of resource sharing can be seen in figure 3.4.

In the case of figure 3.4, only one register is shared. Resource sharing can also be extended to include combinatorial logic as well. Depending on how the combinatorial logic is implemented in hardware, a multiplexer is often more efficient to implement than a multiplier, at least for large bit widths. For Actel, the architecture is based on multiplexers [16]. This means that in order to achieve an efficient physical synthesis, the design should also be based on multiplexers. One way of trading multipliers for multiplexers with resource sharing can be seen in figure 3.5. Note that in this design, timing is more critical than in the design from figure 3.4 due to the fact that a logic block will have to monitor and enforce sequentiality in the shared logic. Additionally, more registers are added to store the data before the final adder. However, these registers acts as pipelining on the critical path, shortening it by one adder. This will in turn increase the performance of the design, as discussed in section 3.3.1.

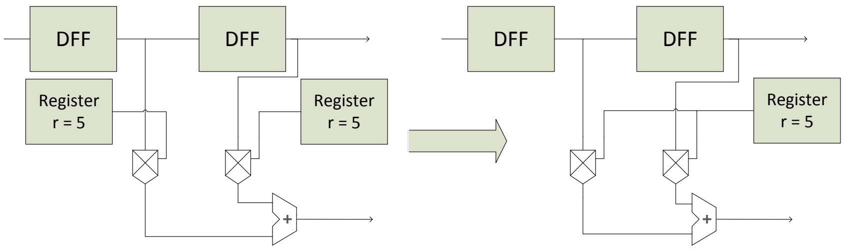


Figure 3.4: Resource Sharing Illustration

[1]In odd-order symmetrical FIR filters, the center coefficient is used only once.

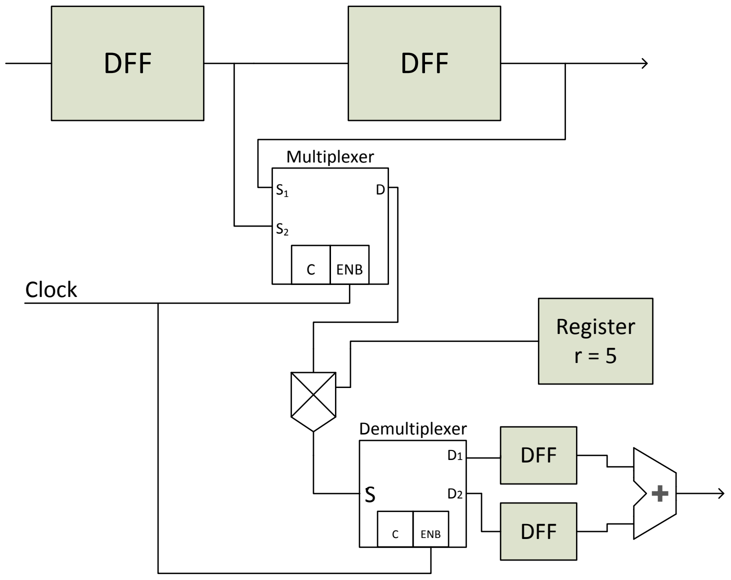


Figure 3.5: Resource Sharing, Continued

3.3.3 Constant Input Multipliers

In a FIR-filter, all of the multipliers are based on the fact that one input is constant. Since the possible outputs from one multiplier with one constant input port is based on the range of the second input, the output values can be predicted when compiling, using either logic or memory. This is usually done in the synthesis process, and the designer does not need to handle it. The number of operations needed in the multiplier depends on the number of "ones" in the multiplicand. The reason for this is because the product of two N-bit numbers can be expressed as [3]

$$P = A \cdot X = \sum_{k=0}^{N-1} a_k 2^k X \tag{3.3.2}$$

where $A = [a_0, \cdots, a_{N-1}]$. Equation 3.3.2 suggests that if A is constant for all values of X, and the number of non-zero elements in A is reduced, the complexity of the multiplication will also be reduced. For example if A is ”11101”, a_k will be non-zero for four instances of the summation, which results in four additions. Using two’s complement number representation, A would here be -3. If the sign of A is inverted, A would be 3, or 00011 in two’s complement. A would then only need two additions to complete the summation in equation 3.3.2. However, the product would now be $-P$ rather than P.

One feature of the FIR filter structure is that a multiplication is always followed by an addition. This can be seen in the following pseudo code from a symmetric FIR filter:

```
SmallAdder <= X[N] + X[L-N];
Multiplier <= SmallAdder * Coefficient;
```

```
LargeAdder[N] <= LargeAdder[N-1] + Multiplier;
```

A trait of using two's complement number representation is that addition is equally computational complex for both positive and negative numbers [3]. Therefore, one can use the product $-P$ instead of the original product P from the multiplier, and replace the adder with a subtracter. This can be achieved by introducing an inverter at one of the inputs of the adder. For numbers represented by many bits, this extra logic will be compensated by the reduction in multiplier logic.

To utilize the optimization described above, one can invert the constant FIR filter coefficient, given that this inversion leaves a coefficient where the majority of bits are zero. The filter behavior can now be described with the following pseudo code:

```
SmallAdder <= X[N] + X[L-N];
Multiplier <= SmallAdder * (-Coefficient);
LargeAdder[N] <= LargeAdder[N-1] - Multiplier;
```

Chapter 4

Existing Analog Filter

The current analog filter, which the proposed digital filter will replicate has the frequency response shown in figure 4.1. The filter is currently used in the Galileo Satellite Search and Rescue Transponder (SART). The purpose of the filter is to remove unwanted frequency components, such as noise and mixing products from the signal. The frequency modulated signal inside the passband of the filter can be located in one of several bands, defined by the specification for COSPAS-SARSAT 406 MHz distress beacons [17]. The signal information is irrelevant for this thesis, and it will therefore be treated as an isolated signal with a bandwidth of 100 kHz.

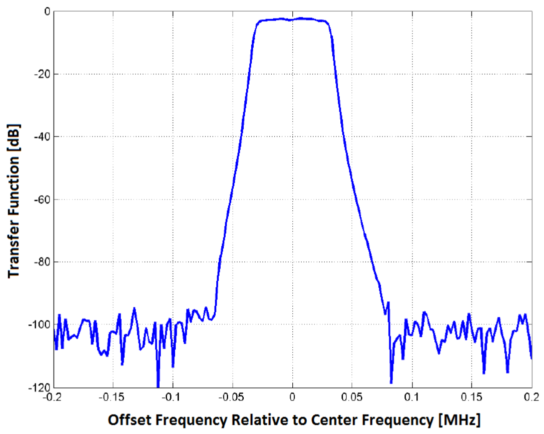


Figure 4.1: Current Filter Response

As previously mentioned, the filter is placed within the satellite SART unit. The basic functionality of the filter chain can be seen in figure 4.2. The actual input frequency of the SART, centered at 406 MHz - 406.1 MHz is mixed down to 70.98 MHz before it is received by this analog filter. The output of the SART is a bandpass filtered signal centered at around 1544 to 1545 MHz, in the *L-Band* [18], [19].

The expected frequency components of the filter from figure 4.1 can be seen in fig-

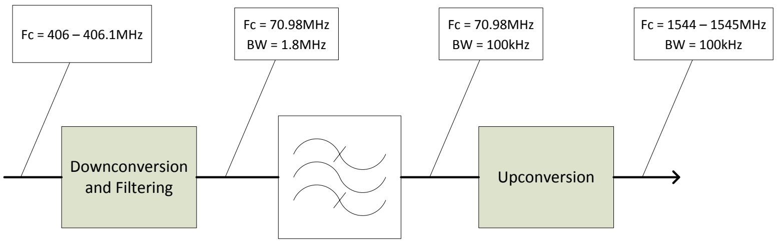


Figure 4.2: SART Converter Chain

ure 4.3. As one can observe, there are three signal components in the input signal. The strongest component is the signal at 70.98 MHz, which is the passband of the signal filtered at the front end SAW filter of the receiver. This SAW filter has a bandwidth of approximately 1.8 MHz, while the relevant signal has a total bandwidth of 100 kHz. The frequency components in the signal are unwanted. These components are the result of a mixing process, where the IF signal, which is placed at 406 MHz is mixed with a local oscillator of 335 MHz. From the problem description in section 1.2 it was given that undesired frequency components were to be expected at 741 MHz and 670 MHz, with a bandwidth of 1.8 MHz.

Furthermore, the input signal noise outside the 1.8 MHz bandwidth is assumed to be wide band and uniformly distributed, and one can assume that the stopband of the preceding SAW filter is sufficiently attenuated.

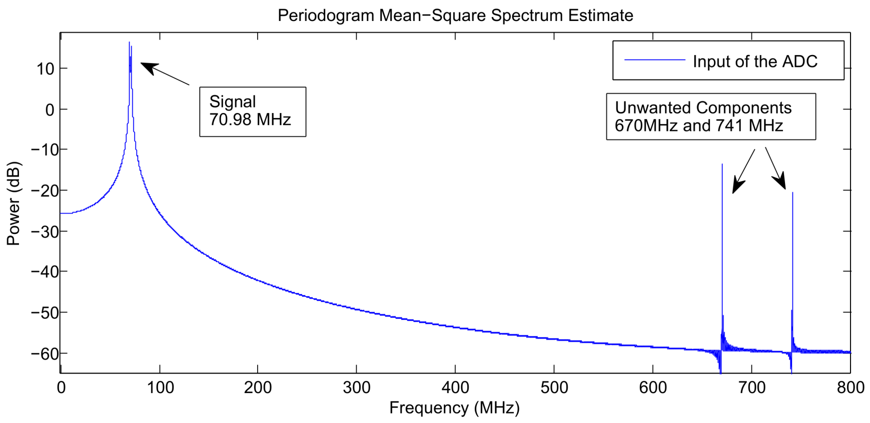


Figure 4.3: Expected Input Frequency Placement

Chapter 5

Methods and Tools

To understand the signal processing techniques, and to efficiently design the filter coefficients, *MathWorks Matlab R2012a* was used. Matlab-scripts were created to simulate the DSP-components in detail, for increased insight in functionality and testability. The HDL version of the filters were designed with *Aldec Active-HDL Student Edition version 9.1*. Two *C++* based programs were created to ease the design of the symmetric FIR filters. The first program, called *prototypefilter.cpp* creates a rough version of a FIR filter, where all components are declared explicitly. This filter makes it easier to perform optimizations at the lowest level. The second program is more interactive, and the generated filters can be customized from a terminal window without recompiling. This program is called *generatefilter.cpp*. This script uses more efficient *generate* statements for component declarations. However, the program still unwraps the generate statemetns which creates the large adders, as these will be altered for area-optimizations in section 7.2.2. Both programs requires the filter coefficients being present in a separate file, called *coeff.txt*.

C++ was also used to create a script which generates testbenches for the VHDL Filters. Separate scripts were created for testing the individual filters, the filters with reusable logic described in section 7.2.1, the complete filter system and for the complete filter system with extra wires for probing.

Digital simulations of the filters were performed in Active-HDL and *Mentor Modelsim SE-64 10.2*. *Synplify Premier with Design Planer E-2011.03-SP1* was used for HDL synthesis, to estimate and make sure that the design would satisfy the requirements in performance and area for the given FPGA. For FPGA-testing *Quartus II 30-bit 12.0 Web Edition* was used. This Quartus II installation used *SignalTap II Logic Analyzer* for communication with the FPGA. The FPGA was mounted on an Altera DE2-70 development board, and a *Terasic THDB-ADA* high speed ADC and *Digital to Analog Converter* (DAC) daughter card was mounted to the FPGA's *General Purpose Input/Output* (GPIO) interface.

All VHDL, Matlab and C++ code can be found in the enclosed zip-file.

Chapter 6

Proposed Digital Filter

As explained in section 2.2.1, a bandpass, linear phase FIR filter with transition bands of 50 kHz at each side, centered at 70.98 MHz would be infeasible to implement directly in any space qualified FPGA. When generating a symmetric FIR bandpass filter which fulfills the requirements from section 1.2 using Matlabs filter design software, the result is a filter consisting of 1685 stages. When synthesizing this filter in VHDL, using Synplify Premier, the specifications in table 6.1 are estimated. These results clearly states that this filter would not be possible to implement directly in the space qualified FPGAs discussed in section 3.2.3. Note that the no estimations were completed for the Virtex 4QV FPGA, as the Synplify Premier routing and fitting process failed as a result of the design size. The passband ripple of this filter was estimated to 1.8 dB.

FPGA	Combinatorial Cells	Register Cells	Clock Frequency
RTAX2000S	587%	220%	27.3 MHz
RTAX4000S	313%	117%	27.3 MHz
Virtex 4QV	N/A	N/A	N/A

Table 6.1: Direct Design of the Filter

However, as explained in chapter 4, the input of the filter is a bandpass signal with a bandwidth of 1.8 MHz, and it could be assumed that the stopband is sufficiently attenuated by a preceding SAW filter. This suggests that the sampling techniques described in chapter 2 would be well suited for this application. To be safe, it is assumed that the input signal has a bandwidth of 2 MHz, and not 1.8 MHz. Hence, guard bands of 100 kHz are added to each side of the signal so that the digital filter will be more robust against imperfections in the preceding filter chain.

6.1 Initial Sampling, Mixer and Filter

Since the filtering will occur inside an FPGA, the analog signal would need to be sampled and quantized with an ADC. As specified in the problem description, the target FPGA for this filter should be space qualified, preferably an Actel RTAX2000S. This FPGA does not have any DSP blocks, meaning that all arithmetic components will be implemented as logic, and no dedicated hardware will be used. This FPGA also has a limited performance, and the sampling frequency should be chosen accordingly. The sampling process should also move the center frequency towards *dc*.

The proposed ADC for this design should have an analog bandwidth above the highest frequency component of the signal, at 70.98 MHz + 1 MHz, but it should preferably also be lower than the mixing components at 670 MHz and 741 MHz. The output of the ADC should be able to handle a sufficient amount of bits so that the quantization noise discussed in chapter 2.1.2 will be kept to a minimum. An ADC which fulfills these requirements is the Texas Instruments *ADS5463-SP*, which is a space qualified 12-bit ADC with an analog bandwidth of 500 MHz [20]. The input voltage range of the ADC is $2.2\,V_{pp}$. It should be noted that there are several suitable alternatives to this ADC. According to equation 2.1.1, the step sizes of this ADC would be

$$\Delta = \frac{2.2\,\mathrm{V}}{2^{12}-1} = 0.537\,\mathrm{mV} \tag{6.1.1}$$

and the quantization noise, according to equation 2.1.6

$$P_{n,rms} = \frac{0.537 \cdot 10^{-3}}{\sqrt{12}} = 1.55 \cdot 10^{-4}\,\mathrm{W} \tag{6.1.2}$$

Using equation 2.1.11, the estimated SNR of the ADC output for a given rail to rail tone would be

$$SNR = 12 \cdot 6.02\,\mathrm{dB} + 1.76\,\mathrm{dB} = 74\,\mathrm{dB} \tag{6.1.3}$$

Note that the result from equation 6.1.3 is for ideal operation, with a perfect rail to rail tone. The datasheet for the ADC states an SNR of 65.3 dB relative to full scale (dBFS) [20].

When undersampling the input at the allowed 30 MHz ± 5 MHz, the resulting signal will be mirrored down to the baseband, which will be 15 MHz ± 2.5 MHz. The main focus when choosing the sampling frequency will be to analyze where the frequency components will be located in the baseband. To which location a frequency will be folded down to can be observed with the help of Matlab. A table of baseband scenarios of selected sampling frequencies within the allowed region defined in the problem description can be seen in table 6.2. None of these sampling frequencies would break the limitations of the sampling frequency in undersampling applications stated in equations 2.1.15 and 2.1.16, since $Fs >> B$.

It should be noted that the original 70.98 MHz input will be inverted in the baseband when the sampling frequency is lower than approximately 29 MHz. This is because the signal will located in an even numbered Nyquist zone in these cases. Equation 2.1.18 states that the degredation in SNR is dependent upon the number of signal replications when undersampling. With a sampling frequency of 28 MHz the SNR degradation would be

$$D_{SNR} = 10\log(6)\,\mathrm{dB} = 7.78\,\mathrm{dB} \tag{6.1.4}$$

A sampling frequency of 30 MHz would result in a degradation of

$$D_{SNR} = 10\log(5)\,\mathrm{dB} = 6.99\,\mathrm{dB} \tag{6.1.5}$$

meaning that if a sampling frequency above 29 MHz is chosen, a gain of 0.79 dB of SNR will follow.

$F_s \backslash F_{input}$	70,98 MHz	741 MHz	670 MHz
25 MHz	4,02	9	5
26 MHz	7,02	13/dc	6
27 MHz	10,02	12	5
28 MHz	13,02	13	2
29 MHz	12,98	13	3
30 MHz	10,98	9	10
31 MHz	8,98	3	12
32 MHz	6,98	5	2
33 MHz	4,98	15	10
34 MHz	2,98	7	10
35 MHz	0,98	6	5

Table 6.2: Resulting Frequency Components After Sampling [MHz]

When studying table 6.2, it is important to notice if the mixing components at 741 MHz and 670 MHz will be placed within the passband of the filter. This could lead to distortion in the signal. For example, when sampling at 28 MHz, one of the mixing components will be placed as close as 0.02 MHz from the center frequency of the signal. Even though these mixing components will be suppressed greatly by the analog bandwidth of the ADC, any interference in the passband should be avoided, which makes these sampling frequencies less attractive.

The next consideration when choosing the sampling frequency is how complex the filter at the next stage will have to be. This filter is, as with any FIR filter, dependent upon the sampling frequency and the steepness of the transition band. The main goal of this filter is to remove any frequency components outside the 2 MHz input bandwidth. This way, the signal and noise inside the 2 MHz bandwidth can be treated as one isolated signal, without significant noise in the remaining baseband.

Two approaches of reducing the complexity of the initial sampling, mixing and filtering were considered. The first approach is to sample so that the signal and the undesired components are placed as far apart from each other as possible. This is the case for sampling frequencies 33 MHz to 35 MHz. The other approach is to design the relevant signal and the undesired components within a small region, possibly overlapping each other in the noise region of the 2 MHz bandwidth. This would be the case of for example the 30 MHz and 32 MHz sampling frequencies. The signal could in these cases be mixed down as one isolated signal consisting of both the relevant signal and the noise.

A sampling frequency of 30 MHz was chosen for the ADC and FPGA, as this frequency would mean that the signals could be mixed down as one isolated signal, due to the fact that all frequency components in the baseband would be located in the upper baseband. The 30 MHz sampling frequency will also have the benefit of a reduced degradation in SNR after sampling compared to lower sampling frequencies, as shown by equation 6.1.5. Additionally, since the FPGA is expected to be strained for performance, the design would be best suited for a low sampling frequency, which makes the 30 MHz sampling frequency more attractive than the higher frequencies.

By using this sampling frequency, the input will be aliased down as shown in figure 6.1. The mixing components will be placed at 9 MHz and 10 MHz in the baseband, which

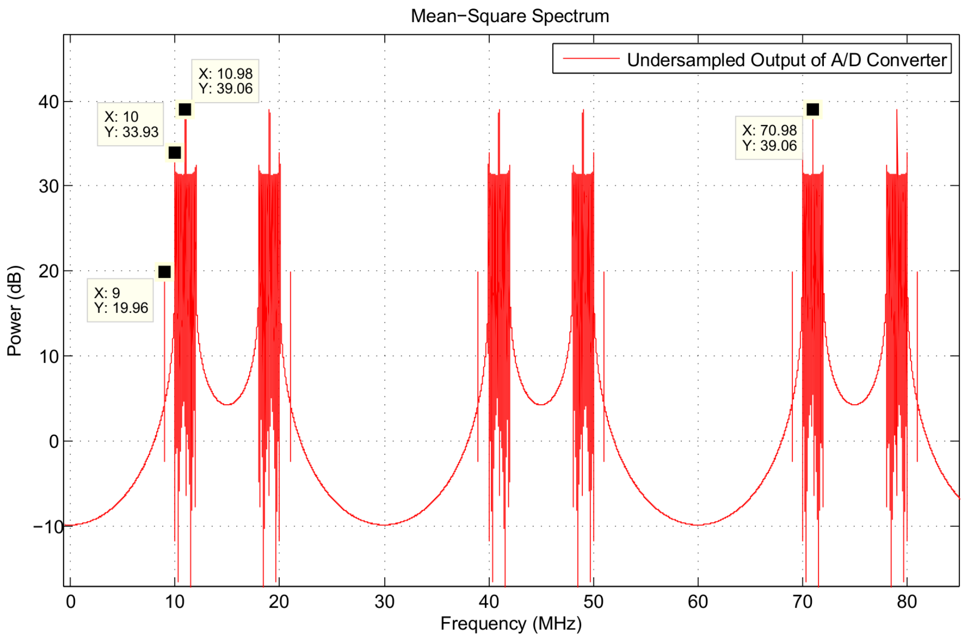


Figure 6.1: Aliased Signal at 30MHz Sampling

means that one of the components will be placed within the 2 MHz bandwidth. Unlike the mixing components illustrated as single frequencies in figure 6.1, the real mixing components also have a 1.8 MHz bandwidth. This means that the mixing component centered at 10 MHz will have some degree of noise at

$$10\,\text{MHz} + \frac{1.8}{2}\,\text{MHz} = 10.9\,\text{MHz} \tag{6.1.6}$$

These frequency components will not interfere with the real signal, now placed at 10.98 MHz $\pm\frac{100}{2}$ kHz, since the lower limit of the real signal is placed at 10.93 MHz.

The next step of the system is to mix down the center of the signal. As explained in section 3.2.1, this can be achieved quite efficiently by choosing the relationship between F_S and F_{LO} carefully. By choosing the mixing factor $\frac{F_s}{F_{LO}}$ to 4, the signal would only need to mix with four constant samples of the LO sine wave. These samples could be set to 1, 0, -1 and 0. This operation is trivial to implement in hardware, as the only logic needed is a counter, a multiplexer and a simple inversion and addition with 1 when the input is multiplied by -1. The result of this mixing can be seen in figure 6.2, where the center frequency now is placed at 3.48 MHz.

At this point, the signal should be filtered with a lowpass filter. All frequencies above 3.48 MHz ± 50 kHz are defined as unwanted. The lowest frequency of the upper mixing component from figure 6.2 is placed at 10.56 MHz which gives a filter transition band of

$$10.56\,\text{MHz} - 3.53\,\text{MHz} = 7.03\,\text{MHz} \tag{6.1.7}$$

at a sampling frequency of 30 MHz. This filter, called *Coverfilter*, was realized as a fixed-point lowpass symmetric FIR filter with 14 tap weights. The frequency response of the

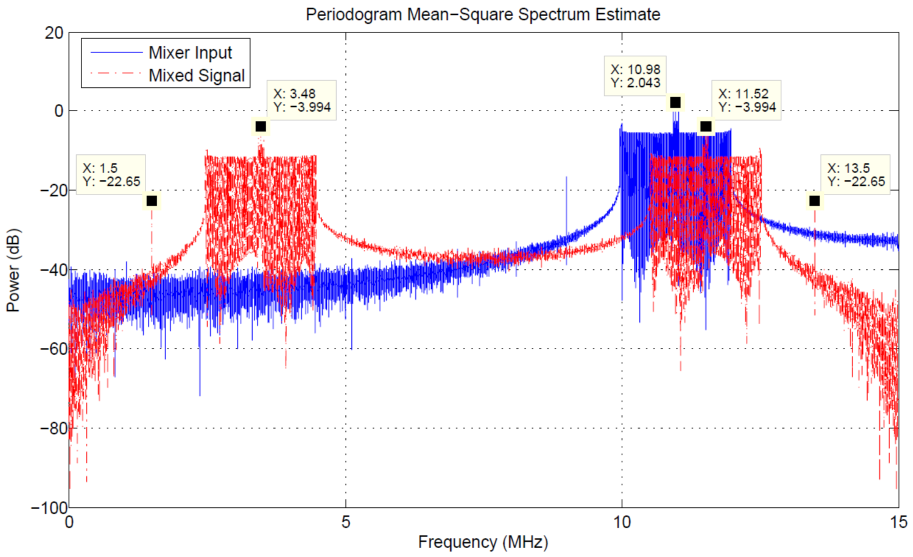


Figure 6.2: Down-Mixed Signal

quantized Coverfilter, as well as the reference full precision Coverfilter can be seen in figure A.1 in appendix A. When quantizing the filter to fixed-point representation, a coefficient resolution of 13 bit was found suitable for the required attenuation. When comparing the normalized fixed-point response of the Coverfilter with the full precision filter response, one can see a significant quantization error in the stopband. The passband is more or less preserved after quantization. The signal after this filter can be seen in figure 6.4.

A block diagram of the mixer and coverfilter can be seen in figure 6.3.

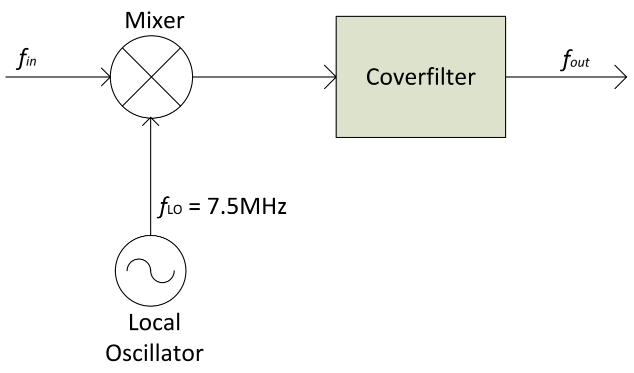


Figure 6.3: Mixer and Coverfilter

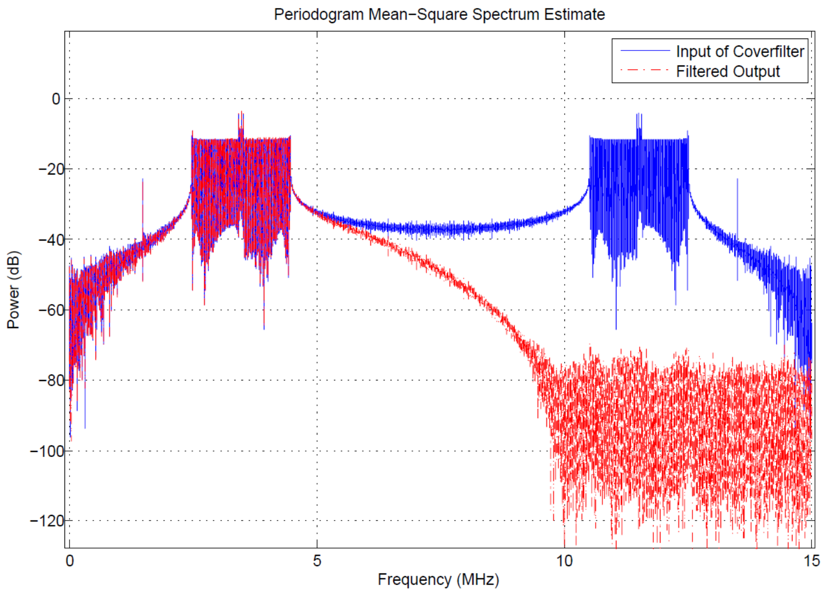


Figure 6.4: Signal after Coverfilter

6.2 Signal Decimation

To reduce the complexity of the filter, a decimation stage is needed. As explained in section 2.4, decimation is a process consisting of two operations. First, the signal is filtered with either a highpass filter or a lowpass filter, depending on how the signal is decimated. The second step is to reduce the sampling frequency by removing samples at a given rate.

The first decimation stage uses the Coverfilter from the mixing process as a decimation filter. As seen in figure 6.4, this lowpass filter attenuates all present frequency components above approximately 4.5 MHz. The signal can now be successfully decimated by a factor of 3, resulting in a sampling frequency of 10 MHz. The frequencies ranging from 5 MHz to 15 MHz in the original signal will be mirrored down into the new baseband. However, these components are suppressed sufficiently by the Coverfilter, which suggests that they will ideally not affect the resulting downsampled signal. This process can be seen from the Matlab simulation in figure 6.5.

The next decimation will be by a factor of 2. This stage is somewhat different from the first decimation stage, in that it is the signal in the upper frequency band ($F_s/4 < F < F_s/2$) of the downsampled signal from figure 6.5 which is relevant. This means that the frequency components currently below $F_s/4$ will have to be attenuated before downscaling the sampling frequency. As this stage will only downsample the signal by a factor of 2, the resulting signal is expected to have a new center frequency of

$$F_{s,new} - F_{center} = 5\,\text{MHz} - 3.48\,\text{MHz} = 1.52\,\text{MHz} \qquad (6.2.1)$$

Since the lowest (undesired) frequency component before this decimation is centered at 1.5 MHz, it will be placed only 20 kHz apart from the new center frequency, which is

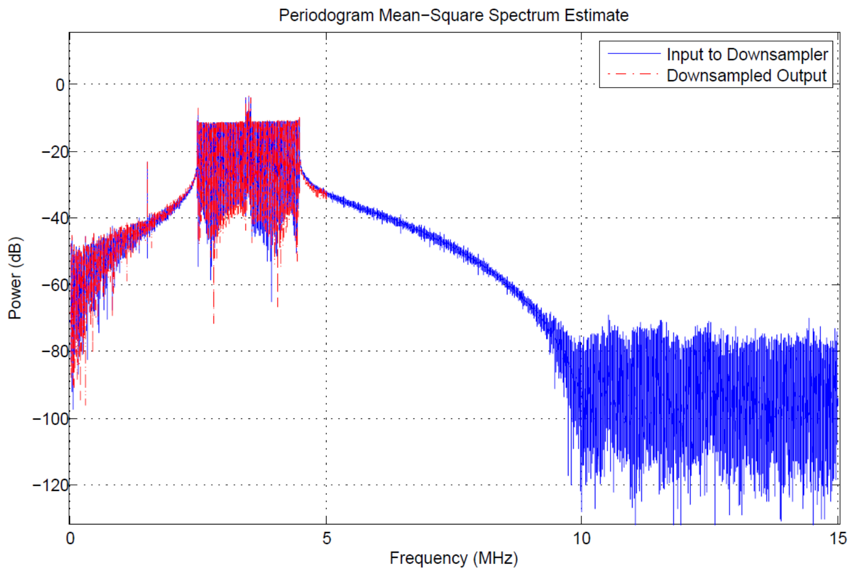


Figure 6.5: Signal at F_s = 30 MHz and F_s = 10 MHz

unacceptable. To remove this frequency component, as well as reducing the noise floor around 1.52 MHz, a highpass filter is implemented before the downsampling process. A suitable symmetric FIR filter was designed and converted to fixed-point arithmetic, with 13 bit precision coefficients. The filter phase and frequency response can be seen in figure A.2. This filter will henceforth be referred to as *HdDec2*. After the filtering the signal is downsampled by 2, which results in the signal shown in figure 6.6. Note that the signal is inverted at this point, meaning that the previously highest frequency components of the signal now are the lowest, as explained in section 2.4

The next two stages of the decimation chain are identical to the second stage, with the exception of the decimation filter characteristics. The first of these two stages decimates by two, leaving the center frequency of the signal at

$$2.5\,\text{MHz} - 1.52\,\text{MHz} = 0.98\,\text{MHz} \tag{6.2.2}$$

This is a frequency which is occupied by the noise previously defined within the isolated input signal, as seen in figure 6.6. This suggests that the 2 MHz bandpass signal no longer can be seen as an isolated signal, but as a 100 kHz signal surrounded by noise. To successfully implement this decimation stage, the noise currently located at 0.98 MHz ± 50 kHz should be filtered out. Ideally, this filtering can be done with a bandstop filter where only the region 0.98 MHz ± 50 kHz is attenuated. However, seeing that bandstop filters uses significantly more resources than a single transition filter, a highpass filter is introduced before the downsampling stage. The filter response of this filter can be seen in figure A.3, and the signal after the decimation stage can be seen in figure 6.7. The filter at this decimation stage was implemented as two cascaded highpass filters, called *HdDec3*.

At this point the signal is again inverted in the frequency domain, resulting in a cancellation of the previous inversion.

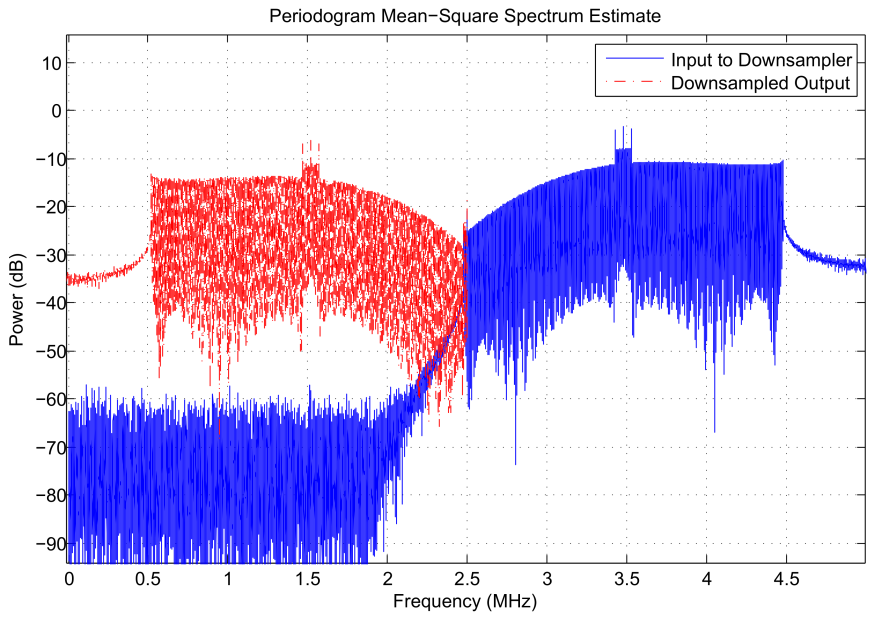


Figure 6.6: Signal at $F_s = 5$ MHz

As with the previous two stages, the final stage of decimation decimates by a factor of 2. The signal after this stage will then be centered at

$$1.25\,\text{MHz} - 0.98\,\text{MHz} = 270\,\text{kHz} \tag{6.2.3}$$

This is also a frequency currently occupied by unfiltered noise, which means a new decimation filter needs to be implemented. The filter response of this filter can be seen in figure A.4. This last decimation filter is called *HdDec4*, and it is followed by a final downsampling by a factor of 2. The signal is now inversed in the frequency domain.

6.3 Bandpass Filtering

At this point, the sampling frequency is low enough for the ratio between the sampling frequency and the steepness of the transition bands to realistically achieve a good bandpass filtering, with a reasonable number of filter tap weights. The bandpass filter coefficients were, as with the decimation filters, designed with the Parks-McClellan algorithm in Matlab. The response of the bandpass filter, *HdBand*, can be seen in figure A.5 in the appendix. This filter is significantly larger than the rest of the filters, due to the fact that it has two transitions between passband and stopband, as well as a much higher attenuation than the decimation filters. The coefficient precision of the HdBand filter was set to 18 bit. The signal after the bandpass filtering can be seen in figure 6.8. As this filter was quite complex with a very high attenuation, some ripple in the passband was to be expected. This ripple was determined to 1.8 dB. The filter specifications from section 1.2 did not mention any requirements of the passband ripple.

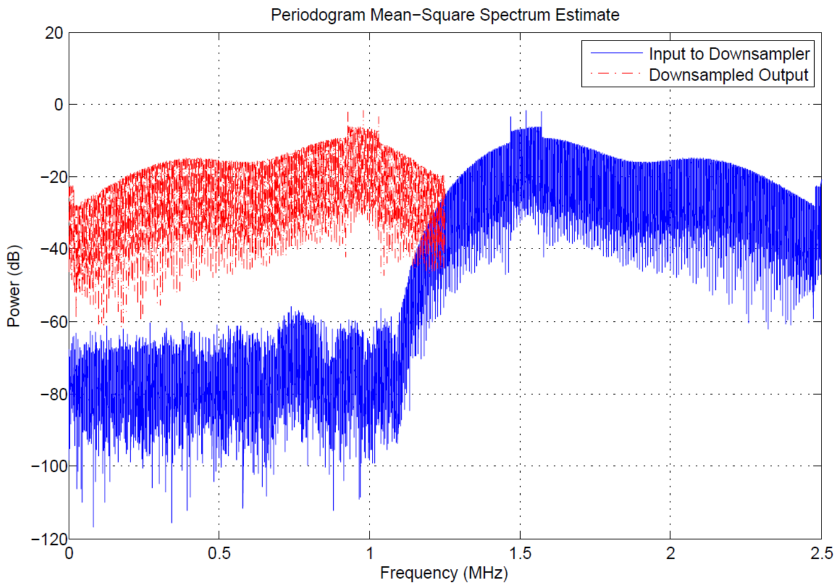


Figure 6.7: Signal at $F_s = 2.5$ MHz

The signal has now been filtered according to the specifications. However, since the next step of the SART will be to mix the signal up to the L-band before transmission. A signal placed at 270 kHz will produce mixing components with a very close vicinity relative to the L-band frequency, as explained in section 2.3. The undesired output of the mixer would consequently require a very complex analog filter to remove, which would make the whole digital filter more or less redundant. Therefore, an interpolation process will be needed to scale up both the sampling frequency and the signal center frequency before the digital processing is complete.

6.4 Signal Interpolation

As a design choice, the upsampling factors were decided to be the same as in the decimation process. The benefit of this approach is that the filters designed for the decimation process can be reused in the interpolation process, to reduce the number of filters which had to be designed and implemented in hardware. However, the filters are implemented as separate physical logic components, so this will not result in any performance- or spacial gains by itself. The resulting center frequency placements and sampling frequencies of the interpolation process can be seen in table 6.3.

At the 30 MHz clock domain, the interpolation product centered at 3.48 MHz is chosen as the desired signal, and the signals at 6.52 MHz and 13.48 MHz are defined as unwanted. The reason for this was because the signal at 3.48 MHz would not be inversed compared to the input signal, and a lowpass filter which removes the other two unwanted signals would be quite simple to implement. This filter was designed as two smaller FIR filters, called *HdOut* and *HdOut2*. The filter responses of these filters can be seen in the appendix, in

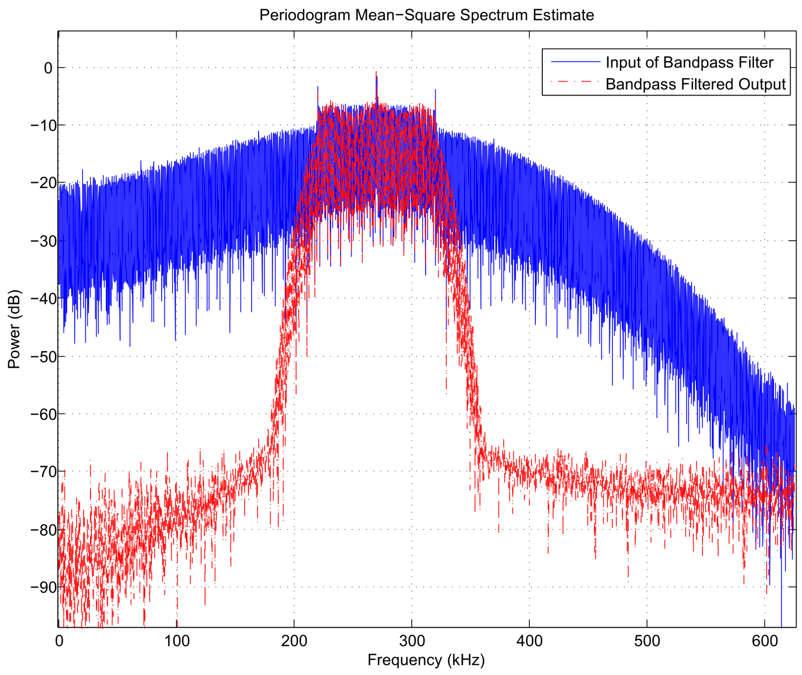


Figure 6.8: Signal After Bandpass Filtering

Stage	Int. Factor	F_s	Center Frequency	Unwanted Component(s)
1	2	2.5 MHz	0.98 MHz	0.27 MHz
2	2	5 MHz	1.52 MHz	0.98 MHz
3	2	10 MHz	3.48 MHz	1.52 MHz
4	3	30 MHz	3.48 MHz	6.52 MHz and 13.48 MHz

Table 6.3: Interpolation Stages

figures A.6 and A.7 respectively.

After this last filtering process, the signal could have been mixed up to the original input frequency at 10.98 MHz. This process would have mixed the signal with a local oscillator of $10.98\,\text{MHz} - 3.48\,\text{MHz} = 7.5\,\text{MHz}$, which is similar to the mixing at the input. This step was dropped due to the following reasons:

- Although this mixing process could reuse the mixer at the input, an additional filter at the output of the mixer would be required, to remove any mixing components at $7.5\,\text{MHz} - 3.48\,\text{MHz} = 4.02\,\text{MHz}$. This could be costly in an already area strained design.

- The 30 MHz clock domain is the critical clock domain by far, and adding additional logic along this path might cause the performance requirements to be broken.

- The goal of the SART system, as described in chapter 4 is to filter out the distress signal and transmit it at a much higher frequency. Therefore, the signal will be mixed up at a later stage in the SART chain, and the if the output frequency of this filter is 3.48 MHz or 10.98 MHz would be less significant given that the unwanted mixing components could be removed efficiently with for example a SAW filter.

All the processes and filters described in sections 6.1 to 6.4 were combined and implemented in VHDL as seen in figure 6.9. The VHDL-code can be seen in the enclosed zip-file. This parent filter structure will be called the *Masterfilter*. Since all internal components in the Masterfilter has a linear phase response, due to the exclusive use of internal FIR filters, it is assumed that the Masterfilter itself has a linear phase response. The reason for this is that there are no feedback nodes in the filter, which suggests that the principle of superposition applies [21].

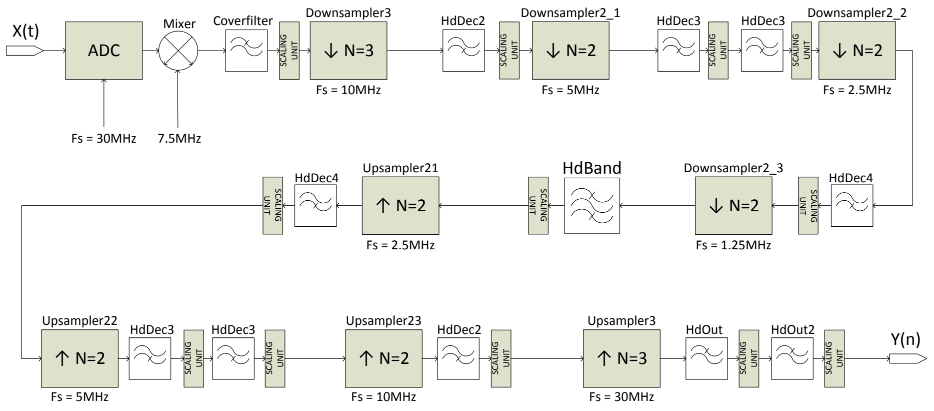


Figure 6.9: The Masterfilter

It should be noted that the DAC process and mixing following the FPGA filter is not discussed in this thesis, as the main focus lies on the FPGA digital signal processing. The ADC process is handled, since the undersampling which occurs as a result of this process facilitates the simplified filtering at the 30 MHz system clock frequency. Since the sampling frequency of the FPGA is set to 30 MHz, it would not be possible to transmit the filtered signal at the original 70.98 MHz center frequency. If this had been a strict requirement, it would be possible to send the output of the FPGA and DAC through an analog mixer followed by an analog filter which removes the undesired mixing component. However, as described in chapter 4, the output of this filter is destined to be mixed up to the L-band. This suggests that the intermediate frequency 70.98 MHz is not necessary for the functionality of the SART.

6.5 Clock Hierarchy

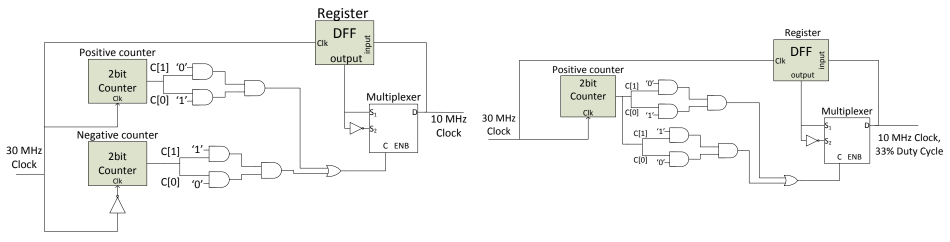
To be able to process the signal in different sampling domains, an efficient clock generator would be needed. In a timing-critical digital system such as this, all clock domains should preferably be implemented using PLLs, as these ensures phase and frequency stability. However, in order to make the design as portable as possible, the clock generator was also manually implemented.

Most clock domains in the filter were implemented with a simple boolean variable and an inverter, since the division of the frequency was by a factor of two. An example of this is the generation of the 5 MHz clock signal, which uses the 10 MHz signal as a reference and divides it by two. This can be seen in the following VHDL code, where *tmp5* is a boolean signal declared within the architecture.

```
--5MHz clock generator
clk_proc3:process (clksig_10)
begin
  if (rising_edge(clksig_10)) then
    tmp5 <= not(tmp5);
  end if;
end process;
clksig_5 <= tmp5;
```

Note that the generated clock signal (*tmp5*) is stored in a register, and that the actual 5 MHz clock signal (*clksig_5*) reads its value from this register. By adding this extra register, the clock source has a stable value to transmit, and clock jitter is reduced to a minimum.

In order to generate a 10 MHz clock signal from a 30 MHz source, a division by three is needed. This is not as trivial, as FPGA-synthesizable flip-flops cannot be sensitive to both rising- and falling edge of the reference clock. Therefore, two processes are needed, where one uses an inverted clock as an input, while the other uses the original clock. This approach can be seen in VHDL code in appendix E.1. This way of generating a 10 MHz clock signal can also be seen in figure 6.10a.



(a) 10MHz Clock Signal With 50% Duty Cycle (b) 10MHz Clock Signal With 33% Duty Cycle

Figure 6.10: Generation of 10 MHz Clock Signal

Another approach for achieving a 10 MHz clock from a 30 MHz reference clock is to use duty cycling. Since this clock is used for sampling, and only the rising edge of the clock is used as reference, it does not matter how long the signal stays high or low as long as the clock edge is detected. The functionality is only defined by the clock period. This attribute allows a simpler clock generator where the first clock cycle of the 30 MHz clock generates a high value on the 10 MHz clock, and the two next periods of the 30 MHz clock sets the 10 MHz clock low. An illustration of this can be seen in figure 6.10b. Note that one less counter is needed in this design.

6.6 Filter Specifications

6.6.1 Group Delay

Since the filter system consists exclusively of FIR filters the group delay of the filter will ideally be constant, as discussed in section 2.2.1. The number of stages, as well as the delay of each filter can be summarized with table 6.4. The group delay was calculated using equation 2.2.4. These delays adds up to a total filter delay of 55.7μs. This delay

Filter	Stages	F_s [MHz]	Number	Group Delay
Coverfilter	13	30	1	217 ns
HdDec2	16	10	2	1.6μs
HdDec3	22	5	4	8.8μs
HdDec4	20	2.5	2	8μs
HdBand	91	1.25	1	36.4μs
HdOut	27	30	1	450 ns
HdOut2	13	30	1	217 ns

Table 6.4: Group Delay Summary

is, as a result of the FIR structure, constant for all frequencies. The remaining DSP-components in the filter, such as the mixer are not included in this calculation, as the delay of these components is much lower than that of the filters.

6.6.2 Overflow

Overflow could occur in two general places in each individual filter, either in the internal wire connections in between the large adders, or at the output of the filter. A multiplication in VHDL between an N-bit value and an M-bit value will require a product of width $N+M$ bits, which will not overflow. However, if several products of width $N+M$ are added together, which is the case in an FIR filter, there is a chance of the sum of these values being larger than $N+M$ bits. To calculate if this could be a possibility in any of the proposed filters, equation 3.1.4 was used to determine the maximum allowed input value which would not result in overflow. For example, the Coverfilter has an internal bit width of 26 bit. The absolute sum of the coefficients, $\sum|f[k]|$, amounts to 5924, which means the maximum allowed input which does not result in overflow will be

$$X_{max} = \frac{\pm 2^{26-1}}{5924} = \pm 5664.15 \tag{6.6.1}$$

To represent ±5664 as a binary number,

$$\lceil \log_2(5664.15) + 1 \rceil = 13\,\text{bit} \tag{6.6.2}$$

will be needed. Since the input of the Coverfilter is only 12*bit*, an internal overflow will never occur. Overflow-calculations for each individual filter can be seen in table 6.5.

The second location which could generate overflow is at the output of each internal filter, as a result of bit width reduction. As explained in section 3.1.1, this could be countered by first dividing the output by a constant *scalingfactor*, lowering the output dynamic

Filter	Int. Bitwidth	$\sum \lvert f[k] \rvert$	X_{max}	$\lceil \log_2(X_{max}) + 1 \rceil$	Input Width
Coverfilter	26	5924	±5664	13	12
HdDec2	28	12152	±11044	15	14
HdDec3	26	3904	±8594	15	14
HdDec4	27	5641	±11896	15	14
HdBand	33	1045256	±4110	14	14
HdOut	26	3194	±10505	15	14
HdOut2	21	96	±10922	15	14

Table 6.5: Overflow Summary

range. The output bit width can then safely be reduced using the VHDL statement *resize*. The value of the scalingfactor was initially set to 1, and a proper value was determined from simulations of the individual filters.

Chapter 7

Synthesis and Resource Optimization

7.1 Initial Synthesis

The first synthesis of the Masterfilter indicated that the filter design proposed in chapter 6 would not fit into the RTAX2000S FPGA. This was expected, due to the limitations in the FPGA architecture, explained in section 3.2.3. The initial synthesis of the systems area and clock domains can be seen in table 7.1.

Parameter	Requirement	Estimation	Resource Useage
clksig_1_25	1.25 MHz	18.4 MHz	6.8%
clksig_2_5	2.5 MHz	27.8 MHz	9%
clksig_5	5 MHz	31.1 MHz	16.1%
clksig_10	10 MHz	32 MHz	31.3%
CLK_30MHz	30 MHz	27.4 MHz	109.5%
C-Cells	21504	24306	113%
R-Cells	10762	4833	45%

Table 7.1: Initial Synthesis Report

As seen in table 7.1, both the CLK_30MHz and the C-Cell estimation failed to fulfill the filter requirements at this stage.

7.2 Optimizations in HDL

To counter the issues described in section 7.1, several optimizations were concidered and implemented in the filter.

7.2.1 Resource Sharing

Prior to the initial synthesis in section 7.1, the only optimization in filter resource usage was the use of symmetric filter structures, as shown in section 2.2.1. However, as table 7.1 suggests, further area optimizations would be needed.

An efficient method of optimizing in the filter structure would be to trade off performance for area. As described in chapter 6, similar filters were used in both the decimation stages and the interpolation stages. This means that the filter coefficients will be equal in both stages, and they can be shared between the decimation- and interpolation filters. One can even take this a step further, by allowing both filters within the same clock domain to share multipliers and adders. This would require a reference clock with twice the original frequency in addition to some controll logic, as discussed in section 3.3.2.

As the initial synthesis report indicated in section 7.1, performance is not an issue at certain clock domains, and the reduced multipliers and adders from the resource sharing more than makes up for the extra added logic. Consequently, adding this optimization has the potential of reducing the area of the total filter significantly. As a rough measurement, two equal N-stage filters will initially require N multipliers, N small adders and N large adders. By using resource sharing, this will be reduced to $N/2$ multipliers, $N/2$ small adders and $N/2$ large adders. However, the filter delay elements will not be applicable for resource sharing, as this would affect the functionality of the filter. These new (reusable) filters with shared logic are called *HdDec3RU* and *HdDec4RU*. Note that the 2.5 MHz clock domain will now only be used for upsampling and downsampling, and not in any of the filters. This will lower the strain on this clock domain. However, the 10 MHz clock domain will now drive both the HdDec2 filter and the two HdDec3RU filters. This will increase the strain on this domain.

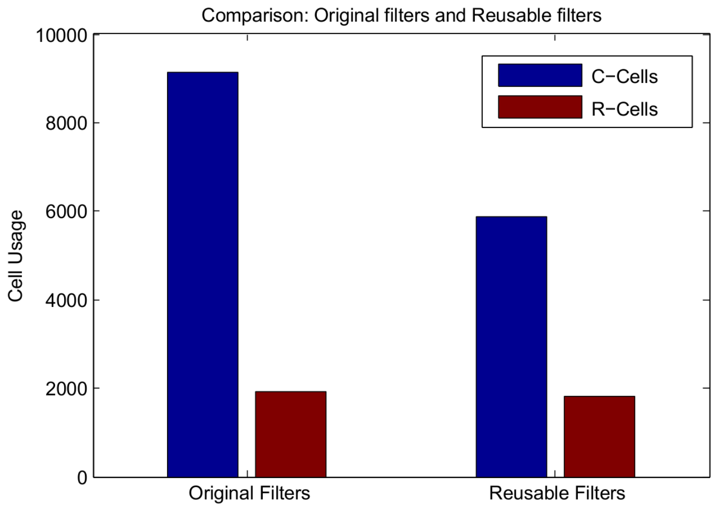


Figure 7.1: Improvement of Reusable Filters

An illustration of a filter with shared logic can be seen in figure C.1 in the appendix. Size estimations with and without the reusable filters conducted with Synplify Premier can

	Combinatorial Cells	Register Cells
HdDec3RU	1955	591
HdDec4RU	1955	647
HdDec4 + HdInt1	2948	604
HdDec3 + HdInt2	3094	662

Table 7.2: Optimization by Filter Reusage

be seen in table 7.2. Figure 7.1 shows the combined HdDec3 and HdDec4 area usage in the Masterfilter, compared with equivalent use of HdDec3RU and HdDec4RU filters. The resource usage was measured for the RTAX2000S FPGA.

7.2.2 Filter Coefficient Optimization

As explained in section 3.3.3, the filters in this system uses multipliers with one constant input which have a good possibility of being optimized for both area and performance. However, as synthesis tools tend to have a seemingly sporadic behavior, and it is not always predicable where an optimization might cause a gain or loss in performance, each filter coefficient needs to be inspected individually. This was done by first looking at the filters which did not initially satisfy the requirements in performance. Each coefficient of this filter was inverted, and the corresponding adders after the coefficient multiplication were replaced with subtracters. The filter was then synthesized and the estimated size and performance was recorded. After this test was conducted for the individual coefficients, the inversions which improved the performance were combined and the circuit was tested again. The improvements of each filter after these changes can be seen in table 7.3. The synthesis targeted the RTAX2000S FPGA, and no synthesizer optimization settings were enabled during these tests.

Filter	Size[1]	Imp. Size[1]	Gain	Original Clk	Imp. Clk	Gain
HdCover	878	838	4.56%	35.2 MHz	37 MHz	5.11%
HdDec2	1438	1279	11.06%	33.6 MHz	38.5 MHz	14.58%
HdDec3RU	2032	1745	14.12%	32.7 MHz	35.5 MHz	8.56%
HdDec4RU	1955	1807	7.57%	8.8 MHz	17 MHz	93.18%
HdBand	9379	8987	4.18%	16.6 MHz	17 MHz	2.4%
HdOut	2070	1858	10.24%	29.4 MHz	31.4 MHz	6.8%
HdOut2	654	510	22.02%	63.1 MHz	74.7 MHz	18.38%

Table 7.3: Original and Improved Filter Coefficients

7.2.3 Optimizing for Performance

Although the limitations of the FPGA set strict requirements for the area of the design, the timing of the filter was also critical to achieve the desired outcome. For the filter to work correctly, the delay of the critical paths in each filter would need to be below the period of

[1]Size measured in C-Cells

the filter clock. However, this does not necessarily mean that every internal filter would need to be able to achieve a 30 MHz clock frequency. As there are several clock domains within the filter system, each filter is only required to uphold the frequency of which it is designed for. This would suggest that only the 30 MHz filters should be optimized with pipelining, as this was the only domain which struggled to uphold the timing requirements, as seen in table 7.1.

Clock Domain	Before Pipelining[1]	After Pipelining	Improvement
1.25 MHz	18.5 MHz	17.7 MHz	-4.5%
2.5 MHz	149.1 MHz	164.1 MHz	10%
5 MHz	16.3 MHz	16.5 MHz	1.2%
10 MHz	20.5 MHz	15.7 MHz	-23.4%
30 MHz	29.9 MHz	32.8 MHz	9.7%

Table 7.4: Improvement after Pipelining

After applying a pipeline stage at the end of the 30 MHz filters, the improved timing shown in table 7.4 was achieved. The pipeline was added after the last of the adders in the combinatorial part of the filter, before the scaling unit. The filter was now within both the area and performance requirements of the FPGA. The improved filter area estimation targeted at the RTAX2000S FPGA can be seen in table 7.5.

As discussed in section 3.2.3, the Actel RTAX2000S FPGA was not the only FPGA suited for this project. Therefore, two other space qualified FPGAs of equal or greater performance and area were tested with the filter system. In these tests, the FPGAs' area usage and performance were estimated for both the filter design with reusable filters (w/RU) and without the reusable filters (wo/RU). The results can be seen in figure 7.2, where the critical area characteristics were logged. In the RTAX2000S and RTAX4000S, the area was determined by the C-Cell count, and in the Xilinx Virtex 4QV the area was determined by the number of LUTs used.

Parameter	Requirement	Estimation	Resource Usage
C-Cells	21504	20487	95%
R-Cells	10762	4877	45%

Table 7.5: Optimized Synthesis Report

The Masterfilter after all the optimizations can be seen in figure 7.3.

[1]Timing estimations performed before the pipelining was added, but after the reusable filters from section 7.2.1 and coefficient optimizations from section 7.2.2 were implemented

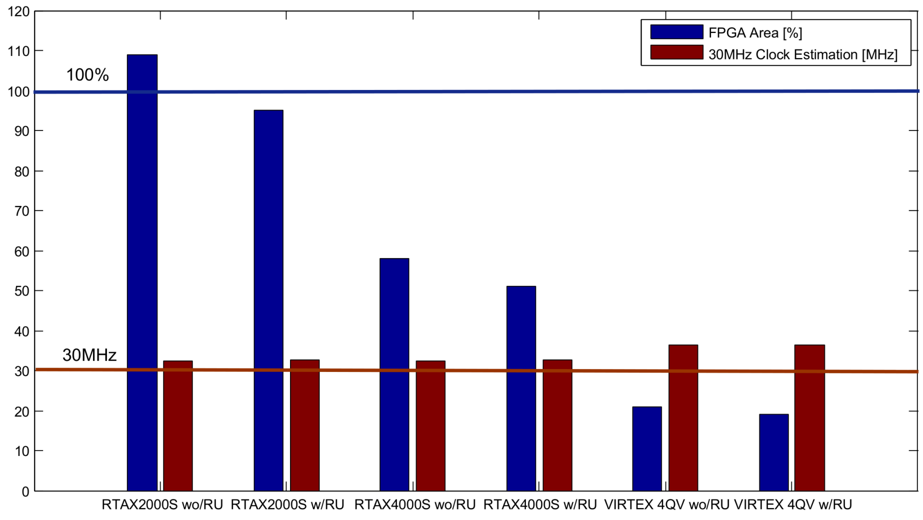


Figure 7.2: Area Usage and Performance for Space Qualified FPGAs

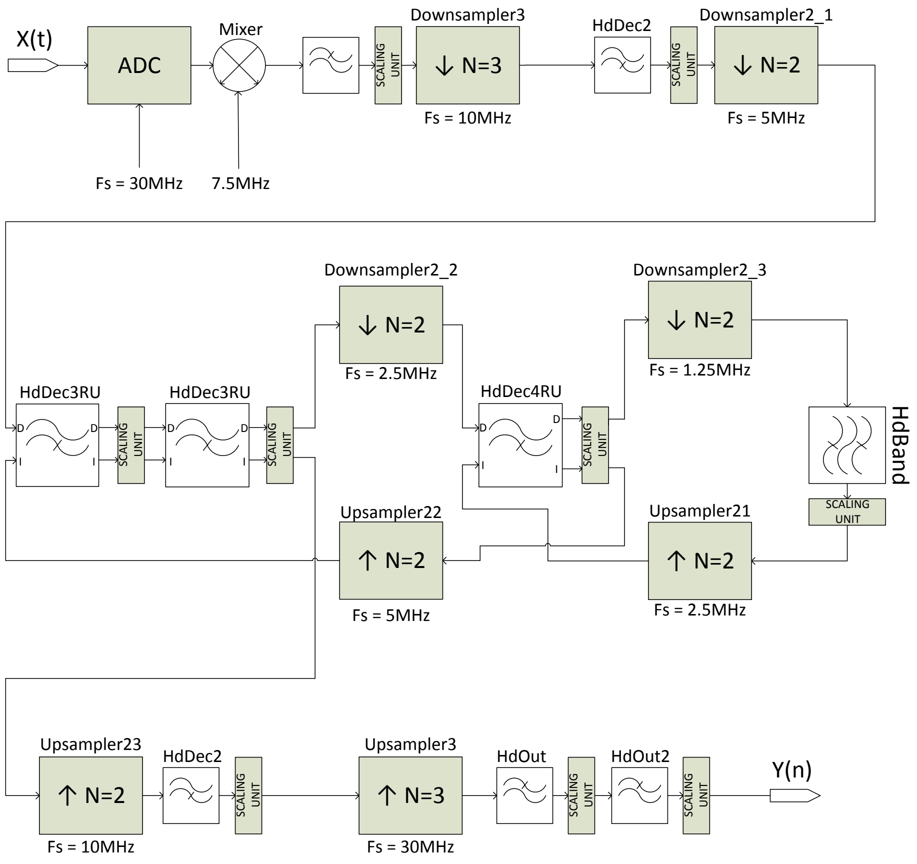


Figure 7.3: The Optimized Masterfilter

Chapter 8

Simulation and Verification

Several different testing approaches were used in the verification of the digital filter system. These tests and simulations included both simulation of functionality in the Modelsim environment, and actual testing on an FPGA.

8.1 Digital Simulation

After a filter was designed in Matlab, and the parameters were translated into the proper VHDL code, simulations were used to determine the correctness of the implementation. The digital simulations were divided into three levels of tests:

1. Impulse Responses
2. Tone testing
3. Spectrum analysis

8.1.1 Impulse Responses

These structural tests covered the basic correctness of the implementations in VHDL. This was done by sending a scaled *Dirac delta function* into each internal filter [22]. In practice this means sending the value 1 into the filter, but in the digital filter where the output is scaled down by a factor of N, the input should also be set to N. For example, if the output is scaled down five bit positions to remove overflow, N will be set to $2^5 = 32$. After one clock cycle the input is set back to zero. When this N has propagated through the system, the filter should have transmitted the filter coefficients, or the impulse response. This test would uncover most implementation errors in the filter, related to the coefficients and delay line. In addition to the filters being tested, the mixer, clock generator, downsamplers and upsamplers were also examined. The results of these simulations can be seen in appendix B. The results from the bandpass filter simulations can be seen in a digital waveform file in the enclosed zip-file.

8.1.2 Tone Testing

The second test of the internal filters included testing of individual tones in both the passbands and stopbands of each filter.

This was done by testing with a series of signals consisting of a single constant tone, and observing the output of the filter. This was performed for a handful of selected tone frequencies. This test would give a quick way of uncovering errors early on in the design phase, and it could be used to uncover issues with overflow and bit-precision. Overflow at the output would mainly occur in the passband, so rail-to-rail tones in these regions could indicate roughly the amount of bits needed to safely represent the output. This would also help determine the ideal Scalingfactor when resizing the signal from the size of the internal filter calculations, to the size of the output. The results of these tests were compared with the expected results retrieved from the filter characteristics in appendix A. The measured responses were also normalized to their highest value, and the deviation from the expected attenuation was calculated. The results of this test can be seen in appendix D.1.

8.1.3 Spectrum Analysis

The final computer simulation of the design involved a comprehensive spectrum analysis of the Masterfilter. To achieve this, a custom spectrum analysis setup was designed. This setup consisted of a C++ script which generates N testbenches, where N depends on the resolution of the analysis. These testbenches fed the filter with a constant frequency sine signal for a long enough time for the filter to stabilize. The output of the filter was stored in a datafile, which could be analyzed in Matlab. To calculate the spectrum of the response, Matlab would run through each of the N simulation response files and perform an analysis on the data. A second version of the Masterfilter was created with probe outputs between each filter stage, meaning that each signal could be traced through the system. This way, it could be assured that it was the actual input frequency that propagated through the passband, and not a bi-product from the internal signal processing. The input of the system, and corresponding output of the filter can be seen in figures 8.1 and 8.2, respectively. The output of each intermediate stage of the filter system can be seen in appendix F.

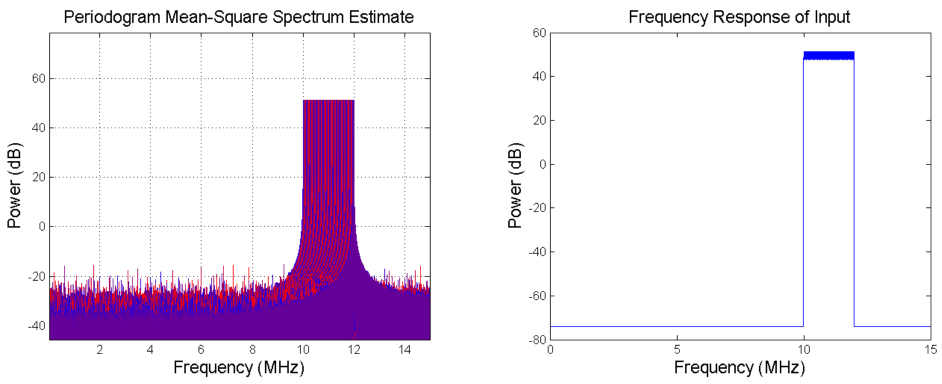


Figure 8.1: Simulation Input

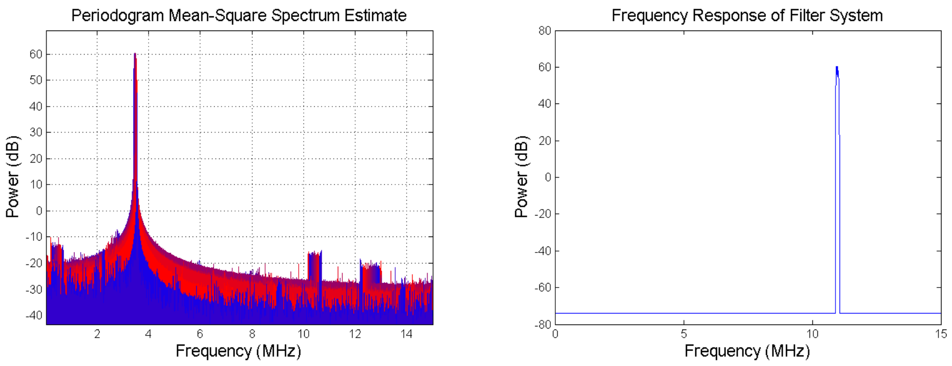


Figure 8.2: Simulation Output

The left side of the simulation results depicts the spectrum estimate of each signal at the given stage. Note that all the signal spectrum results are plotted on top of each other in the same figure, which means that this figure is mostly relevant for viewing frequency locations within the baseband, and not attenuation and noise floor. This is because a stronger signal in one position will always overlap a weaker signal in the same position, so only the strongest frequency components in all the combined tests are shown. However, color is added to separate each signal, which gives a decent idea of how the signal is affected in the frequency domain.

The right hand image shows the signal power of the given input frequency at the current stage. This value is retrieved by finding the power of the frequency component at which the signal is expected at the current stage, and storing it as the frequency power of this input stimuli. For example, in figure 8.1 the input test with frequency 11 MHz is expected to be found at 11 MHz, since no processing has occurred at this stage. On the other hand, in figure 8.2, the input test with frequency 11 MHz is expected to be found at $11\,\text{MHz} - 7.5\,\text{MHz} = 3.5\,\text{MHz}$. Therefore, in figure 8.2 the power of the frequency component at 3.5 MHz in the left hand figure is plotted at 11 MHz in the right hand figure. A closer look at the transition bands and the passband of the output can be seen in figure 8.3. Note that the frequencies shown in figure 8.3 also has a 7.5 MHz offset, where the real output will be centered at 3.48 MHz as previously explained.

This way of modeling the signal is interesting for two reasons. Firstly, one can observe the power of the signal at separate frequencies, which can indicate if the Scalingfactor of each filter is correct. The second reason is that one can see how the input signal is peeled off at each stage in the filter, and how the passband remains intact, as seen in appendix F.

All input tests which yielded a constant zero at the output, as well as frequencies outside the 10 MHz to 12 MHz range were modeled as -74 dB. The reason for this was that the maximum theoretical SNR for the ADC described in chapter 6 was determined to this value, according to equation 6.1.3. It should be noted that a ripple in the passband was detected during these simulations. The maximum ripple in the passband was measured to 2.84 dB.

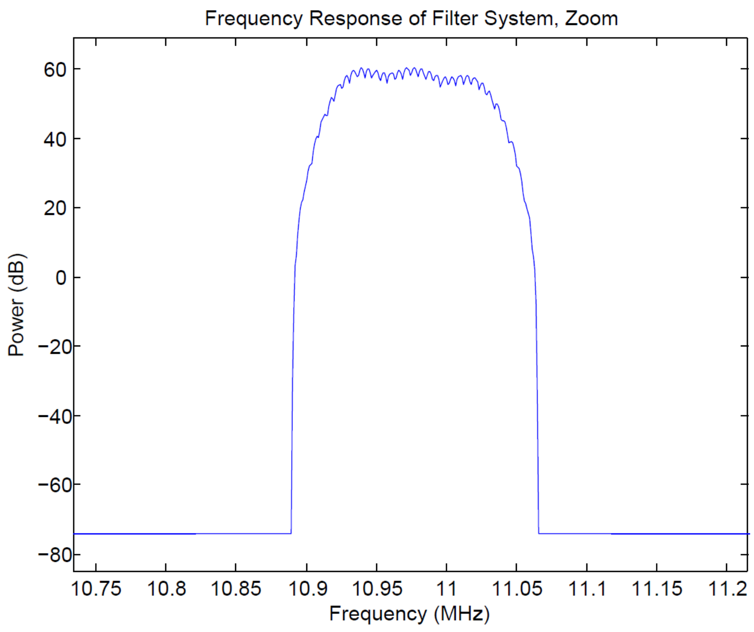


Figure 8.3: Simulation Output, 7.5 MHz Offset

8.2 FPGA-Testing

The final, and most realistic test conducted on the filters was testing in an actual FPGA. Although the design was already extensively tested with Modelsim simulations, there were several issues related to the physical implementation, such as timing and connectivity. The filters were implemented and tested on an *Altera Cyclone II EP2C70F896C6N* FPGA on an Altera DE2-70 development board. The goal of this test was to assure that the simulated behavior of the filter would translate to an actual system. Ideally, an Actel RTAX2000S would have been used, but this FPGA was unfortunately not available for this project.

The stimuli for these tests were generated within the FPGA itself, using the *numerically controlled oscillator* (NCO) intellectual property (IP) from Altera. The NCO was implemented using the CORDIC architecture described in section 3.2.1. The NCO was programmed to either be set to a constant frequency, or controlled by the switches of the Altera DE2 board. It should be noted that a stable 30 MHz clock was not available on the Alera DE2 board. Therefore, a 28.86 MHz crystal oscillator originally meant for the DE2-70 TV-decoder was routed as the system clock. The functionality of the filter should remain intact, but actual frequencies are shifted. For example, the input passband center frequency at 10.98 MHz with a sampling frequency of 30 MHz will now be placed at

$$f_c = 10.98\,\mathrm{MHz} \cdot \frac{28.86\,\mathrm{MHz}}{30\,\mathrm{MHz}} = 10.56\,\mathrm{MHz} \tag{8.2.1}$$

An illustration of the test setup can be seen in figure 8.4. Both the input and the output of the system were stored in the FPGA *random access memory* (RAM), and transmitted to the computer on request from the SignalTap II Interface.

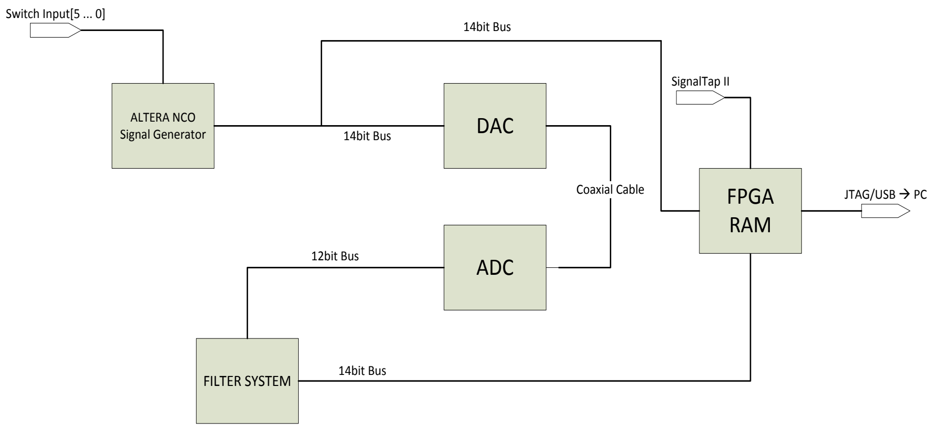


Figure 8.4: Testing Setup on FPGA

The first test on the FPGA was to assure the correct functionality of each individual filter, by sending tones with a constant frequency interval into the different regions of the filters, in the same manner as described in section 8.1.2. Responses were gathered from the SignalTap II interface, and processed with Matlab. The time used between each frequency was significantly greater than the filter delay calculated in section 6.6.1, and the length of the recorded results were maximized, only limited by the FPGA RAM size. This was done to reduce the error resulting from the lack of phase control when performing the frequency analysis on the result. As the Fourier transform is inherently continuous any discontinuities, even at the ends of the signal, will result in increased noise when transforming from the time domain to the frequency domain. This effect can be reduced by using window functions in Matlab, but some noise will still occur. A brute force solution for reducing this noise further is to use longer sequences of the signal, making the noise from the discontinuities less significant than the actual signal data. The results of the individual filter tests on the FPGA can be seen in appendix G. The expected results can be seen in appendix A. Note that both the decimation- and interpolation channels of the HdDec3RU- and HdDec4RU filters were tested separately. The frequency responses of both channels were expected to be equal.

The next test performed on the FPGA was testing of the complete Masterfilter. The VHDL code, along with the IP cores were compiled and the FPGA was programmed via the Quartus II software. The SignalTap II Logic Analyser was used to query the FPGA RAM to retrieve the stored signal. This signal was then stored to a datafile, which could be analyzed in Matlab. The Matlab-script for processing and plotting the datafiles can be found in the enclosed zip-file. A spectrum analysis of the results from the FPGA-testing can be seen in figure 8.5. In this figure, all the separate tests were plotted together, in the same manner as described in section 8.1.3. Aspreviously discussed, the issues created by phase discontinuities in the Fourier transform will also apply to the results in figure 8.5. This suggests that the attenuation would be lower than this figure displays.

In addition to figure 8.5, which gives an idea of the products generated by the filter system itself, a more accurate transfer function analysis showing the real attenuation of the

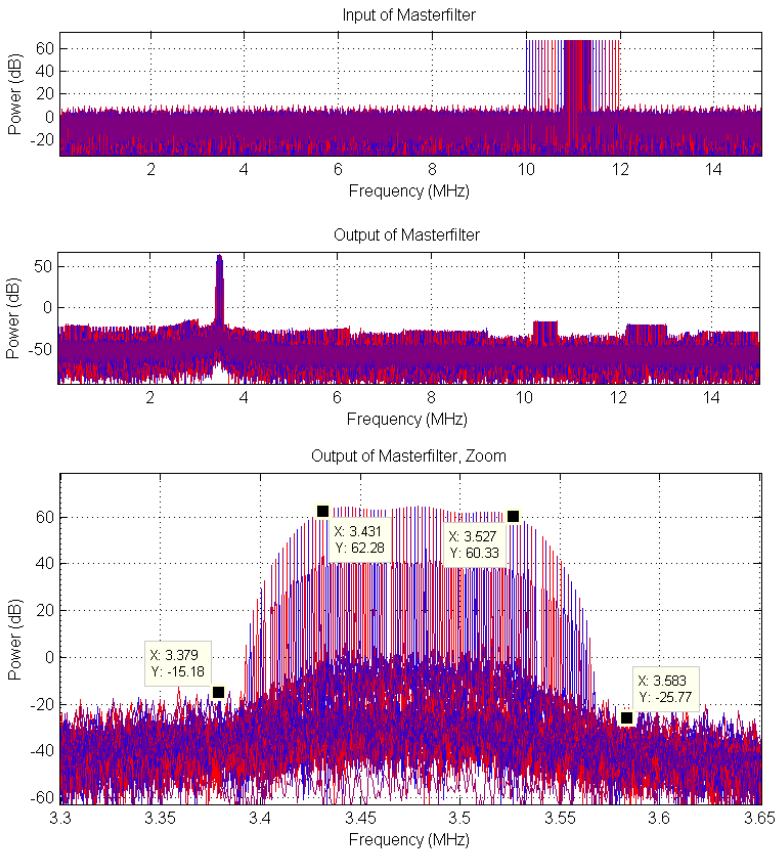
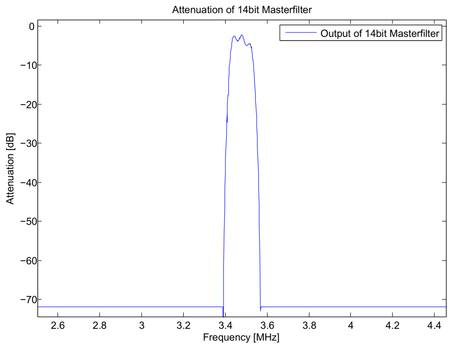


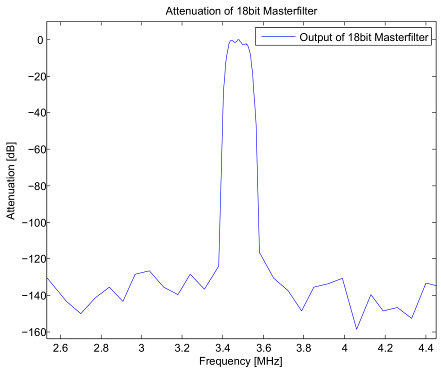
Figure 8.5: Results from the FPGA-Testing

Masterfilter was performed. Since the output of the filter is located 7.5 MHz lower than the input frequency, an offset would be needed to explore the attenuation of the filter. This was solved by recording the power of the tone at input frequency f_i and its given response at frequency $f_i - 7.5$ MHz, and calculating the attenuation. This attenuation can be seen in figure 8.6a. It should be noted that the attenuation of the filter in figure 8.6a resulted in a constant zero at frequencies outside the pass- and transition bands. To illustrate the attenuation, these values were modeled as the maximum SNR calculated in equation 6.1.3, -74 dB.

Since the Masterfilter uses a coefficient precision which allows a significantly higher attenuation at the internal bandpass filter, a new version of the Masterfilter with 18 bit internal bit width instead of 14 bit was created. This was done to determine what attenuation the filter could theoretically achieve when not limited by bit width. The same filter attenuation test was performed on this filter, and the normalized transfer function can be seen in figure 8.6b. It should be noted that in order to increase the precision to 18 bit,



(a) Attenuation of 14 bit Masterfilter



(b) Attenuation of 18 bit Masterfilter

Figure 8.6: Attenuation of Masterfilter

all internal filters needed to increase the size of their internal components as well. The precision of the coefficients were not affected by these changes. Note also that only the 2 MHz bandwidth from 10 MHz to 12 MHz was tested for the Masterfilter, as frequencies outside this bandwidth are assumed sufficiently attenuated.

Transition	14 bit Attenuation [dB]	18 bit Attenuation [dB]
Stop 1 - 3.38 MHz	$-\infty$	-124.14
Pass 1 - 3.43 MHz	-5.93	-4.62
Pass 2 - 3.53 MHz	-6.32	-3.48
Stop 2 - 3.58 MHz	$-\infty$	-115.84

Table 8.1: Effect of Increasing Precision

From the results illustrated in figure 8.6 the key attributes of the filter summarized in table 8.1 were determined. Note that for the 14 bit filter, the last recorded attenuation before the output was attenuated to a constant zero were -98.38 dB at 3.391 MHz and -72.69 dB at 3.56 MHz. When examining the passband of the filters, a small but significant ripple was detected. In the 14 bit filter this ripple was determined to 2.85 dB, while the 18 bit filter yielded a ripple of 2.79 dB.

The code used for the FPGA-testing can be found in the enclosed zip-file. Note that the code used for these tests did not apply the reusable filters, but rather the original filters for HdDec3 and HdDec4. The reason for this was that these filters raised issues with timing that the FPGA could not handle. Timing issues occurred sporadically after resets, suggesting that for these filters to work, all the clock domains needed to be strictly synchronized. This was not always the case with the Cyclone II FPGA. The simulations suggests that this could have been fixed with the use of an additional PLL, as discussed briefly in section 3.2.2. However, the internal PLL of the Cyclone II FPGA could only handle frequencies down to 9.38 MHz [23]. This could not satisfy the timing requirements of the HdDec4RU filter at the 5 MHz domain.

Chapter 9

Discussion

In the previous two chapters, the digital filter proposed in chapter 6 was synthesized, tested and verified with digital simulations and in an FPGA. The tests were targeted at the filters functionality, performance and area. Where the filter failed its requirements, optimizations were applied and new tests were performed. As a design decision, the output of the filter was located 7.5 MHz below the input. This was done to reduce the complexity of the 30 MHz clock domain of the filter, as explained in section 6.4. Since all the internal components in the Masterfilter have a linear phase, it is assumed that the Masterfilter itself also has a linear phase, as a result of the principle of superposition. An additive delay will occur as a result of the internal filters, but the delay will be constant for all frequencies in the passband. The response will also be inherently stable, since only FIR filters were used and no feedback occurs within the Masterfilter.

As seen in table 7.3 and figure 7.1, both the reusing of internal filters and the coefficient optimizations provided a significant improvement in the area estimations of the filter. The excess area generated from these optimizations were used as a trade off for more performance, as shown by the pipelining scheme implemented in section 7.2.3. This pipelining optimization resulted in a degradation of performance in the slower clock domains, as more logic was implemented. However, the critical clock domain at 30 MHz, achieved a 9.7% increase in performance, which suggested that the filter would now comply with the specifications.

In section 8.1.1 the implementations of the internal filters were tested using impulse responses, and the output can be seen in appendix B. When these results were studied, they proved to match the expected results, seen in appendix A. These results gave a good indication that the filters were implemented correctly in VHDL, according to the filters designed in chapter 6. However, these tests only took one pulse as input, so other issues with for example step responses, timing and overflow could not be determined.

The tone testing described in section 8.1.2, with corresponding results in appendix D.1 proved the general characteristics of the individual filters. For the filters with the least precision and complexity, the simulation proved quite close to the expected results. On the other hand, in the more complex filters with high attenuation such as the HdBand-filter, there were significant deviation in the measured attenuation compared with the expected results. One possible explanation for this might originate in the precision of the filter output. As seen in table D.3, the 169 kHz tone was expected to yield an attenuation of

−115 dB, but the recorded attenuation was only −69.5 dB. A possible explanation for this is that the −115 dB attenuation assumes an output bit width of $W_{coefficients} + W_{input} = 33$ bit. However, the output bit width is only 14 bit, which will result in a significantly reduced attenuation.

In section 8.1.3, a spectrum analysis was performed to determine the transfer function of the Masterfilter. This simulation was conducted on the VHDL code in the Modelsim environment. At the interconnecting points between each internal component in the Masterfilter, a figure showing the frequency response of the recorded signal was created, as described in detail in section 8.1.3. When studying the analysis results in appendix F, one can observe how each filter stage removes certain components from the original input. One can also observe how signal mixing and altering of the sampling frequency affects frequency components in the baseband. From the results of the computer simulated spectrum analysis seen in figure 8.2, it is apparent that all input frequencies outside the filter passband and transition bands will be completely attenuated. However, as explained in section 8.1.3, these simulations were performed in ideal conditions in Modelsim which means that noise from other sources such as quantization and timing issues were not accounted for.

When testing the filter in the Altera Cyclone II FPGA as described in section 8.2, a few issues arise. First off, the timing which functioned during the digital simulations only worked partially in the FPGA. The optimized reusable filters introduced in section 7.2.1 required a very stable clock signal synchronized with the preceding filters, which was not always the case in the FPGA. During some tests, the filters worked sporadically depending on when the circuit was reset, indicating a timing related problem, as discussed in section 3.2.2. The timing issues did not affect the system when using the less sensitive HdDec3- and HdDec4 filters instead of the reusable filters. Since area was not critical in the Cyclone II FPGA, these filters were implemented.

The results from the FPGA-testing, shown in figure 8.5, suggests a perfect attenuation of input frequencies less than 50kHz above and below the passband. The passband itself is about 100 kHz, which was required by the filter specifications. One can also observe the frequency placement of the filter output, 7.5 MHz below the input frequency as intended by the design in section 6.

When testing the Masterfilter attenuation outside the passband and transition bands, a constant zero was recorded at the output. This was expected, since the internal attenuation of the filter was designed to be greater than the 14 bit filter output could handle, as suggested in section 8.2. In figure 8.6a one can observe this effect from the Masterfilter transfer function. In figure 8.6b, one can see the actual attenuation of the filter, where the internal signal precision of the filters are extended to 18 bit. The number of stages and the coefficient precision in the internal filters are not altered in the 18 bit Masterfilter. These results suggests that if the precision of the filter is high enough, one can get up to well over 120 dB attenuation in the stopband, which exceeds the requirements stated in the problem description. However, this increase in precision is expected to cause a drastic increase in the area usage, due to the use of larger multipliers.

As shown in sections 8.1.3 and 8.2, there is a notable ripple in the passband of the Masterfilter. This ripple is insignificant compared to the attenuation of the filter, but one should be aware of it, as it amounts to almost 3 dB in some regions. The ripple occurs due to the fact that the passband of the internal filters in the Masterfilter are not completely flat. This ripple, which seems insignificant at each filter stage, will sum up to a more significant ripple. However, most of the ripple originates in the internal HdBand filter, which has a passband ripple of 1.8 dB. If a filter with less ripple is desired, this HdBand could be optimized. As previously shown, the attenuation of the proposed HdBand filter is not fully utilized in the 14 bit Masterfilter, suggesting excess attenuation could be traded for reduced passband ripple. Comparing the proposed Masterfilter with the Masterfilter generated directly in Matlab described in chapter 6, the Masterfilter proposed in this thesis proved approximately 1 dB higher passband ripple. Specific requirements of the passband ripple, apart from the current filter transfer function shown in figure 4.1, were not specified in the problem description in section 1.2.

It is important to note that the FPGA-tests described in section 8.2 were performed using an Altera Cyclone II FPGA rather than a space qualified FPGA, such as the Actel RTAX2000S. This means that the results from these tests might not be representable for other FPGA architectures. On the other hand, the filter code is designed to be very portable. No vendor specific logic was used in the computer simulated code, which means that it should work more or less without any customization, given that the FPGA can handle the filter requirements. However, using a non vendor specific clocking scheme caused problems with timing in the Cyclone II FPGA. It should be noted that an Altera specific PLL was used to synchronize the FPGA and the ADC/DAC interface. This should not affect the filter behavior.

One design feature of the filter proposed in chapter 6 which was not tested in this thesis, was the undersampling process. The real input of the filter system should in fact be centered at 70.98 MHz and undersampled down to 10.98 MHz, as described in section 6.1. This phenomena is well documented, and it was assumed that this would not need any further testing.

A convenient implication of the modular multistage FIR structure used in the Masterfilter is that if one wishes to use the filter for a wider or more narrow passband, it would be sufficient to only replace the HdBand filter. This would be much simpler than designing a completely new filter from scratch, given that the new passband is within the limitations of the current HdBand filter. No limitations would apply for how narrow the passband could be, but the width would be limited by the sampling frequency of the HdBand filter, 1.25 MHz.

Since this filter was designed to be as general as possible, it should be customized to fit specific FPGAs with given input and output precision. For example, if the output of the FPGA was to be connected to an 8 bit DAC, the excess attenuation in the filter which would not be used could be traded for area, performance or a more smooth passband with reduced ripples. The issues related to the digital to analog conversion after the signal filtering were not discussed in detail in this thesis. However, a suggested method of reaching the original 70.98 MHz input was mentioned, where an analog mixer and filter was applied.

Chapter 10

Conclusions

In this thesis, a digital bandpass filter with input sampling frequency 30 MHz, a bandwidth of 100 kHz and transition bands of less than 50 kHz was designed. The input signal of the filter, which had an initial bandwidth of 1.8 MHz centered at 70.98 MHz was designed to be downsampled to a baseband of 15 MHz, and centered at 10.98 MHz. The output of the filter was chosen to be centered at 3.48 MHz, with a sampling frequency of 30 MHz. The filter was designed as a multistage decimation and interpolation filter with clock domains reaching from 30 MHz down to 1.25 MHz. The filter, called Masterfilter, was designed and simulated with Matlab, and implemented using VHDL. To fit the Mastefilter into a space qualified FPGA, performance- and spacial optimization techniques such as pipelining and resource sharing were applied. Additional support tools for generating VHDL FIR filters and testbenches were created using C++. The Masterfilter was first tested using Modelsim with input generated in Matlab, and then tested in an actual FPGA, using a built in NCO signal generator.

The following structural specifications were determined after the filter was designed:

- The Masterfilter will require 95% of the combinatorial cells in the Actel RTAX2000S FPGA. Additionally, the Masterfilter would also fit into the Actel RTAX4000S and Xilinx Virtex 4-QV FPGAs, both with and without the timing intensive resource sharing optimization.
- The Masterfilter will fulfill the performance specifications in the Actel RTAX2000S FPGA. The critical clock domain in the Masterfilter, the 30 MHz domain, achieved an estimated performance of 32.8 MHz.
- The delay of the Masterfilter was determined to $55.7\,\mu$s.
- The Masterfilter was determined to have a stable linear phase response, as a result of exclusive use of internal FIR filters with linear phase, without any feedback.

The following results can be summarized from the testing of the Masterfilter on an Altera Cyclone II FPGA:

- With an internal and external bit width of 14 bit, the attenuation at the output exceeded the precision of the filter output. This resulted in a constant zero at the output, suggesting an infinite attenuation. However, the last recorded attenuation in the passband was −72.69 dB at 30kHz above the passband, and −98.38 dB at 39 kHz below the passband. The attenuation in the passband was approximately 6 dB compared to the input signal.

- The maximum actual attenuation of the Masterfilter was determined by extending the internal and external bit precision to 18 bit. This resulted in an attenuation of −124.14 dB 50 kHz below the passband, and an attenuation of −115.84 dB was recorded 50 kHz above the passband.
- The undesired frequency components at 670 MHz and 741 MHz were sampled down and attenuated by both the analog bandwidth of the ADC and in the intermediate filters of the Masterfilter.

These estimations and results suggests that the proposed filter would satisfy both the performance- and area requirements for the Actel RTAX2000S, given that the resource sharing optimization is applied. However, this filter would require a good clock source, which can generate stable clock signals for all the clock domains, as well as inversed clocks. If larger FPGAs are used, such as the RTAX4000S or the Virtex 4QV, the Masterfilter using the original filters without resource sharing would be more than sufficient, leaving extra room in the FPGA for additional functionality. The signal conversion from the digital domain to the analog domain after the FPGA filtering, and mixing up to the original center frequency was not discussed in detail in this thesis.

Chapter 11

Further Work

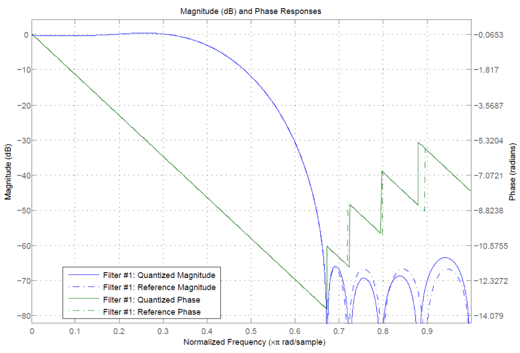
Since this thesis was limited to one semester, there was only time for a limited number of tests and design optimizations. One potential extension of the digital filter would be internal testing of the behavior and structure within the FPGA. In section 8.2 a NCO signal generator was implemented inside the Altera Cyclone II FPGA. This setup could be extended to also include an FFT-unit, which could process the output of the filter. This way, the filter response could be determined very rapidly, and the overflow characteristics could be explored over long sequences of input.

If the filter is to be optimized further, it could be possible to reuse even more components in the two HdDec3RU filters. These filters could be combined into one filter with four separate delay lines using shared arithmetic units. This would require twice the clock frequency for the arithmetic, but it would save roughly half the area it takes to implement the existing arithmetic.

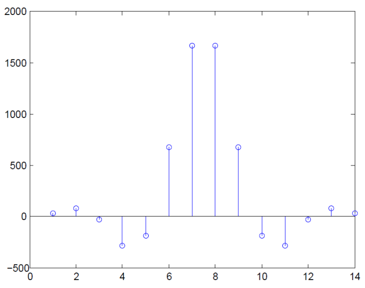
In this thesis, it was assumed that the phase linearity was certain as a result of exclusively linear internal components and the superposition principle, as discussed in chapter 9. However, this feature could also have been measured in the FPGA as it was a strict requirement of the filter.

Appendix A

Filter Frequency and Phase Responses

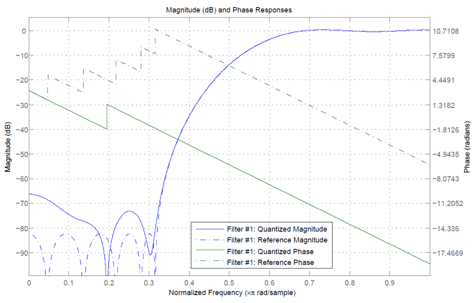


(a) Frequency and Phase Response

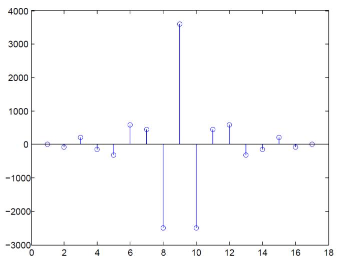


(b) Coefficients

Figure A.1: Theoretical Response of Coverfilter

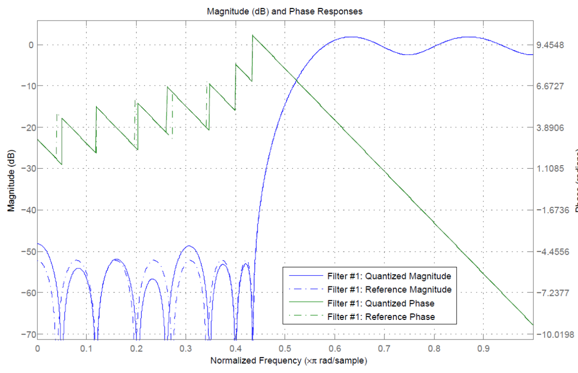


(a) Frequency and Phase Response

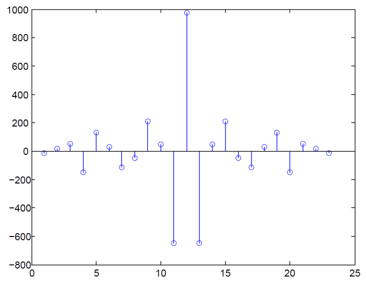


(b) Coefficients

Figure A.2: Theoretical Response of HdDec2

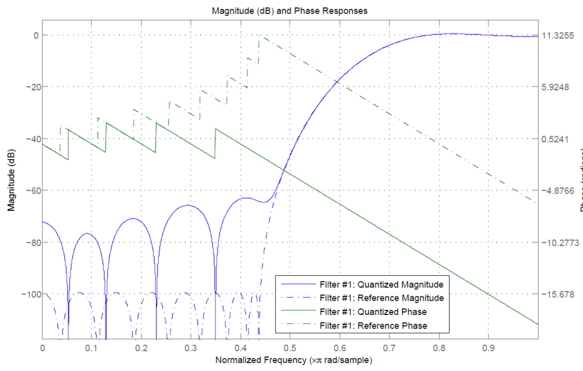


(a) Frequency and Phase Response

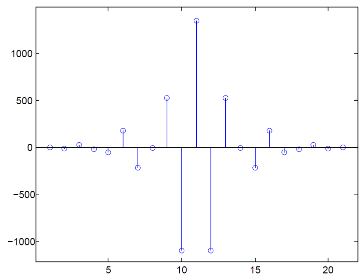


(b) Coefficients

Figure A.3: Theoretical Response of HdDec3

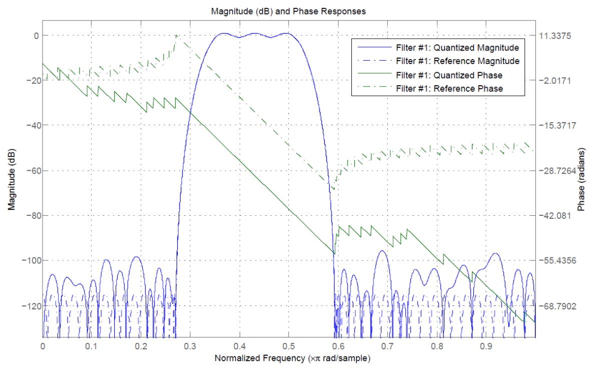


(a) Frequency and Phase Response

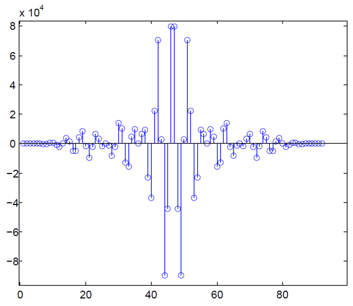


(b) Coefficients

Figure A.4: Theoretical Response of HdDec4

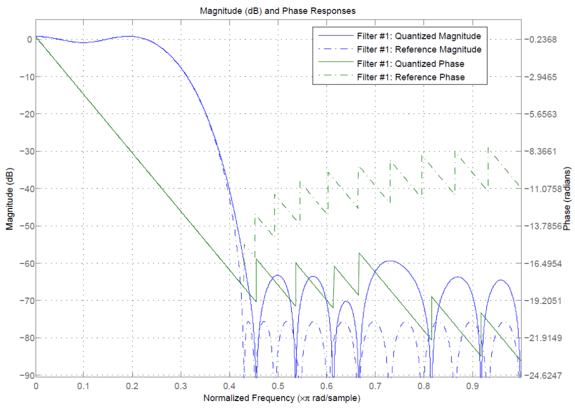


(a) Frequency and Phase Response

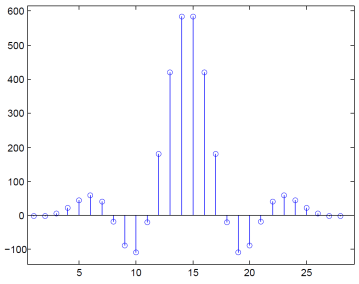


(b) Coefficients

Figure A.5: Theoretical Response of HdBand

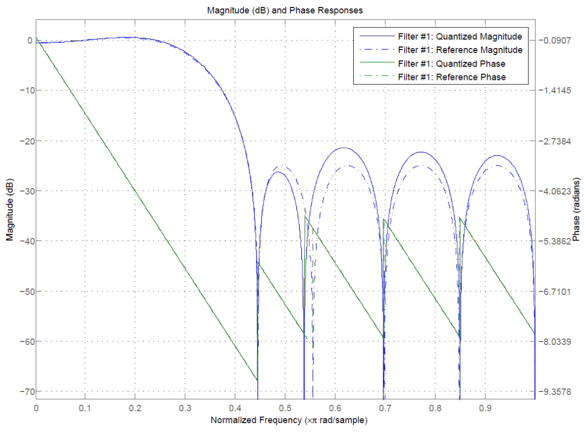


(a) Frequency and Phase Response

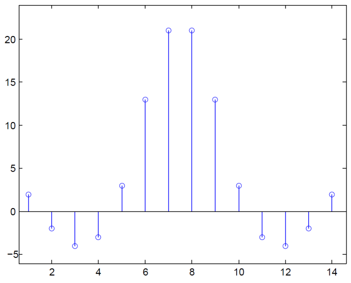


(b) Coefficients

Figure A.6: Theoretical Response of HdOut



(a) Frequency and Phase Response

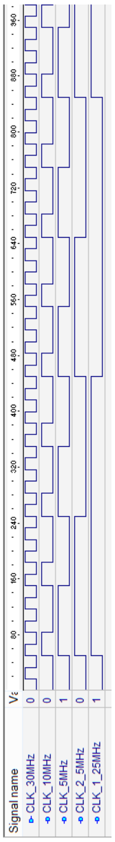


(b) Coefficients

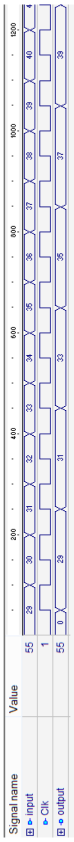
Figure A.7: Theoretical Response of HdOut2

Appendix B

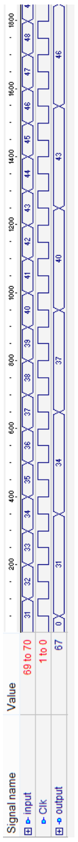
VHDL Simulation



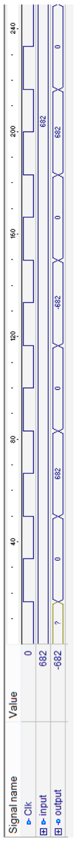
(a) Clock Generator



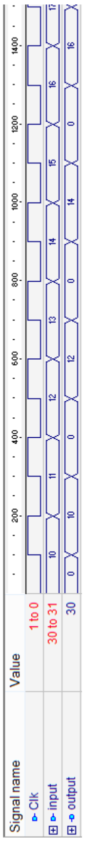
(b) Down-sampler x2



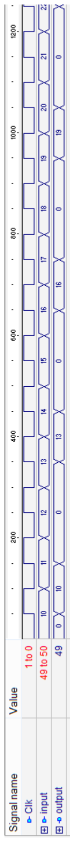
(c) Down-sampler x3



(d) Mixer

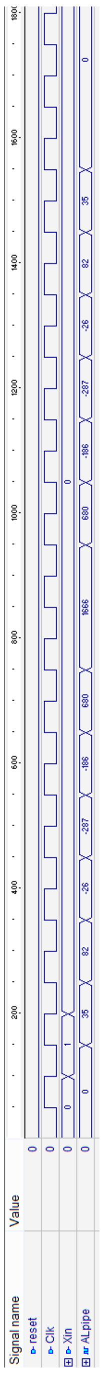


(e) Up-sampler x2

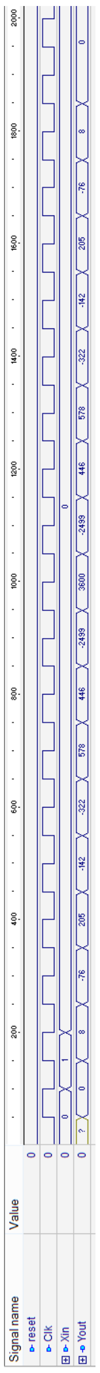


(f) Up-sampler x3

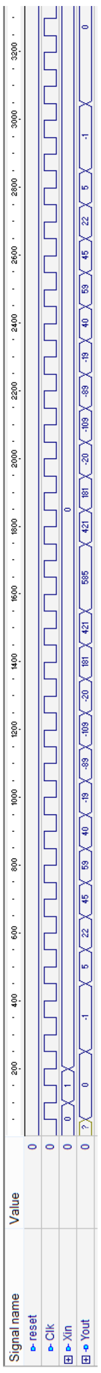
Figure B.1: Waveform Simulations of Processing Units



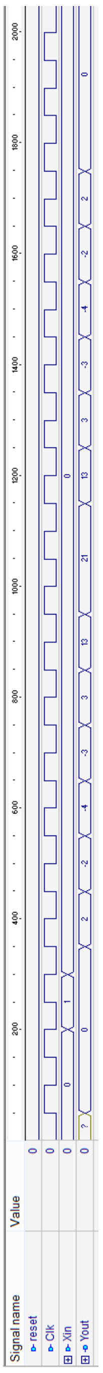
(a) Cover-filter



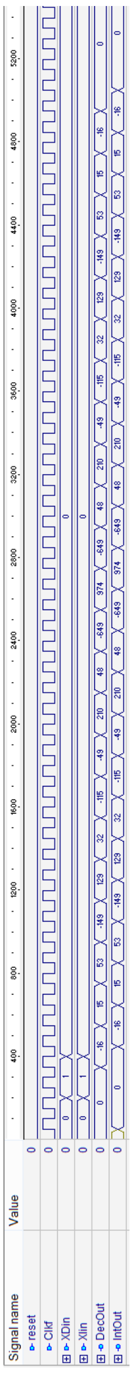
(b) HdDec2



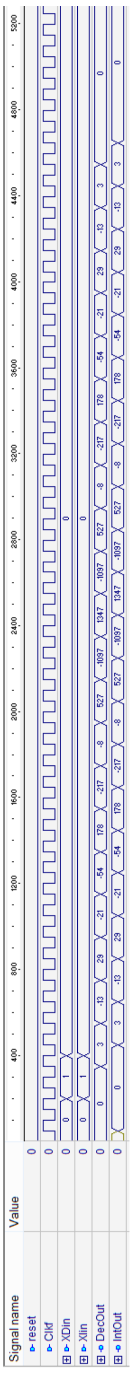
(c) HdOut



(d) HdOut2



(e) HdDec3RU



(f) HdDec4RU

Figure B.2: Waveform-Simulations of Internal Filters in the FPGA

Appendix C

Filter Optimization

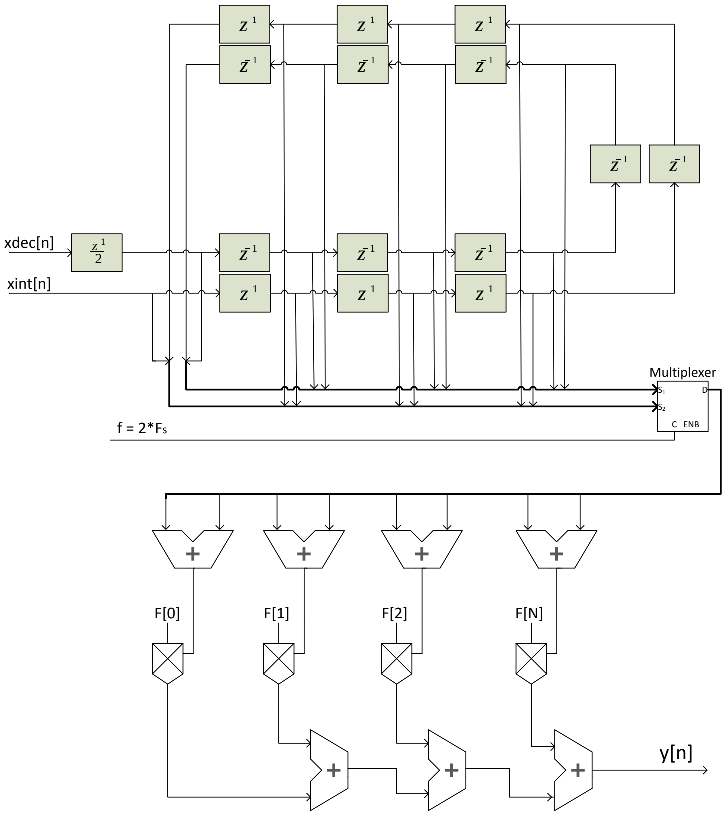


Figure C.1: Reusable Decimation- and Interpolation Filter

Appendix D

Testing of the Filters

D.1 Testing With Tones

F[MHz]	2	4	9	10	13
Expected attenuation [dB]	0	0	-30	-67	-74
Real attenuation [dB]	9	13	-19.4	-54.7	-64.1
Normalized attenuation [dB]	-4	0	-32.4	-67.7	-77.1
Deviation [dB]	4	0	2.4	0.7	3.1

Table D.1: Tone-Test of Coverfilter

F[MHz]	0.5	1	1.5	3.5	4.5
Expected attenuation [dB]	-74	-91	-90	0.18	-0.2
Real attenuation [dB]	-68.4	-72.5	-67.8	-6.38	-5.11
Normalized attenuation [dB]	-63.3	-67.4	-62.7	-1.27	0
Deviation [dB]	10.7	23.6	27.3	1.09	0.2

Table D.2: Tone-Test of HdDec2

F[kHz]	46	169	228	308	371	534
Expected attenuation [dB]	-110	-115	0.8	0.85	-114	-102
Real attenuation [dB]	-72.4	-67	2.5	1	-70	-76
Normalized attenuation [dB]	-74.9	-69.5	0	-1.5	-72.5	-78.5
Deviation [dB]	35.1	45.5	0.8	2.35	41.5	23.5

Table D.3: Tone-Test of HdBand

F[MHz]	3	6	8	11	14
Expected attenuation [dB]	0.74	-40	-90	-60	-70
Real attenuation [dB]	6.61	-35.02	-68.4	-52.8	-64.5
Normalized attenuation [dB]	0	-41.63	-75	-59.4	-71.1
Deviation [dB]	0.74	1.63	15	0.6	1.1

Table D.4: Tone-Test of HdOut

F[MHz]	3	6	8	11	14
Expected attenuation [dB]	0.5	-15.5	-45	-25	-23.2
Real attenuation [dB]	-16.2	-34.4	-76.6	-36.9	-44.4
Normalized attenuation [dB]	0	-18.2	-60.4	-20.7	-28.2
Deviation [dB]	0.5	2.7	15.4	4.3	5

Table D.5: Tone-Test of HdOut2

F[MHz]	0.5	1	1.5	2
Expected attenuation [dB]	-70	-70	1	-0.4
Real attenuation, dec[dB]	-51.6	-45.2	-4.6	-6.5
Real attenuation, int[dB]	-64.7	-60.6	-4.5	-6.4
Normalized att., dec[dB]	-47	-40.6	0	-1.9
Normalized att., int[dB]	-60.2	-56.1	0	-1.9
Deviation, dec [dB]	23	29.4	1	1.9
Deviation, int [dB]	9.8	13.9	1	1.9

Table D.6: Tone-Test of HdDec3RU

F[MHz]	0.2	0.4	0.6	1
Expected attenuation [dB]	-73	-68	-55	0.4
Real attenuation, dec[dB]	-59.3	-51.7	-43.8	5.8
Real attenuation, int[dB]	-59.4	-56.5	-48	5.8
Normalized att., dec[dB]	-65.1	-57.5	-49.6	0
Normalized att., int[dB]	-65.2	-62.3	-53.8	0
Deviation, dec [dB]	7.9	10.5	5.4	0.4
Deviation, int [dB]	7.8	5.7	1.2	0.4

Table D.7: Tone-Test of HdDec4RU

Appendix E

VHDL Code

E.1 10 MHz Clock Generator Code

```
-- 10MHz clock generator
clk_proc:process (CLK_30MHz)
variable poscount : unsigned(2 downto 0) := "000";
variable negcount : unsigned(2 downto 0) := "000";
begin
  if (rising_edge(CLK_30MHz)) then
    if (poscount = "000") then
      tmp10 <= not(tmp10);
      poscount := poscount+1;
    else
      if poscount = "010" then
        poscount := "000";
      else
        poscount := poscount+1;
      end if;
    end if;
  end if;

  if (falling_edge(CLK_30MHz)) then
    if (negcount = "001") then
      tmp10 <= not(tmp10);
      negcount := negcount+1;
    else
      if negcount = "010" then
        negcount := "000";
      else
        negcount := negcount+1;
      end if;
    end if;
  end if;
end process clk_proc;
```

Appendix F

Hardware Simulations

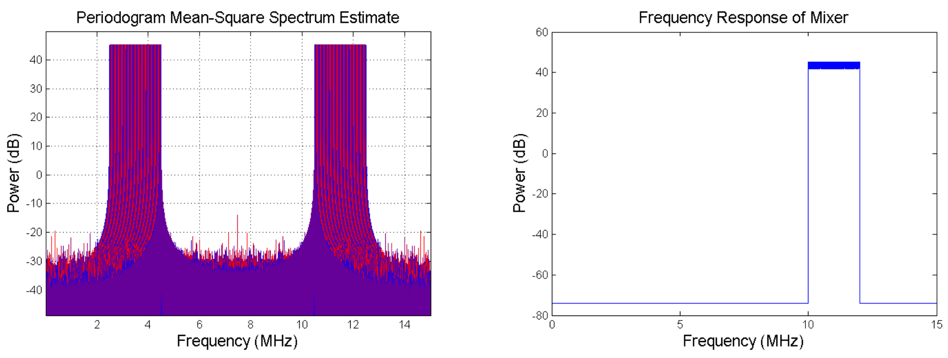


Figure F.1: Mixer Output

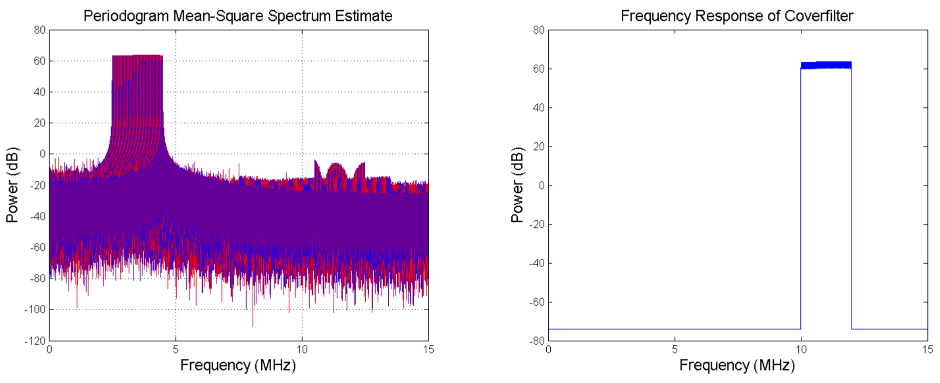


Figure F.2: Coverfilter Output

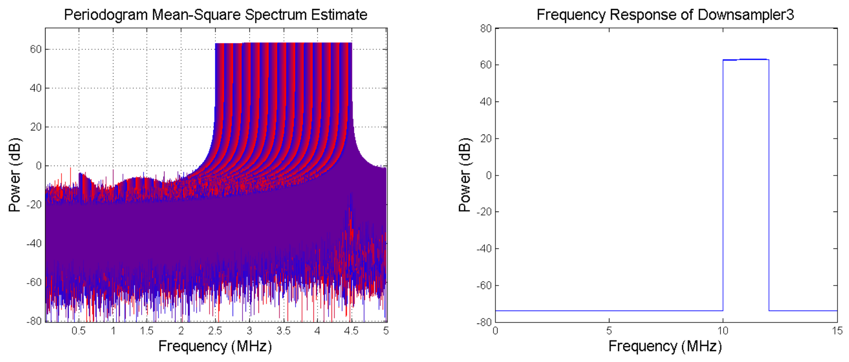


Figure F.3: Downsampler3 Output

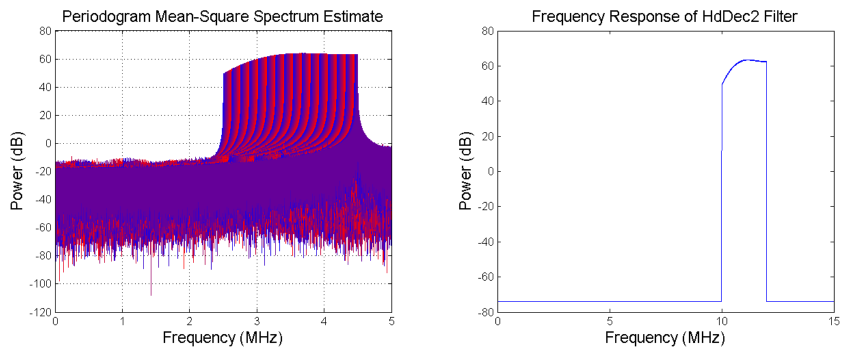


Figure F.4: HdDec2 Decimation Output

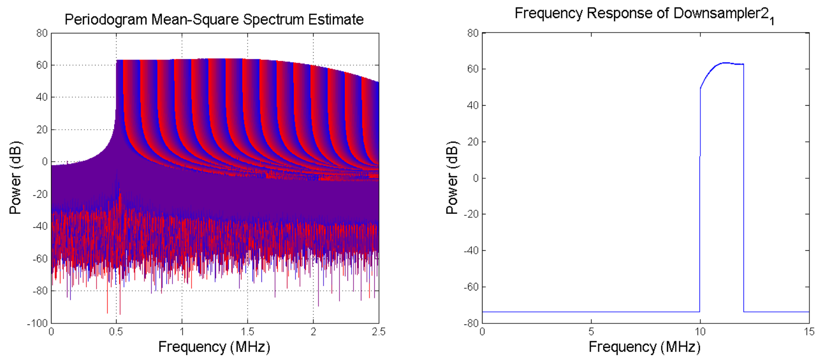


Figure F.5: Downsampler2_1 Output

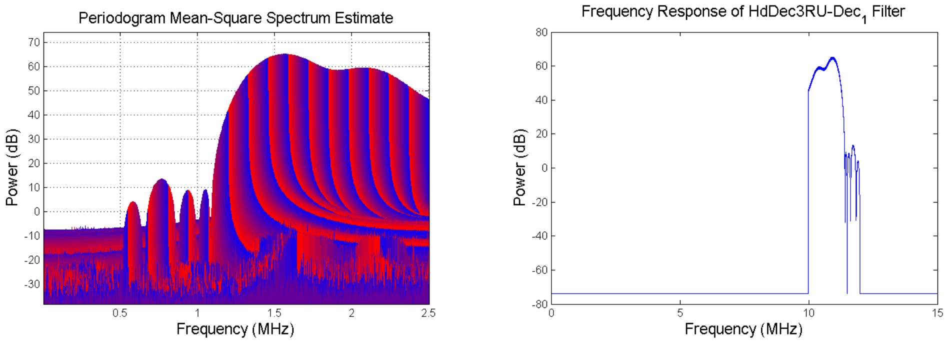


Figure F.6: HdDec3RU Decimation Output, Stage 1

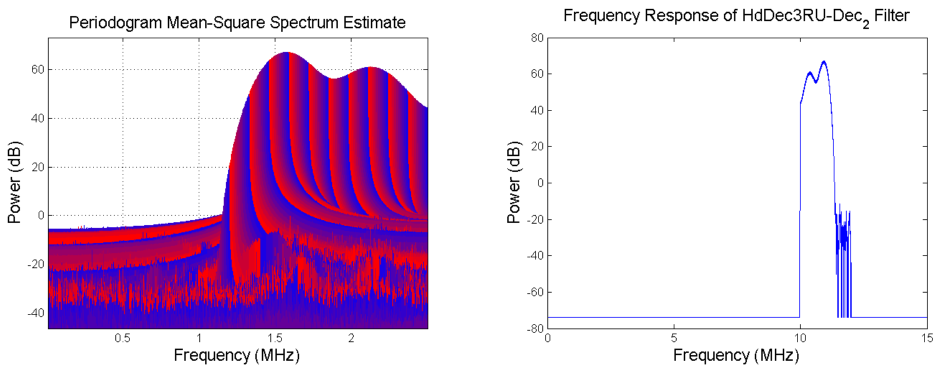


Figure F.7: HdDec3RU Decimation Output, Stage 2

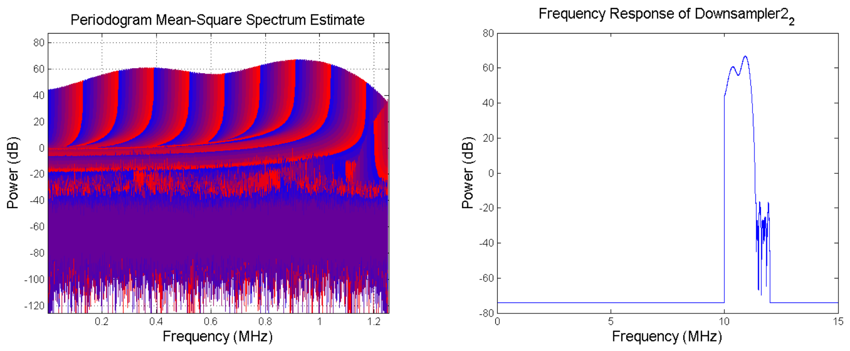


Figure F.8: Downsampler2_2 Output

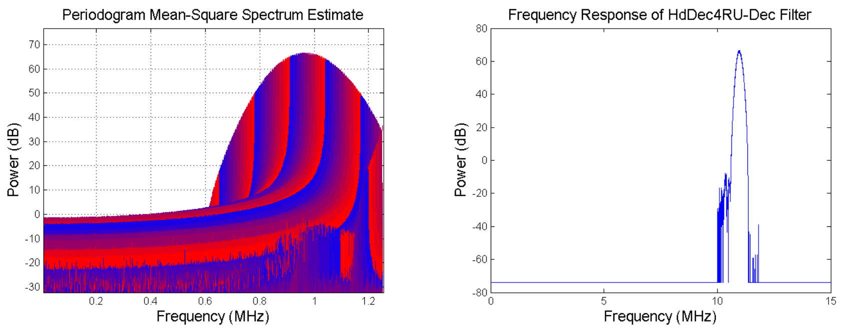


Figure F.9: HdDec4RU Decimation Output

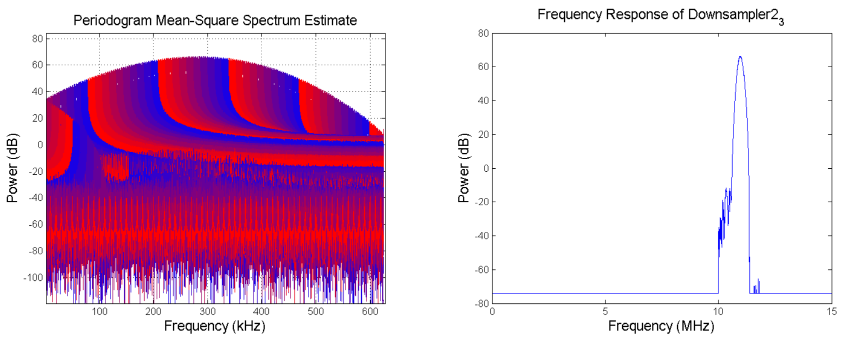


Figure F.10: Downsampler2_3 Output

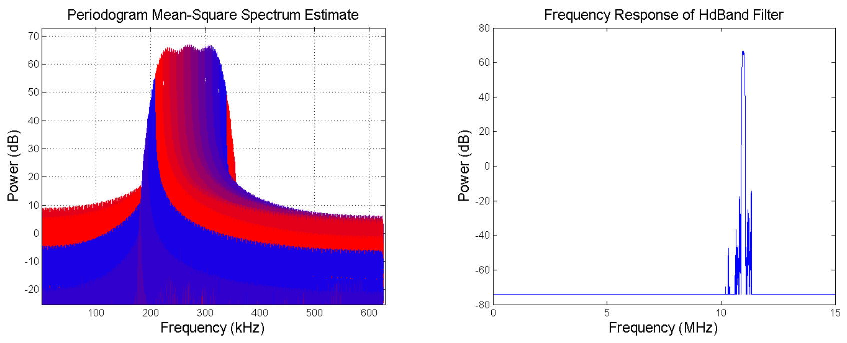


Figure F.11: Bandpassfilter Output

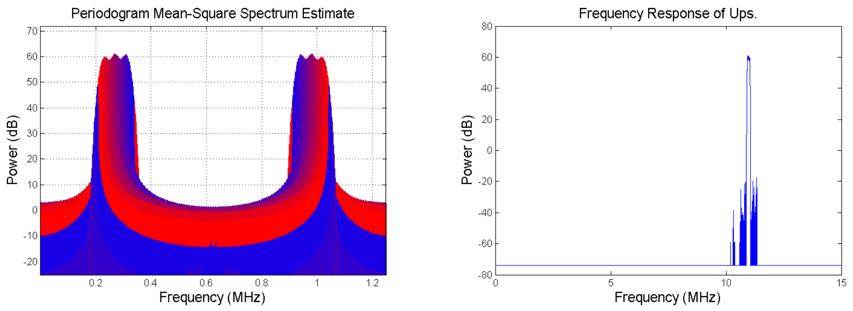


Figure F.12: Upsampler21 Output

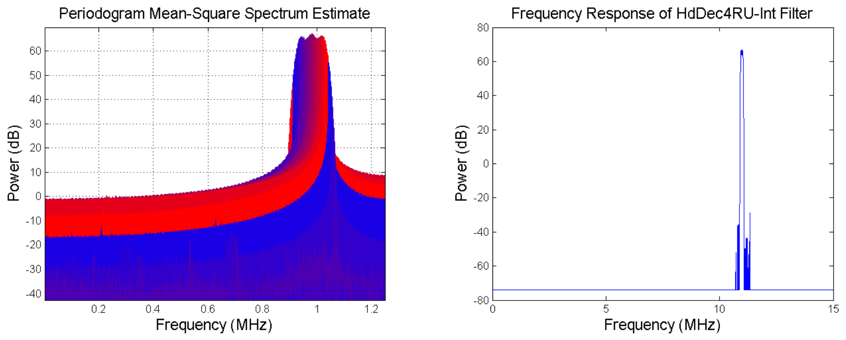


Figure F.13: HdDec4RU Interpolation Output

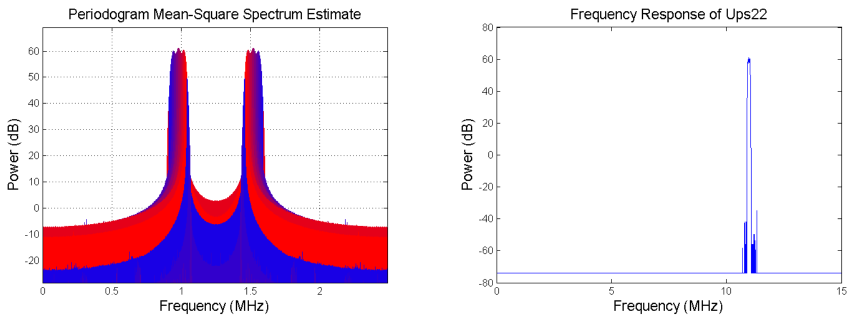


Figure F.14: Upsampler22 Output

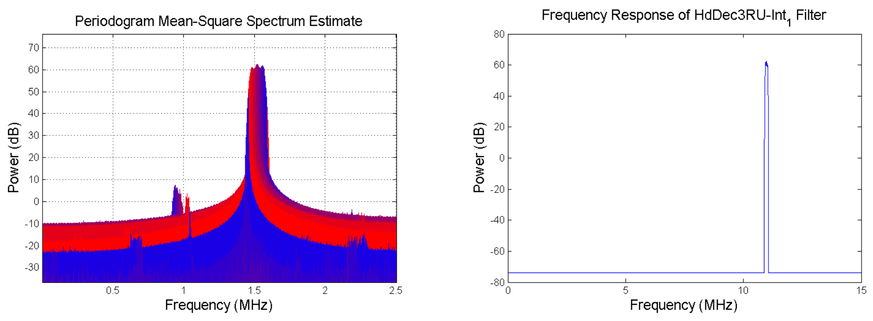


Figure F.15: HdDec3RU Interpolation Output, Stage 1

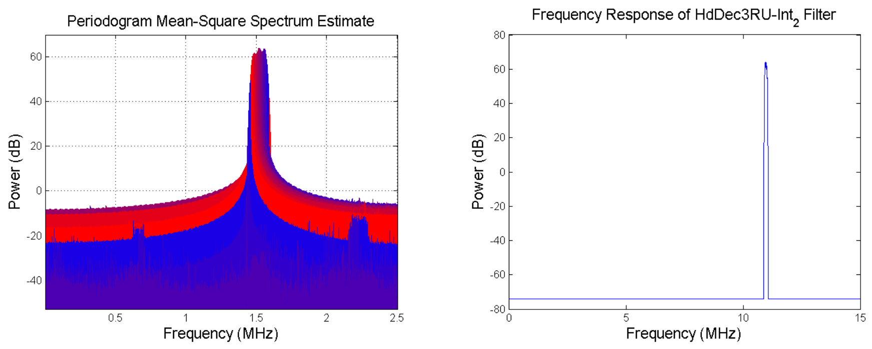


Figure F.16: HdDec3RU Interpolation Output, Stage 2

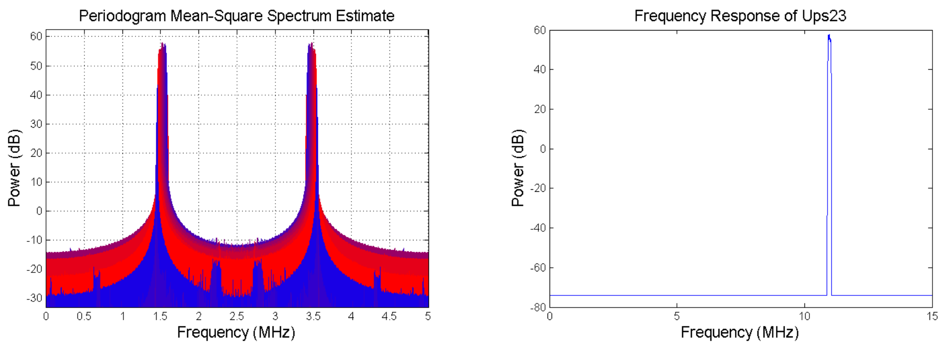


Figure F.17: Upsampler23 Output

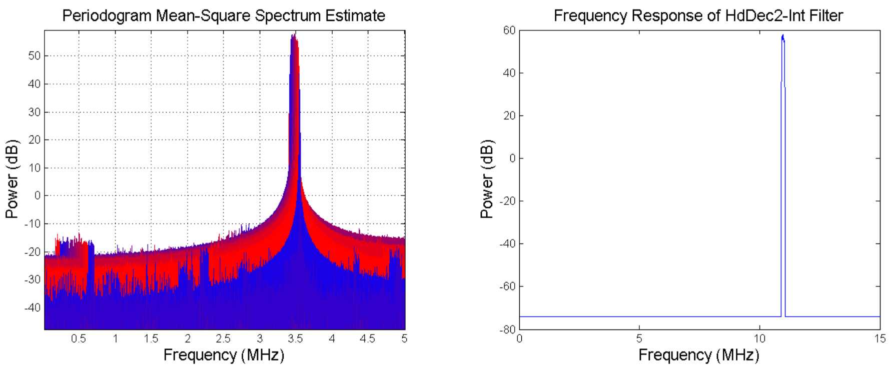


Figure F.18: HdDec2 Interpolation Output

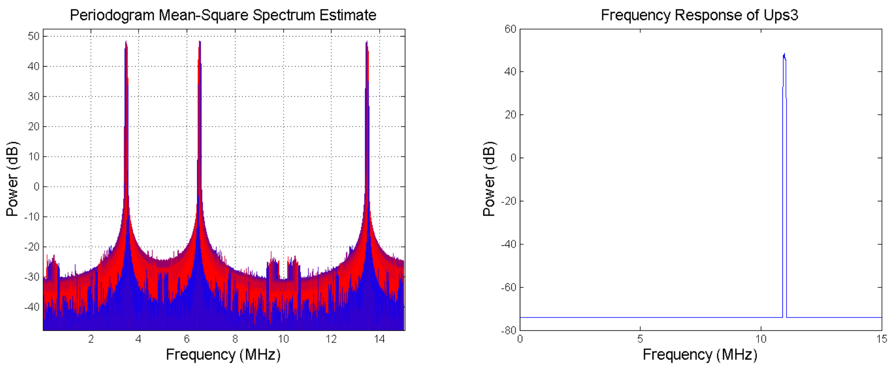


Figure F.19: Upsampler3 Output

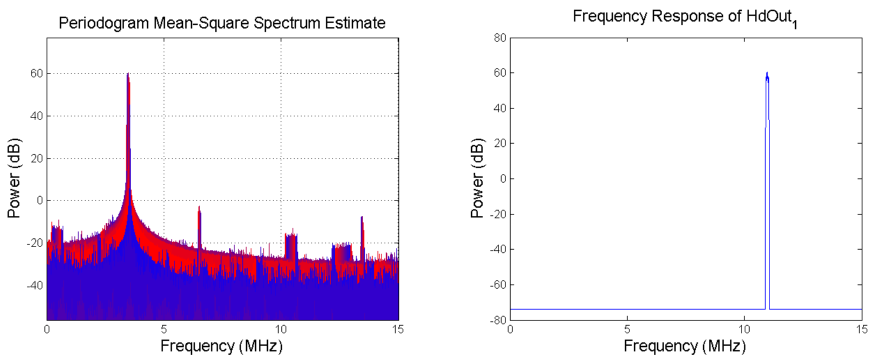


Figure F.20: HdOut Output

Appendix G

Filter FPGA Tests

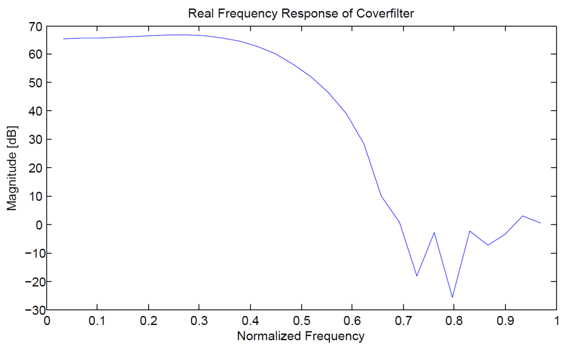


Figure G.1: Hardware Coverfilter Response

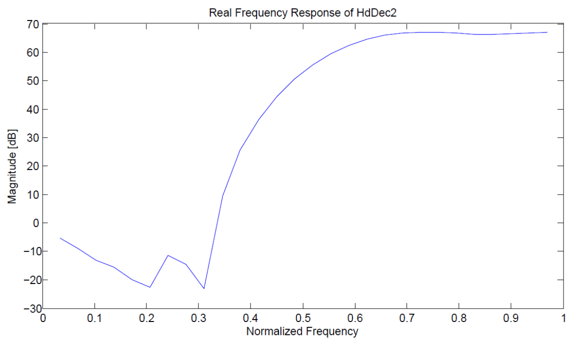


Figure G.2: Hardware HdDec2 Response

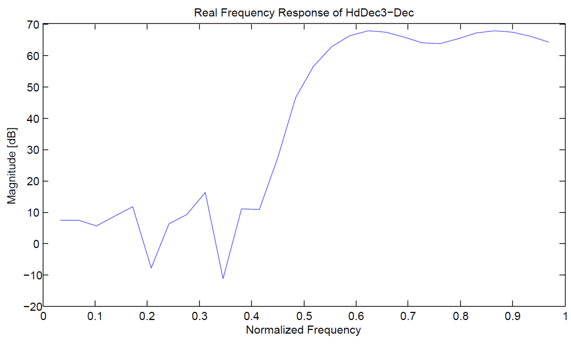


Figure G.3: Hardware HdDec3RU Response, Decimation Channel

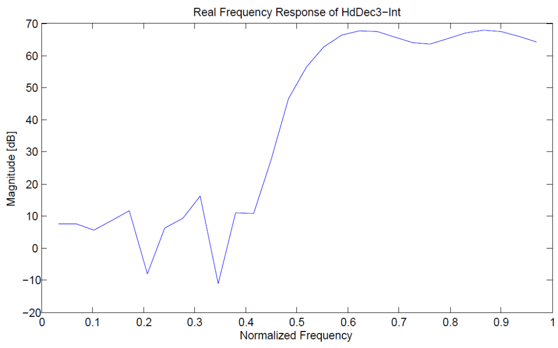


Figure G.4: Hardware HdDec3RU Response, Interpolation Channel

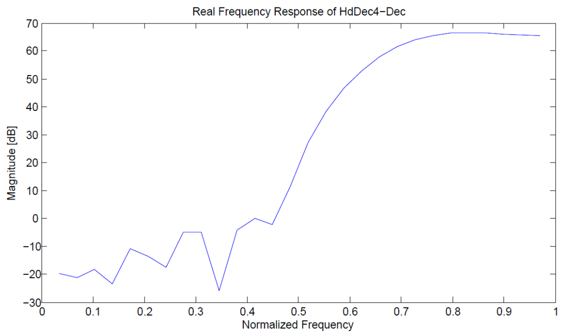


Figure G.5: Hardware HdDec4RU Response, Decimation Channel

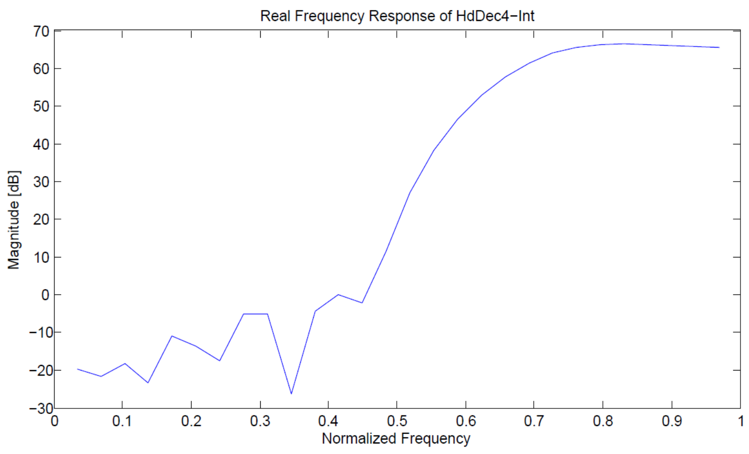


Figure G.6: Hardware HdDec4RU Response, Interpolation Channel

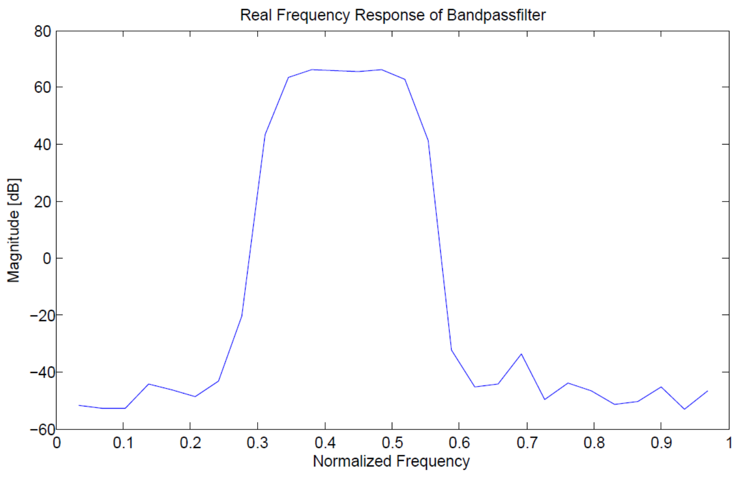


Figure G.7: Hardware Bandpassfilter Response

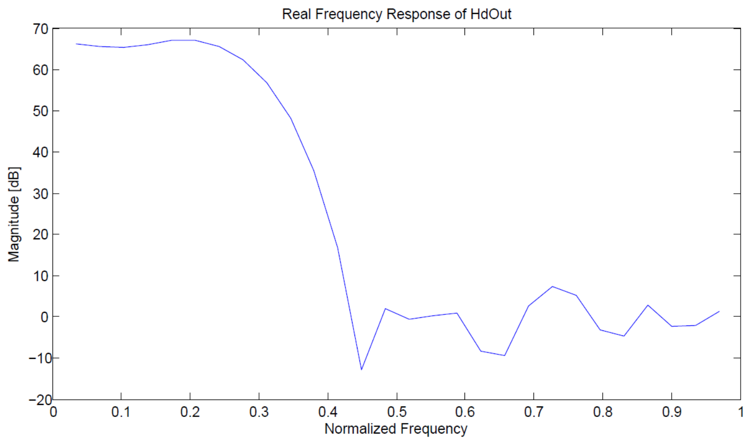


Figure G.8: Hardware HdOut Response

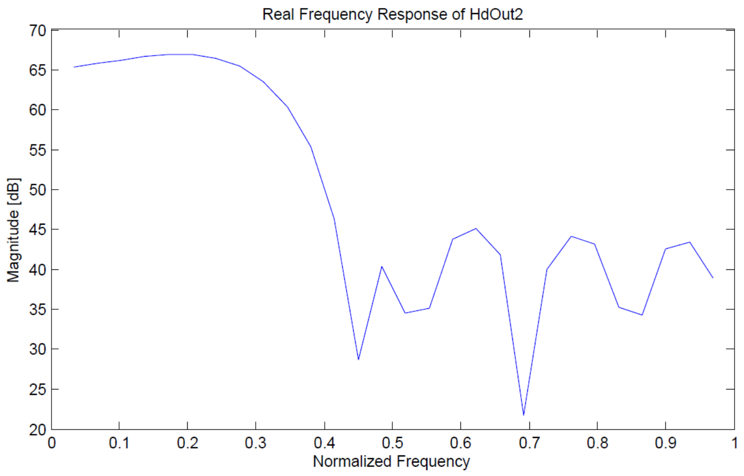


Figure G.9: Hardware HdOut2 Response

Bibliography

[1] Proakis JG, Manolakis DG. Digital Signal Processing: Principles, Algorithms and Applications. Fourth ed. New Jersey: Pearson Education, Inc.; 2007.

[2] Kester W. Taking the Mystery out of the Infamous Formula, "SNR = 6.02N + 1.76dB," and Why You Should Care. 2009. Available from: `http://www.analog.com/static/imported-files/tutorials/MT-001.pdf`

[3] Meyer-Baese U. Digital Signal Processing with Field Programmable Gate Arrays. Third ed. New York: Springer Berlin Heidelberg; 2007.

[4] Lyons RG. Understanding Digital Signal Processing. Third Edition. Boston: Pearson Education, Inc.; 2011.

[5] Vaughan RG, Scott NL, White DR. The Theory of Bandpass Sampling. IEEE Trans. on Signal Processing. vol.39(9), pp. 1973-1984. 1991.

[6] Losada RA. Digital Filters with MATLAB. The MathWorks, Inc.; 2008 May 18. Available from: `http://www.mathworks.com/tagteam/55876_digfilt.pdf`

[7] Pozar DM. Microwave and RF Wireless Systems. Chapter 7. New York: John Wiley & Sons, Inc.; 2001.

[8] dspGuru by Iowegian International. Decimation. [cited 2013 May 8]. Available from: `http://www.dspguru.com/dsp/faqs/multirate/decimation`.

[9] dspGuru by Iowegian International. Interpolation. [cited 2013 May 8].Available from: `http://www.dspguru.com/dsp/faqs/multirate/interpolation`.

[10] Uyemura JP. Introduction to VLSI Circuits and Systems. New Jersey: John Wiley & Sons, Inc.; 2002.

[11] Actel. Global Clock Networks in Actel Antifuse Devices. Application Note AC207. 2004. Available from: `http://www.actel.com/documents/GlobalClk_AN.pdf`.

[12] Morris K. FPGAs in Space: Programmable Logic in Orbit. Electronic Engineering Journal. August 3. 2004. Available from: `http://www.eejournal.com/archives/articles/20040803_space/`

[13] Roosta R. A Comparison of Radiation-Hard and Radiation-Tolerant FPGAs for Space Applications. NASA Electronic Parts and Packaging Program. December 2004.

[14] Microsemi. RTAX-S/SL and RTAX-DSP Radiation-Tolerant FPGAs. Datasheet. Revision 15. May 2012. Available from: `http://www.actel.com/documents/RTAXS_DS.pdf`.

[15] Xilinx. Space-Grade Virtex-4QV Family Overview. Product Specification DS653 v2.0. April 2010. Available from: `http://www.xilinx.com/support/documentation/data_sheets/ds653.pdf`.

[16] Actel. Actel HDL Coding. Style Guide. 2009. Available from: `http://www.actel.com/documents/hdlcode_ug.pdf`.

[17] COSPAS-SARSAT. Specification for COSPAS-SARSAT 406 MHz distress beacons. C/S T.001. Issue 3 – Revision 13. October 2012 Available from: `http://www.cospas-sarsat.org/images/stories/SystemDocs/Current/CS_T001_OCT_2012.pdf`

[18] Hein GW, Godet J, Issler JL, Martin JC, Erhard P, Rodriguez RL et al. Status of Galileo Frequency and Signal Design. Brussels: Galileo Signal Task Force of the European Commission. 2002. Available from: `http://ec.europa.eu/dgs/energy_transport/galileo/doc/galileo_stf_ion2002.pdf`

[19] Elbert BR. The satellite communication applications handbook. Norwood: Artech House, Inc.; 2004.

[20] Texas Instruments. ADS5463-SP, Class V, 12-bit, 500-MSPS Analog-to-Digital Converter. Datasheet. March 2008. Revised August 2012. Available from: `http://www.ti.com/lit/ds/sgls378d/sgls378d.pdf`.

[21] Agarwal A, Lang JH. Foundations of Analog and Digital Electronic Circuits. San Francisco: Morgan Kaufmann Publishers; 2005.

[22] Hassani S. Mathematical Physics: For Students of Physics and Related Fields. Chapter 7. New York: Springer-Verlag; 2000.

[23] Altera. Using PLLs in Cyclone Devices. Application Note AN251 v.1.2, March 2003. Available from: `http://www.altera.com/literature/hb/cyc/cyc_c51006.pdf`.