HIGH SPEED DIGITAL SAMPLE RATE CONVERSION FOR A REAL-TIME GROUND STATION RECEIVER

by

Brian J. Carrick

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTERS OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Jacob Gunther Major Professor	Dr. Todd Moon Committee Member

Dr. Charles Swenson
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2016

Abstract

High speed digital sample rate conversion for a real-time ground station receiver

by

Brian J. Carrick, Masters of Science

Utah State University, 2016

Major Professor: Dr. Jacob Gunther
Department: Electrical and Computer Engineering

Small satellites are becoming a popular spacecraft option because of their low cost, and complexity. Past small satellites had simple communication systems with low data rates that rarely exceeded 38.4 kbps. Engineers, researchers, and students, from Space Dynamics Laboratory and Utah State University, have recently created and launched a pair of small satellites called the Dynamic Ionosphere CubeSat Experiment (DICE). This mission was revolutionary because of its high-speed data link, with data rates up to 1.5 Mbps.

Due to the high-speed data link, it was necessary to create a ground station capable of receiving and processing the received data. For the DICE mission the ground station developed was not capable of receiving and processing the data in real-time. Because the current ground station was not capable of running in real-time it became necessary to develop a real-time solution. The real-time ground station receiver is a hardware solution, that implements signal processing algorithms in an FPGA in real-time. This report describes the theory, design, simulation, implementation, and testing of the sampling rate conversion portion of the system, as well as, other work that can be done to complete the real-time ground station receiver.

(109 pages)

Public Abstract

High speed digital sample rate conversion for a real-time ground station receiver

by

Brian J. Carrick, Masters of Science

Utah State University, 2016

Major Professor: Dr. Jacob Gunther
Department: Electrical and Computer Engineering

Small satellites are becoming a popular spacecraft option because of their low cost, and complexity. Past small satellites had simple communication systems with low data rates.Engineers, researchers, and students, from Space Dynamics Laboratory and Utah State University, have recently created and launched a pair of small satellites called the Dynamic Ionosphere CubeSat Experiment (DICE). This mission was revolutionary because of its high-speed data link.

Due to the high-speed data link, it was necessary to create a ground station capable of receiving and processing the received data. For the DICE mission the ground station developed was not capable of receiving and processing the data in real-time. Because the current ground station was not capable of running in real-time it became necessary to develop a real-time solution. This report describes the theory, design, simulation, hardware implementation, and testing of the sampling rate conversion portion of the system, as well as, other work that can be done to complete the real-time ground station receiver.

To my beautiful wife Dara, and also my amazing children, Makayla, Aylan, and Samuel. I love you all.

Acknowledgments

I would like to thank my wife, Dara Carrick, for her love, support, and hard work. She is an amazing woman, wife, and mother. She has helped me to keep moving forward, even when I was ready to quit. I want to thank her for believing in me when I did not believe in myself. This accomplishment is as much hers as it is mine. I love you Dara. I would also like to thank my children, Makayla, Aylan, and Samuel, for giving me motivation to be a better man, and father.

I would also like to thank my parents, Joe and Debbie Carrick, and my wifes parents, Brad and Clio Grob, who helped when times were hard. I am also thankful for Dr. Jacob Gunther, who saw potential in me that I did not see myself, and encouraging me to pursue a masters degree, also for being a great mentor, advisor, and friend. I am also grateful to Dr. Todd Moon, and Dr. Charles Swenson for their willingness to serve on my committee, reviewing my work, and for teaching me so much through the years. I would also like to thank Space Dynamics Laboratory for providing the hardware that I used to develop and test my project.

Brian J. Carrick

Contents

List of Figures

Acronyms

ADC	analog to digital converter
BSP	board support package
CD	compact disk
DDS	direct digital synthesizer
DICE	dynamic ionospheric cubesat experiment
DIO	digtial input/output
DSP	digital signal processing
FFT	fast Fourier transform
FIFO	first in first out
FIR	finite impulse response
FPGA	field programmable gate array
GSPS	giga-sample per second
IP	Intellectual property
OQPSK	offset quadrature phase shift keying
PC	personal computer
PCIe	peripheral component interconnect express
RAM	random access memory
RTGS	real-time ground station
SRC	sampling rate conversion
SRRC	square root raised cosine
VHDL	VHSIC hardware description language
VHSIC	very high speed integrated circuits

Chapter 1

Introduction

1.1 Background

Each year several state of the art small satellites are developed for various missions, and are becoming more popular because of their low cost, and complexity. In the past these small satellites have had simple communication systems with fairly low data rates that rarely exceeded 38.4 kbps [1]. Space Dynamics Laboratory and Utah State University, using a team of professionals and students, recently created and launched a pair of satellites called the Dynamic Ionosphere CubeSat Experiment (DICE) [2]. DICE was a mission that had two objectives, with the primary objective being to collect data from science instruments aboard the satellite, and the second objective to demonstrate high data rate communications from low earth orbit to earth. The DICE mission was groundbreaking because of its demonstrated ability to have high speed communications on a small low cost satellite, allowing for more data collection and more useful missions.

1.2 Problem being solved and proposed solution

With the creation of high speed communication options for small satellites, a low cost real-time ground station receiver must be developed. For the DICE mission, a ground station was developed that simply took the raw samples from the analog to digital converter (ADC) and dumped them to a hard drive during a satellite overpass. The samples from the ADC that were stored on a hard drive then had to be processed so that the satellite telemetry data could be viewed and analyzed. This was problematic because flight telemetry was not able to be viewed until after the overpass was completed. Creating a real-time ground station receiver would allow the ground station operator to see, in near real-time, telemetry

from the satellite. This could be beneficial to small satellite programs by allowing operators to see potential problems and take corrective action before more serious problems arise. To address this, the proposed project will help the CubeSat community by developing a ground station receiver capable of real-time operation.

The real-time ground station receiver is a collection of hardware, firmware and software, that will ultimately receive digital communications from small satellites, and perform all the necessary signal processing in real-time. This signal processing will take the signal from the ADC and will output error corrected bits. A host application can then decode and use those bits to view and process telemetry data in real-time.

1.3 Scope of project

The real-time ground station receiver will consist of multiple stages of signal processing implemented in hardware. The following outline the specifications of the DICE satellite transmitted signal:

- 3 Mbps bit rate.
- 2.637 Mbps information bit rate with forward error correction coding.
- 468 MHz band center frequency.
- Square root raised cosine (SRRC) pulse shaping.
- 3 MHz bandwidth.
- Offset quadrature phase shift keying (OQPSK) modulation.
- 225/256 error correction coding.

From the specification just listed, the real-time ground station receiver must be capable of the following:

- Downsample signal to 12 MHz.
- Mix signal to baseband and preserve spectrum.

- Perform OQPSK demodulation.
- Automatic gain control.
- Doppler phase correction.
- Interference cancellation.
- Forward error correction.
- Convert symbols to bits.
- Store bits to hard disk.

This project and report will focus on, digital down conversion of the signal of interest to 12 MHz, mixing the signal to baseband, and preserving the specified frequency band.

1.4 Chapter overview

The project report will describe the theory of digital sampling rate conversion, as well as optimizations for real-time operation. As such, the report will be structured as follows. Chapter 2 will describe the theory and math of sampling rate conversion, and signal processing techniques for optimizing the signal processing. Chapter 3 will discuss the design of the sampling rate conversion required for this specific system. There will also be simulations of the system, optimizations used in the sampling rate conversion, and the necessary anti-aliasing filter designs. Chapter 4 will describe the hardware being used on this project, as well as, the development tools and softwares needed for developing on the hardware. Chapter 5 discusses the actual firmware design, implementation, simulation, and real-time tests, as well as, showing simulation and real time test results. Chapter 6 contains the conclusion of this project report and will discuss future work including integrating the sampling rate conversion portion of the project with the remaining real-time ground station receiver.

Chapter 2

Rational sampling rate conversion - A basic overview

2.1 Introduction to sampling rate conversion

Sampling rate conversion is the process of changing the sampling rate of a signal. This process is used in many digital systems, and can be especially useful in systems with limited resources that require real-time operation. This chapter will discuss the theory of Sampling rate conversion starting with basic upsasmpling and downsampling. This chapter will also discuss the the filtering required to prevent aliasing, as well as, techniques for optimizing sampling rate conversion systems for use in high speed, real-time applications [3, 4].

2.1.1 Continuous time to discrete time conversion

Continuous time signals are defined at every instant of time, whereas *discrete time* signals are only defined at discrete samples of time [4]. We can represent these signals using the notation of t for continuous time, and n for discrete time steps where $t \in \mathbb{R}$ and $n \in \mathbb{Z}$. Using this notation, a continuous time function is represented as $x(t)$ with parentheses, while a discrete function is represented as $x[n]$ with square brackets. Figure 2.1 shows a graphical representation of a) a continuous time signal, and b) a discrete time signal. The conversion of a continuous time signal to a discrete time signal is done by taking periodic samples from the continuous time signal, the time interval between each sample is called the *sample period* and is denoted T. The *sampling rate* or *sampling frequency* F_s is defined as $F_s = 1/T$, and is the number of samples per unit of time, usually seconds, and its units are Hz or samples per second when using sampling frequency or sampling rate, respectively. A continuous time function $x(t)$ can be sampled to get a discrete time function $x[n] = x(nT)$, for $n \in \mathbb{Z}$.

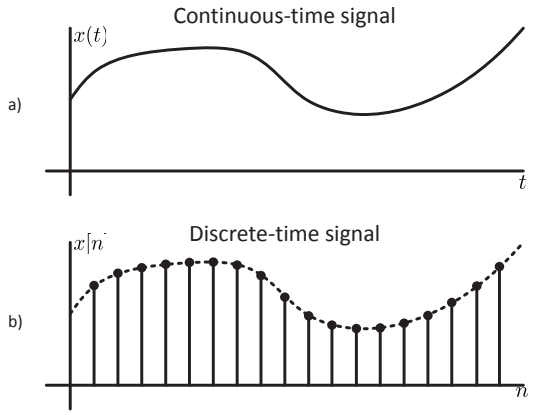

Fig. 2.1: Representation of a) continuous and b) discrete time signal.

The conversion from continuous time to discrete time is usually accomplished using an analog to digital (ADC). Figure 2.2 shows a block diagram representing the conversion process from a continuous time, analog signal to a discrete time, digital signal. The process is made up of three steps, sampling, quantizing, and coding. The sampling process samples the analog, continuous signal and holds its measurement for the duration of the sample period, and is sampled at the sample rate. The quantizer converts the analog amplitude into a discrete amplitude. The final stage of the analog to digital conversion is the coding stage where the quantized values are encoded to digital bits [3, 4].

2.1.2 Rational sampling rate conversion

Sampling rate conversion is the process of changing the discrete time step or sampling period between samples. There are three main ways a sampling rate can change, increasing

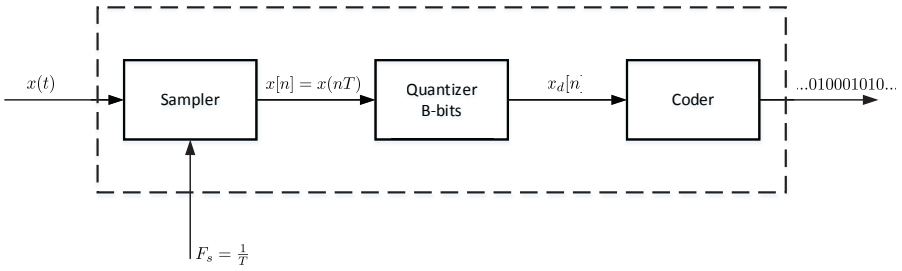

Fig. 2.2: Block diagram of analog to digital conversion process.

the sampling rate, decreasing the sampling rate, and a combination of increasing and decreasing. A sampling rate can be changed by multiplying or dividing the sample period T by an integer, or rational number. The following sections discuss the methods for changing the sampling rate in a discrete time system.

2.2 Increasing sampling rate - Upsampling

Increasing the sampling rate is done by dividing the sample period T by an integer I, with the new sampling period $T_n = T/I$. This process is called *upsampling* and is performed by inserting $I-1$ zeros between each sample. This changes the sampling rate to $F_{ns} = IF_s = I/T_n$. Figure 2.3 shows the upsampling process for $I = 4$. As we look at the frequency domain effects of upsampling we see that the frequencies are compressed and are now periodic $X_u(e^{\pm j\pi}) = X(e^{\pm j\pi I})$. There are then $I-1$ copies of the spectrum called *images*, with are then filtered out using an *interpolation filter* with a cutoff frequency $\omega_c = \pi/I$, and gain of I to compensate for the $1/I$ decrease in signal bandwidth. Figure 2.4 shows the frequency domain representation of upsampling for $I = 4$. This shows that if the cutoff frequency is not π/I then signal energy from the images remains in the interpolated signal, this is called *post-aliasing*. The *interpolator* is a discrete time system used to increase the sampling rate, or upsample a signal, and then remove the images caused by the upsampling process, using an interpolation filter [4].

2.3 Decreasing sampling rate - Downsampling

Decreasing the sampling rate is done by multiplying the sample period T by an integer

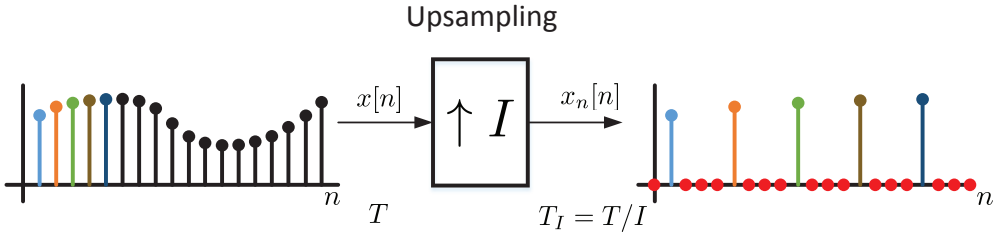

Fig. 2.3: Time domain representation of upsampling for $I = 4$.

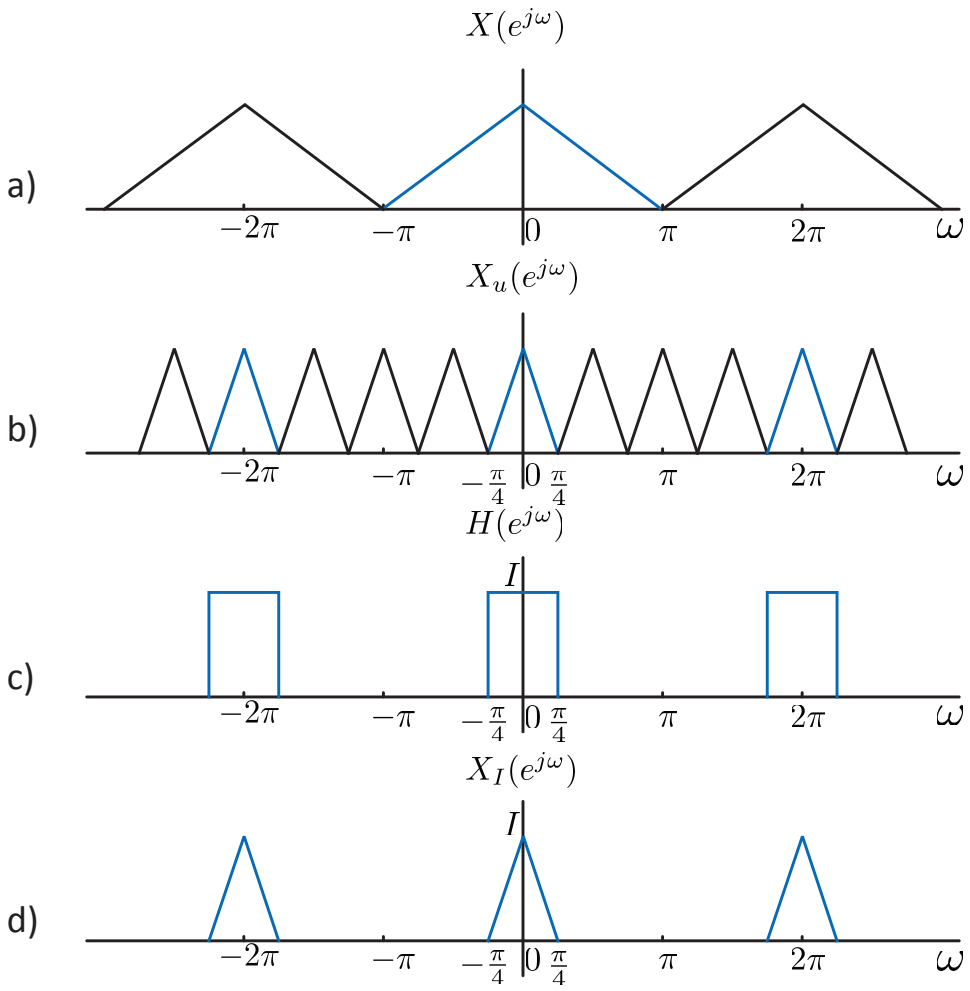

Fig. 2.4: Frequency domain representation of upsampling for $I = 4$, a) original spectrum, b) upsampled spectrum, c) interpolation filter, and d) final spectrum after filtering.

D, with the new sampling period $T_n = DT$. This process is called *downsampling* and is performed by removing $D-1$ samples out of every D samples from a signal. This changes the sampling rate to $F_{ns} = F_s/D = 1/(DT_n)$. Figure 2.5 shows the downsampling process for $D = 2$. From the Figure 2.6, it is shown that the replicas of $X(e^{j\Omega T})$, located at every $n2\pi/T$ with $\{n \in 0, \pm 1, \pm 2, ...\}$ are shifted closer, where $X_D(e^{j\Omega T_D})$ has replicas now located at $n2\pi/T_D$ with $\{n \in 0, \pm 1, \pm 2, ...\}$. *Aliasing* is caused when the now shifted replicas of $X_D(e^{j\Omega T_D})$ overlap, causing distortion in the spectrum of the signal. To avoid aliasing, the spectrum of the signal should be bandlimited, so that the maximum frequency $\Omega_{max} \leq \Omega_s/(2D)$ before downsampling. A signal is usually made to be bandlimited by filtering the signal with a low-pass anti-aliasing filter with a stop-band frequency at $\omega_s = \pi/D$. This filter ensures that replicas, that have been shifted closer together by the downsampling operation, have no overlap. Figure 2.6 show downsampling with aliasing, and downsampling with an anti-aliasing filter applied. The *decimator* is a discrete time system used to decrease the sampling rate, or downsample a signal, first by filtering with an anti-aliasing filter, and then downsampling the signal [4].

2.4 Fractional sampling rate conversion

A sampling rate can also be increased or decreased by a non integer rational number. This is accomplished by both, upsampling by an integer number I, and downsampling by an integer number D. The sampling rate is then changed to $F_{ns} = F_s(I/D)$, and the sample period is now $T_n = (D/I)T$. To increase the sample rate by a rational number, then $I/D > 1$, and conversely to decrease the sampling rate by a rational number, then

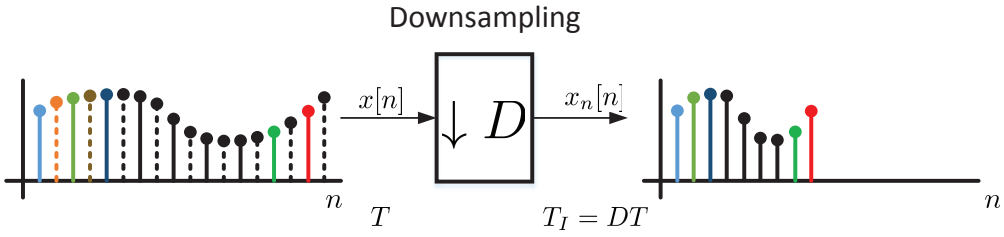

Fig. 2.5: Time domain representation of downsampling for $D = 2$.

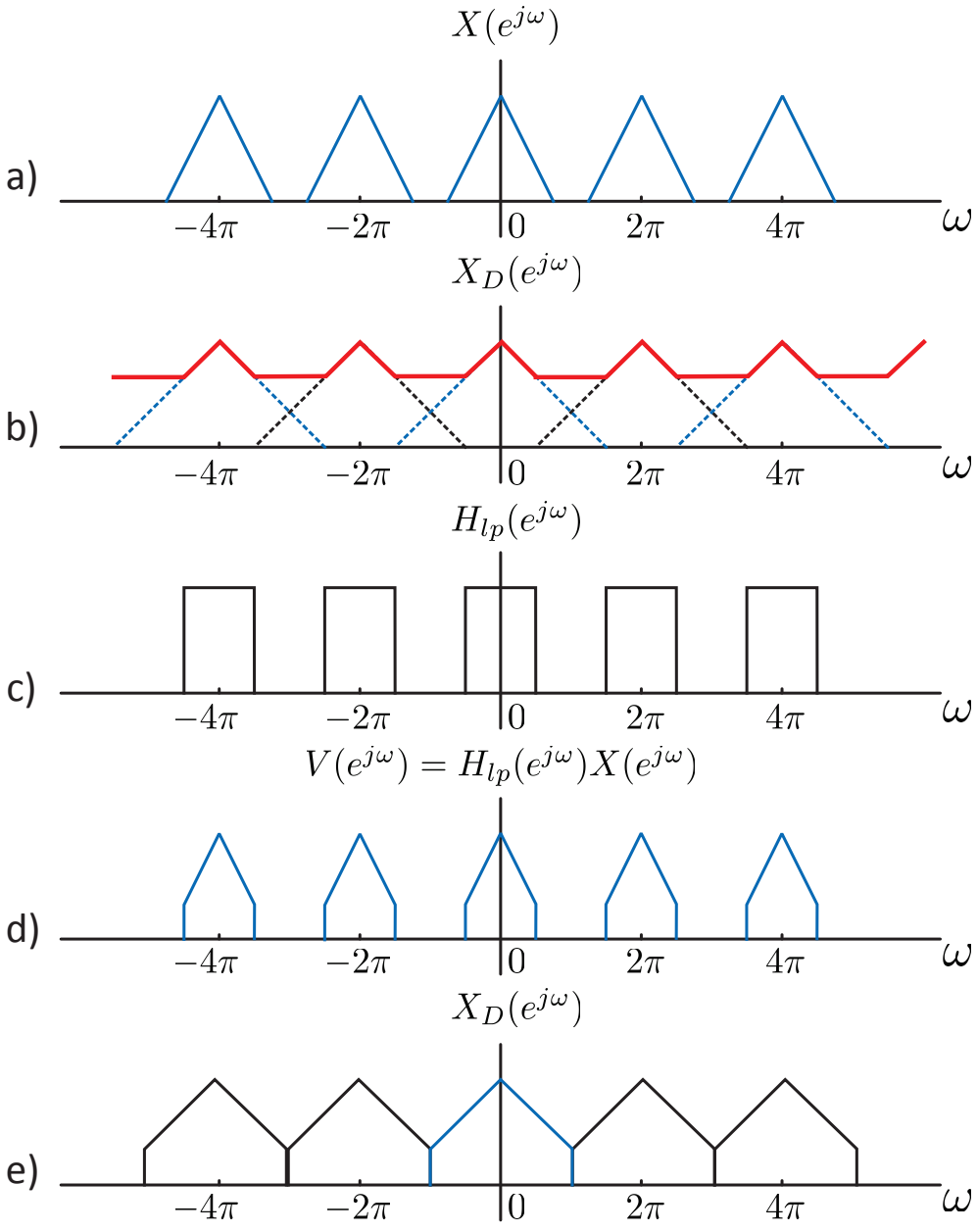

Fig. 2.6: Frequency domain representation of downsampling, with and without aliasing for $D = 2$, a) original spectrum, b) aliased spectrum, c) anti-aliasing filter, d) filtered spectrum, e) final spectrum without aliasing.

$I/D < 1$. The fractional sampling rate converter is implemented by first interpolating the signal with and upsample by I and then filtering with an interpolation filter. Following the interpolator, is a decimator that first filters the signal with an anti-aliasing filter, and then downsamples by D. Because the anti-aliasing filter and the interpolation filter are in cascade and are operating at the same rate, they can be combined into a single filter. This filter is then selected to have a gain of I from the interpolation filter and the cutoff or stopband frequencies are selected as the $min(\pi/I, \pi/D)$. Figure 2.7 shows a cascaded interpolator and decimator for a fractional rate system, and the same system with the combined interpolation and anti-aliasing filters [3, 4].

2.5 FIR filters and polyphase filter structures

An *finite impulse response* (FIR) system, that has real coefficients is described by the equation

$$y[n] = \sum_{k=0}^{M} b_k x[n-k], \tag{2.1}$$

which has an impulse response described by the following

$$h[n] = \begin{cases} b_n, & n = 0, 1, ..., M \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

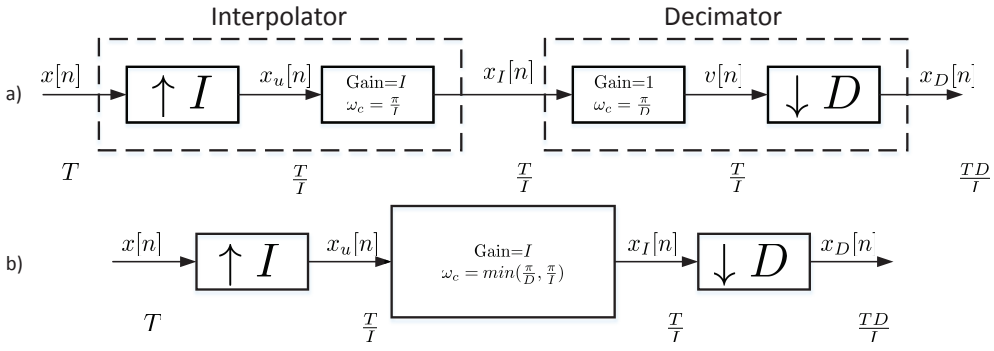

Fig. 2.7: Rational sampling rate converter, a) with cascaded interpolator and decimator, and b) with combined interpolation and anti-aliasing filter.

and has the transfer function

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{n=0}^{M} b_k z^{-n} = \sum_{n=0}^{M} h[n] z^{-n}. \tag{2.3}$$

The FIR filter described by the impulse response int Eq. (2.3) can be restructured using a technique called a *polyphase decomposition*, which becomes useful in the the optimization of rational *multirate* systems. A multirate system is any system that uses sampling rate conversion, meaning that the rates at the beginning of the system are different than the rates at other points of the system. Using the transfer function in Eq. (2.3), for example, if $M = 7$ the transfer function can be written as follows

$$H(z) = h[0] + h[1]z^{-1} + h[2]z^{-2} + h[3]z^{-3} + h[4]z^{-4} + h[5]z^{-5} + h[6]z^{-6} + h[7]z^{-7}. \tag{2.4}$$

This can then be changed into the following,

$$\begin{aligned} H(z) &= (h[0] + h[2]z^{-2} + h[4]z^{-4} + h[6]z^{-6}) \\ &\quad + (h[1]z^{-1} + h[3]z^{-3} + h[5]z^{-5} + h[7]z^{-7}) \\ &= (h[0] + h[2]z^{-2} + h[4]z^{-4} + h[6]z^{-6}) \\ &\quad + z^{-1}(h[1] + h[3]z^{-2} + h[5]z^{-4} + h[7]z^{-6}). \end{aligned} \tag{2.5}$$

From this, Eq. (2.5) can be rewritten using the notation

$$\begin{aligned} H_0(z) &= (h[0] + h[2]z^{-1} + h[4]z^{-2} + h[6]z^{-3}) \\ H_1(z) &= (h[1] + h[3]z^{-1} + h[5]z^{-2} + h[7]z^{-3}). \end{aligned} \tag{2.6}$$

Thus, the transfer function in (2.4) can also be rewritten as

$$H(z) = H_0(z^2) + z^{-1}H_1(z^2). \tag{2.7}$$

This process can be, more generally described, for an L branch polyphase decomposition by the following

$$H(z) = \sum_{m=0}^{L-1} z^{-m} H_m(z^L), \tag{2.8}$$

where

$$H_m(z) = \sum_{n=0}^{\lfloor (M+1)/L \rfloor} H[Ln+m] z^{-n}, \tag{2.9}$$

and M is the order of the filter. Figure 2.8 shows different polyphase decompositions of filters for different values of L. These filters, as already stated, are useful in the reducing the computational complexity of rational sampling rate converters [3, 4].

2.6 Noble identities

The last thing that needs to be discussed, before sampling rate conversion optimization, are the *noble identities*. The noble identities allow cascaded upsamplers or downsamplers, to be moved to the other side of their corresponding interpolation filters or anti-aliasing filters, respectively. The combination of polyphase decomposition and the noble identities allows for more efficient computations. Figure 2.9 shows an interpolator and decimator and their equivalent relations using the noble identities [3].

2.7 Sampling rate conversion optimizations

As systems get faster, computational complexity, becomes more of an issue. Polyphase decomposition of FIR filters and the noble identities are tools that can be used to decrease the computational complexity. Using these tools, an efficient sampling rate conversion system can be developed, that will allow for decreased computational complexity, and increased performance. The following three examples describe the optimizations that can be implemented for upsampling, downsampling, and fractional rate changes.

2.7.1 Upsampling example

In this example the sampling rate will be increased by $I = 3$, from $F_s = 100$ Hz to $F_s = 300$ Hz. Figure 2.10 shows the interpolator being used for this example. The first step in

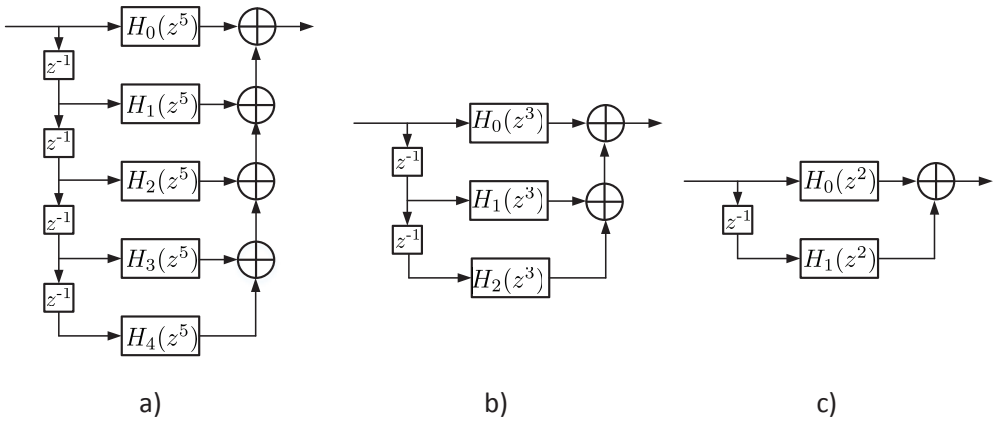

Fig. 2.8: Different polyphase decompositions for a) $L = 5$, b) $L = 3$, and c) $L = 2$.

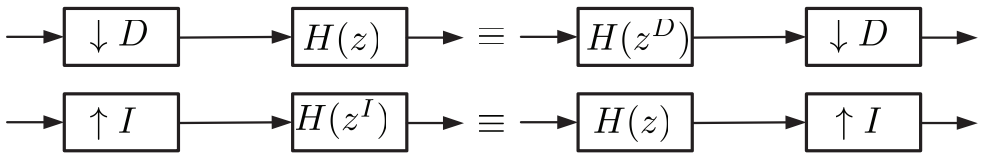

Fig. 2.9: Noble identities for interpolator and decimator.

the optimization is to take the filter and perform an $L = 3$ branch polyphase decomposition. The interpolation stage can be placed on each of the three branches. The next step it to apply the noble identities to each of the three branches and translate the upsampler across each of the polyphase filters. Figure 2.11 shows the polyphase decomposition of the filter, and the interpolation system after the application of the noble identities.

2.7.2 Downsampling example

In this example the sampling rate will be decreased by $D = 2$, from $F_s = 200$ Hz to $F_s = 100$ Hz. Figure 2.12 shows the decimator being used for this example. The first step in the optimization, as with the interpolator described in Section 2.7.1, is to take the filter and perform an $L = 2$ polyphase decomposition. The downsampler can be placed on each of the two branches. The next step is to apply the noble identities to each of the two branches and translate the downsampler across each of the polyphase filters. Figure 2.13 shows the polyphase decomposition of the filter, and the decimation system after the application of the noble identities.

2.7.3 Fractional rate example

For this example the sampling rate will be changed by a fractional rate I/D where $I = 3$, and $D = 2$, from $F_s = 600$ Hz to $F_s = 900$ Hz. Figure 2.14 shows the fractional rate converter being used for this example. The process of optimizing a fractional rate converter is a little bit more involved than the simpler interpolation and decimation examples. The first step of the optimization is to take the filter and perform an $L = 2$ branch polyphase decomposition. Then use the noble identities to translate the downsampler across the filter.

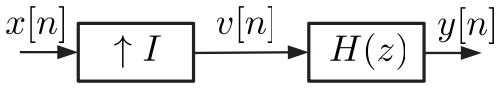

Fig. 2.10: Interpolator for $I = 3$ example.

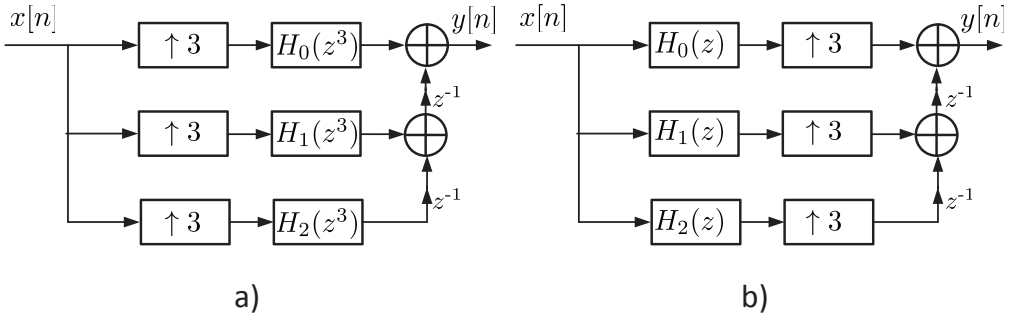

Fig. 2.11: Interpolator for $I = 3$ example with a) polyphase structure, and b) after application of noble identities.

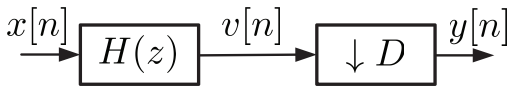

Fig. 2.12: Decimator for $D = 2$ example.

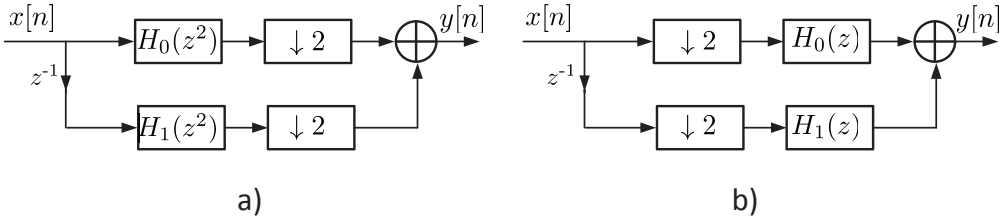

Fig. 2.13: Decimator for $D = 2$ example with a) polyphase structure, and b) after application of noble identities.

Using some mathematical manipulation beyond the scope of this report the upsampler and downsampler swap locations [3, 5]. The next step it to perform a second polyphase decomposition with $L = 3$ branches. The final step is to apply the noble identities one last time to translate the upsampler across each of the three branches in both main branches. Figure 2.15 shows the polyphase decompositions of the filter, and the application of the noble identities.

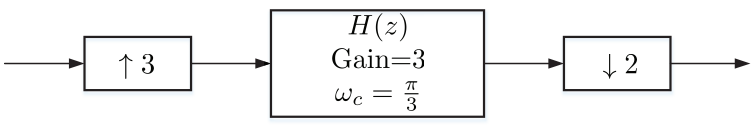

Fig. 2.14: Fractional rate converter for $I = 3$, and $D = 2$ example.

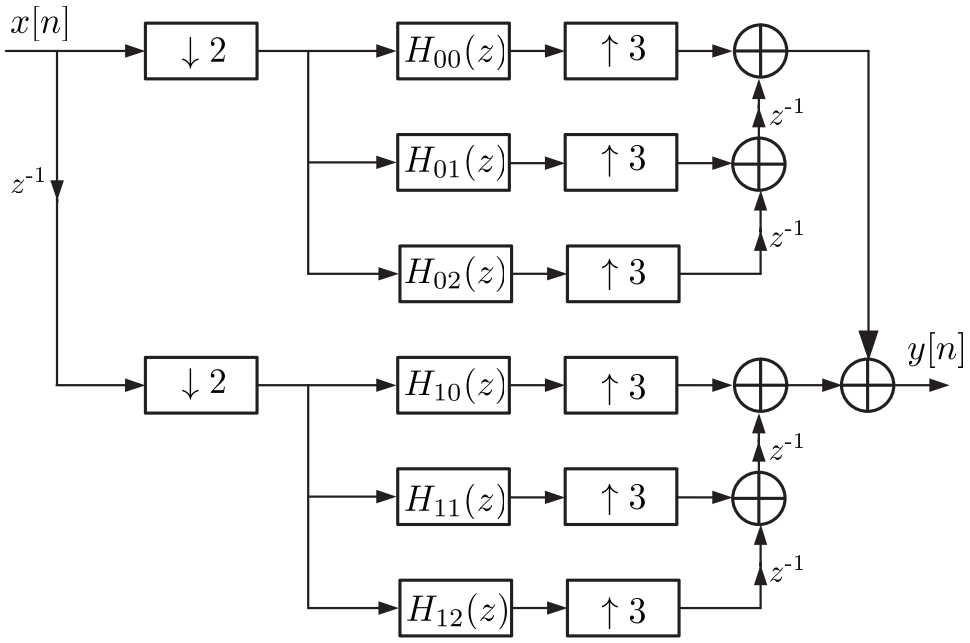

Fig. 2.15: Fractional rate converter for $I = 3$, and $D = 2$ example with polyphase structure, and application of noble identities.

Chapter 3

System design

The real-time ground station receiver consists of many systems that are required to get correct bits from the system, Section 1.3 lists the required sub-systems. Of these subsystems this project and design will focus on the sampling rate conversion and digital downconversion of the signal of interest to baseband. The remaining sub-systems will be designed and developed as time and funding permit.

The sampling rate conversion system will consist of multiple stages of downsampling, as well as, the digital mixing to shift the signal to baseband. Each stage of the design will be optimized to run in real-time on and FPGA. The optimization process will mostly consist of polyphase decompositions of the anti-aliasing filters, and using the noble identities to filter the samples at a lower rate, see Sections 2.5, 2.6, 2.7. This optimization, is especially necessary due to the hardware being used in this design, see Section 4.1.1. The following sections in this chapter will discuss the requirements, design, and simulation of the theoretical system.

3.1 Design requirements

The following are the design requirements for the sample rate conversion system, and the digital downconversion.

1. The system shall be capable of real-time operation without loss of data.
2. The system shall be capable of storing data recorded in a PC hard disk.
3. The system shall sample an analog signal with a sampling rate of 1600 MHz.
4. The system shall receive sample packets from the ADC at a rate of 200 MHz, with 8 samples per packet.

5. The system shall perform an initial downsampling by 8 with a polyphase optimization, as required by the hardware.

6. The system clock of the FPGA shall be faster than the ADC clock to prevent ADC first in first out (FIFO) buffer overflow.

7. The system shall convert the sampling rate of the samples from 1600 MHz to 12 MHz.

8. The sample rate conversion shall not cause aliasing.

9. The system shall digitally mix the carrier frequency of 468 MHz to baseband.

10. The system shall preserve the 3 MHz band centered at 468 MHz during the sampling rate conversion and the digital mixing.

11. The system shall provide at the output, both an in-phase, and quadrature-phase signal, and any necessary valid signal.

3.2 Conceptual design

The purpose of the conceptual design is to determine a system that could feasibly be implemented in real-time. There are many different ways the system could be designed, and the design selected was chosen due to its computational efficiency. The design developed consists of multiple stages, with the first being a downsample by 8 stage as required in Section 5. The filter for the first stage needs to preserve the 3 MHz band as required by Section 10. In order to preserve the band, it is necessary to mix the signal down by 1/4 the sampling rate, or 400 MHz, so that the anti-aliasing filter for the downsampler does not attenuate out the band of interest. Mixing down by 1/4 the sampling rate is very efficient in OQPSK systems, this is due to the inherent periodicity of complex exponentials. Using Euler's identity $e^{j\omega} = \cos(\omega) + j\sin(\omega)$, for $\omega = n\pi/4$, and $n = 0, 1, 2, 3, \ldots$, which gives $e^{j\omega} = \{1, j, -1, -j, 1, j, \ldots\}, or \cos(\omega) = \{1, 0, -1, 0, 1, 0, -1, 0 \ldots\}$, and $\sin(\omega) = \{0, 1, 0, -1, 0, 1, 0, -1 \ldots\}$, which allows for a mix down, using the in-phase and quadrature-phase branches of the system, by 1/4 the sampling rate with no multiplies. Figure 3.1 shows the conceptual design of the sampling rate conversion system.

The second stage consists of the final mix down to baseband by 68 MHz or .38 of the current sampling rate. This is followed by a anti-aliasing filter and a downsample by 5. The third stage is a fractional rate converter with an upsample by 3, followed by an anti-aliasing/interpolating filer, and then downsampled by 2. The last stage, is an anti-aliasing filter followed by an downsample by 5. This brings the band to baseband, and reduces the sampling rate to 12 MHz.

3.2.1 System simulation using Matlab

To verify that the conceptual design algorithm for the sampling rate conversion system worked as required, a simulation was designed using Matlab to test the systems functionality. Ideally, it would be desirable to have an end-to-end transfer function of the designed system, however this is not possible because mixing and sampling rate conversion systems are not linear, time-invariant systems. Therefore, the best that can be done is to show simulations of the signals passing through the system. The simulation created a sample signal in our band at 468.5 MHz. This signal was also introduced with noise and other sinusoids out of the band, to ensure that the band was preserved. The Matlab code used in the simulation can be found in appendix B.

Test signal and stage 1 simulation

Figure 3.2(a) Shows the original test signal, and Figure 3.2(b) shows its spectrum. The next Figure 3.3(a) shows the spectrum of the signal after mixing down by 1/4 the sampling frequency, and then, Figure 3.3(b) shows the spectrum after running it through the first low pass filter used in Figure 3.4. The last step of the simulation for stage 1 is to downsample by 8, with the downsampled signal being shown in Figure 3.5(b), and the spectrum of the downsampled signal shown in Figure 3.5(a).

Stage 2 simulation

The first part of stage 2 is to mix the signal down, by 68 MHz or .34 of the sampling frequency, to baseband. Figure 3.6(a) shows the spectrum of the signal that has been mixed

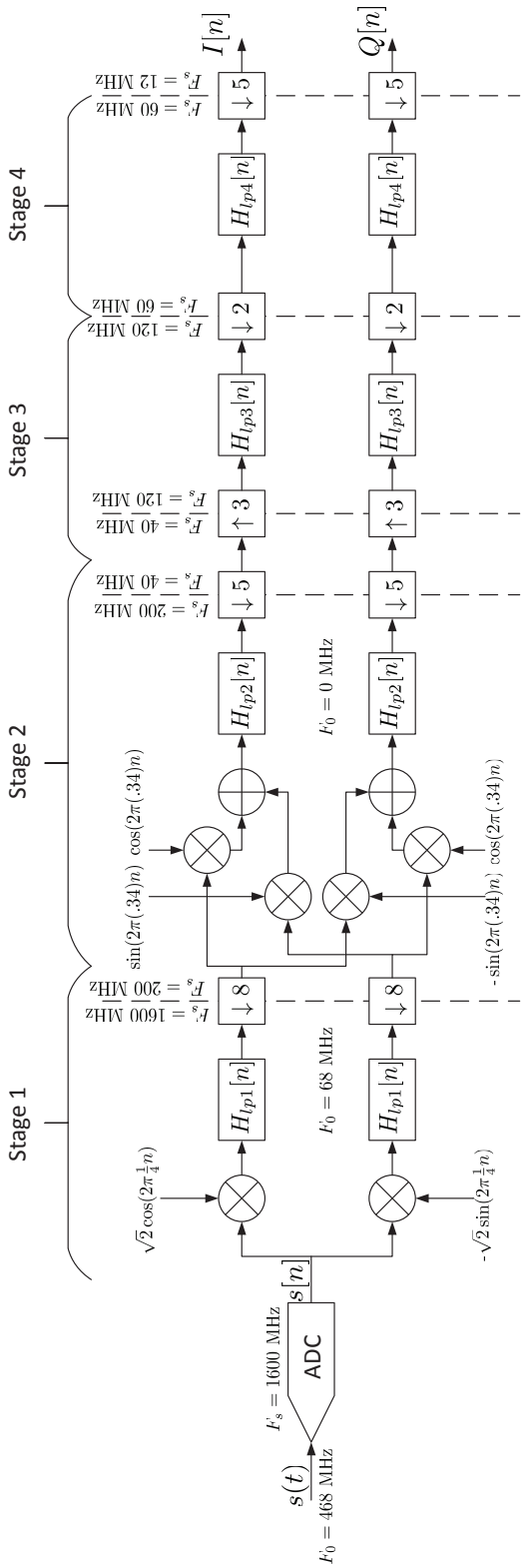

Fig. 3.1: Conceptual design of the sampling rate conversion system.

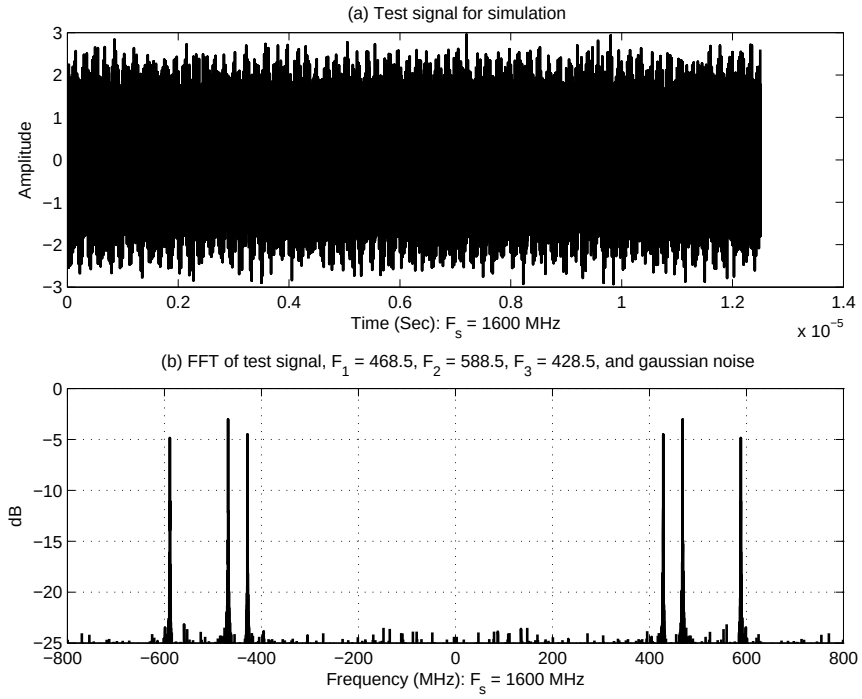

Fig. 3.2: Test signal (a), and its spectrum (b), $F_s = 1600$ MHz.

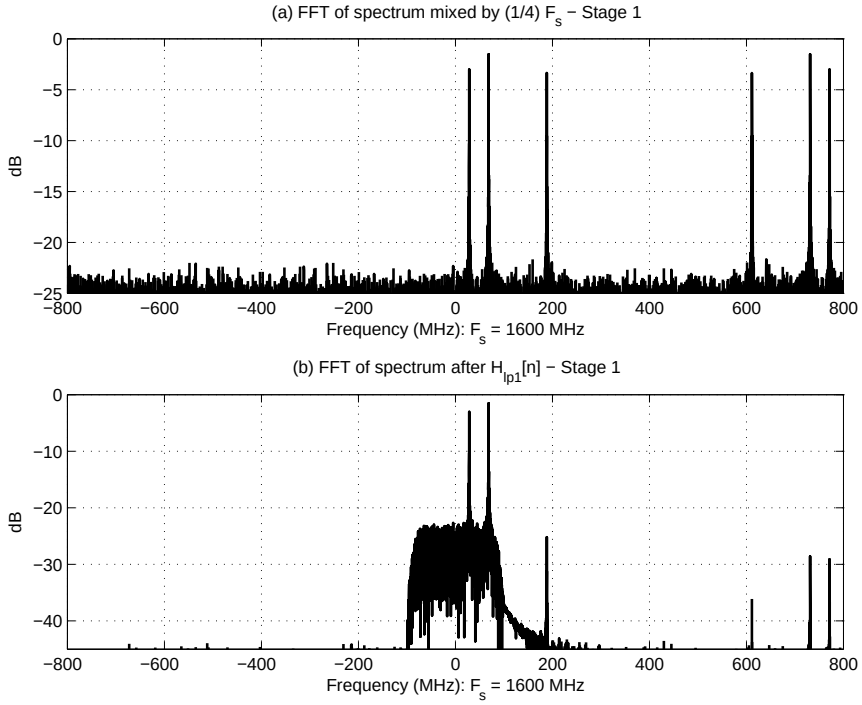

Fig. 3.3: Test signal spectrum after, (a) first mixing down by 1/4 the sampling frequency, and (b) low pass filtering, $F_s = 1600$ MHz.

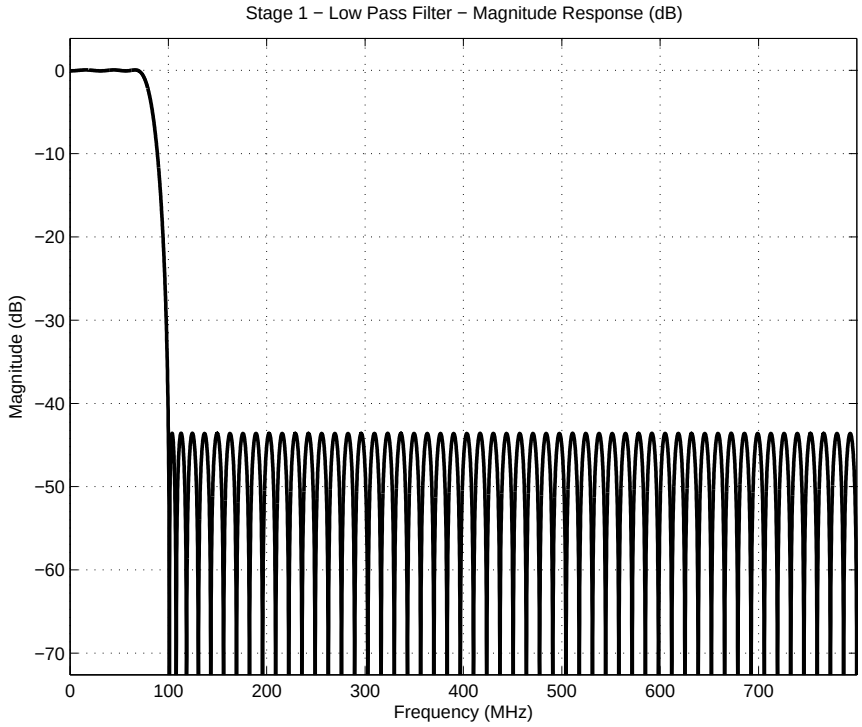

Fig. 3.4: Low pass filter used in stage 1 of the design, $F_s = 1600$ MHz.

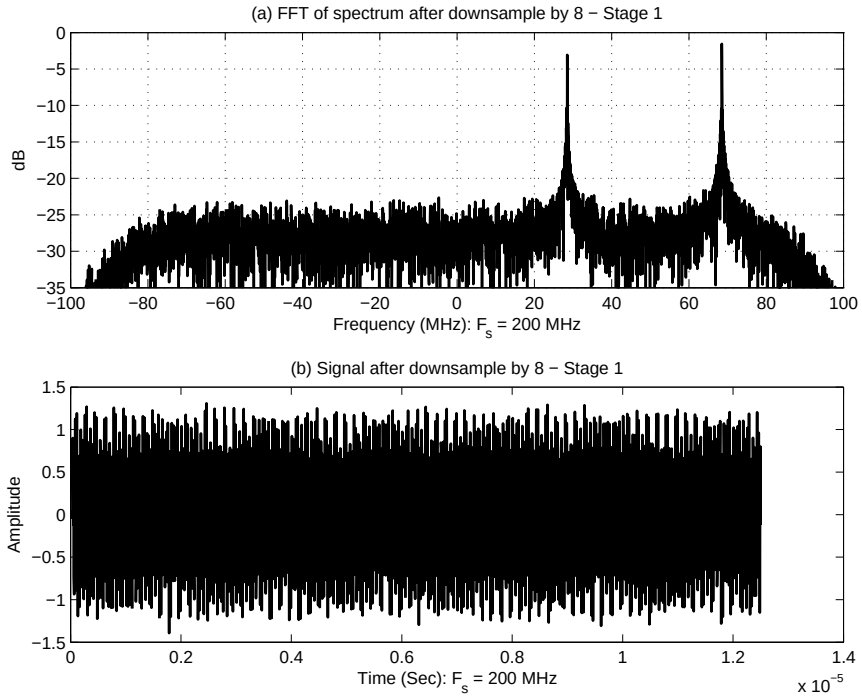

Fig. 3.5: Downsample by 8, (a) signal spectrum after downsampling, and (b) signal after downsampling, $F_s = 200$ MHz.

down to baseband, and Figure 3.6(b) shows the signal after being mixed to baseband. The next step in stage 2, is to pass the signal through the low pass filter shown in Figure 3.7. Figure 3.8(a) shows the spectrum of the signal after running though the low pass filter, and Figure 3.8(b) show the new signal. After the low pass filter the signal is downsampled by 5, where Figure 3.9(a) shows the new, downsampled spectrum, and Figure 3.9(b) shows the downsampled signal.

Stage 3 simulation

The first part of stage three is to upsample by 3, Figure 3.10(a) shows the spectrum of the upsampled signal, notice the images caused by the upsampling, Figure 3.10(b) shows the signal upsampled by 3 with 2 zeros between each sample. The signal is then filtered using the filter shown in Figure 3.11. Figure 3.12(a) shows the spectrum of the filtered signal with the upsampling images removed, and Figure 3.12(b) shows the interpolated signal. The last step of stage 3 is to downsample the signal by 2. Figure 3.13(a), shows the spectrum of the signal after the downsample by 2, and Figure 3.13(b) shows the downsampled signal.

Stage 4 simulation

The final stage will filter the signal and then downsample it to 12 MHz. Figure 3.14 shows the low pass filter used in stage 4. Figure 3.15(a) shows the spectrum of the signal after the stage 4 low pass filter, and Figure 3.15(b) shows the filtered signal. The last step of the process is to downsample by 5, Figure 3.16(a) shows the downsampled spectrum, and Figure 3.16(b) shows the downsampled signal.

3.2.2 Design Optimizations

Stage 1 optimizations

The first optimization in stage 1, is to take the mix down by 1/4 the sampling frequency and translate it through the filter, and the downsample by 8. Figure 3.17 shows how the mixer can be shifted through the filter and the downsampler, thus optimizing it to a multiply

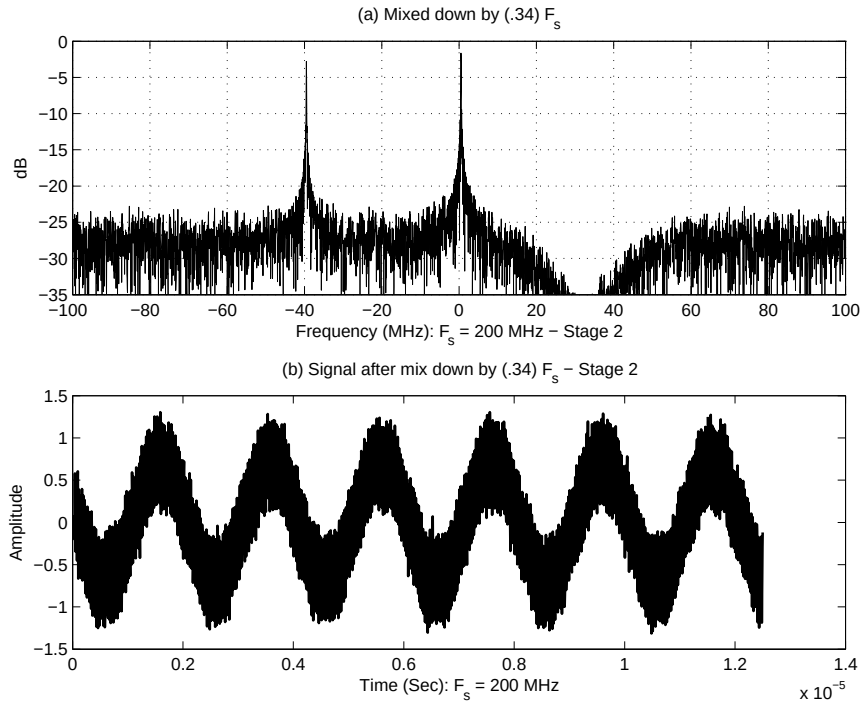

Fig. 3.6: Mix to baseband, (a) shows the spectrum of the mixed signal, and (b) shows the signal, $F_s = 200$ MHz.

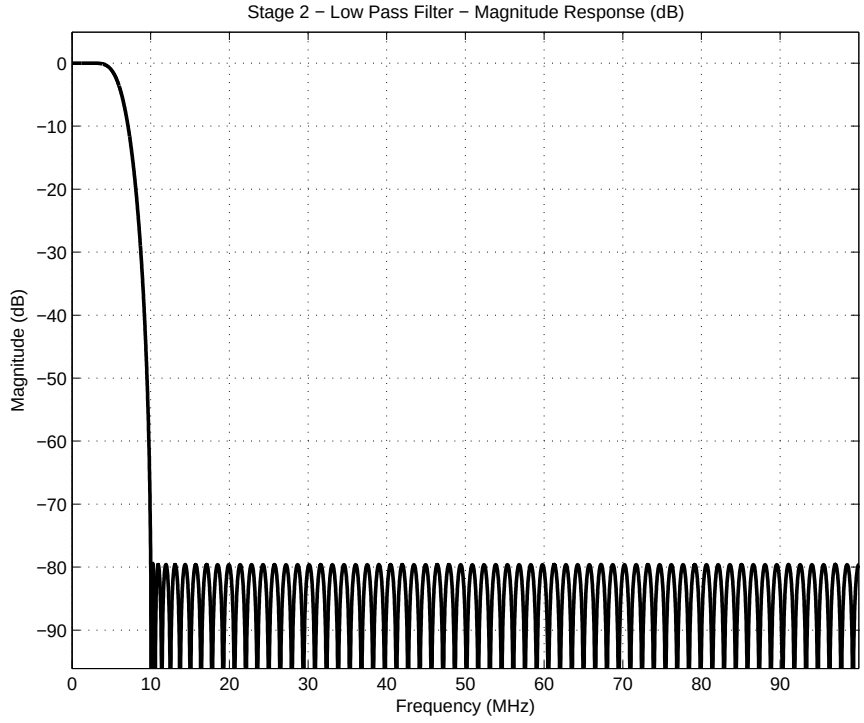

Fig. 3.7: Low pass filter used in stage 2 of the design, $F_s = 200$ MHz.

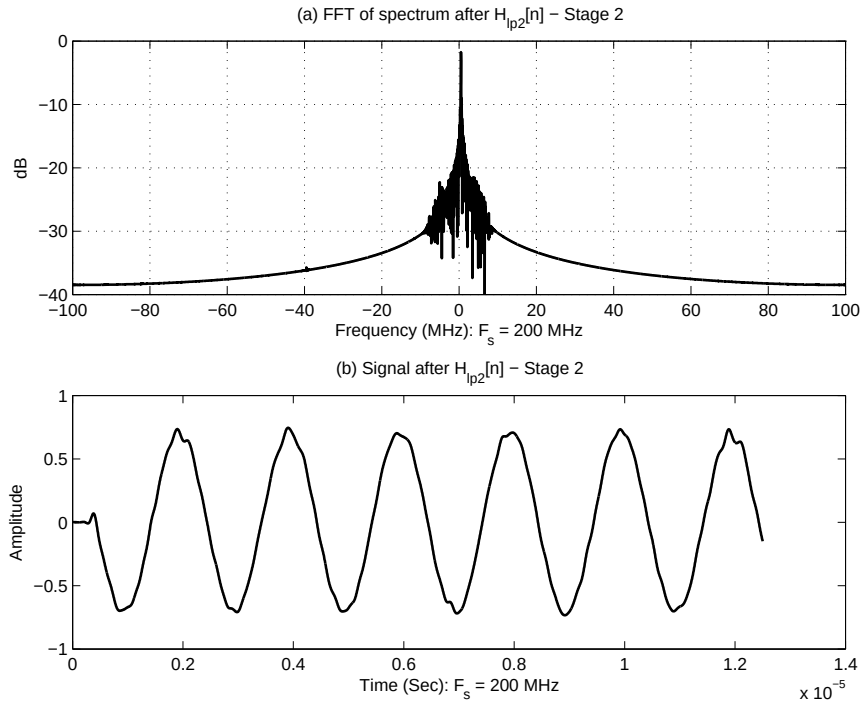

Fig. 3.8: Stage 2 low pass filtering, (a) shows the spectrum of the filtered signal, and (b) shows the filtered signal, $F_s = 200$ MHz.

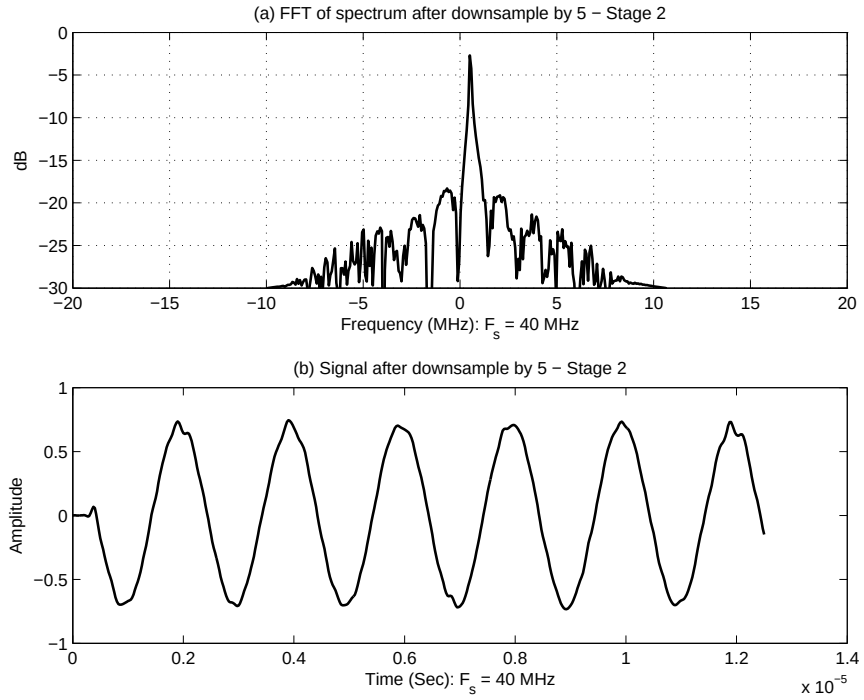

Fig. 3.9: Stage 2 downsample by 5, (a) shows the spectrum of the downsampled signal, and (b) shows the downsampled signal, $F_s = 40$ MHz.

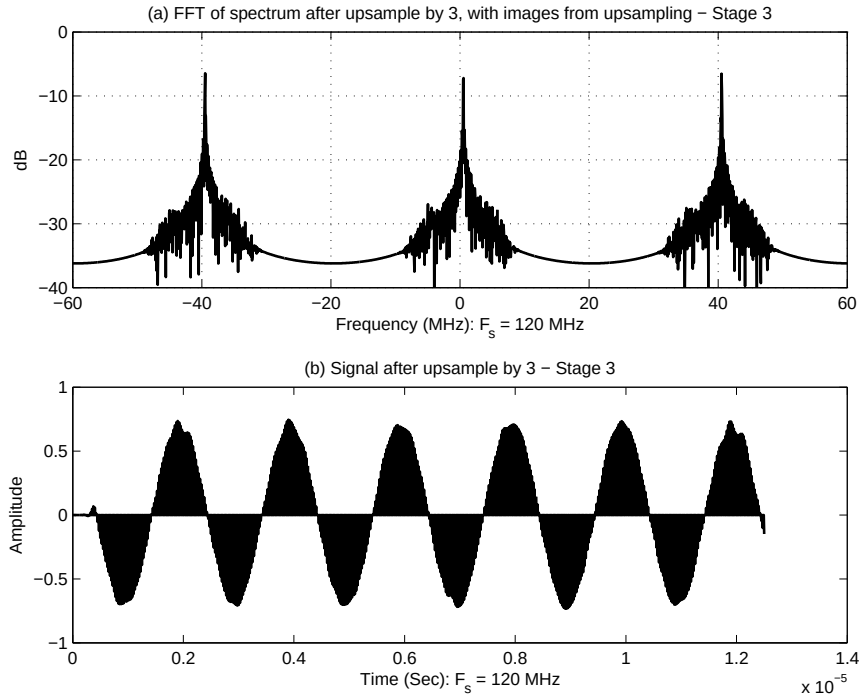

Fig. 3.10: Stage 3 upsample by 3, (a) shows the spectrum of the upsampled signal, and (b) shows the upsampled signal, $F_s = 120$ MHz.

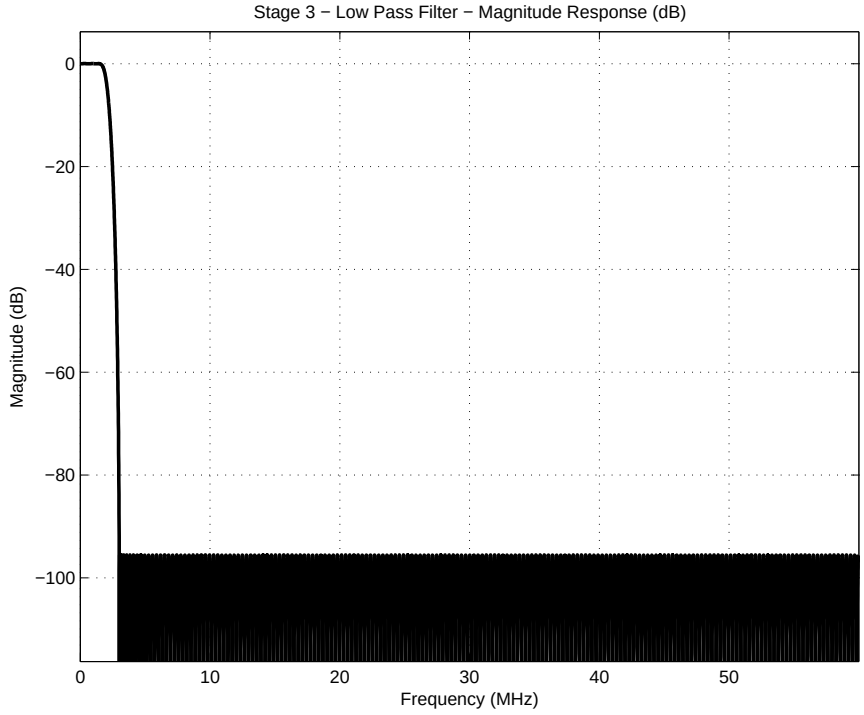

Fig. 3.11: Low pass filter used in stage 3 of the design, $F_s = 120$ MHz.

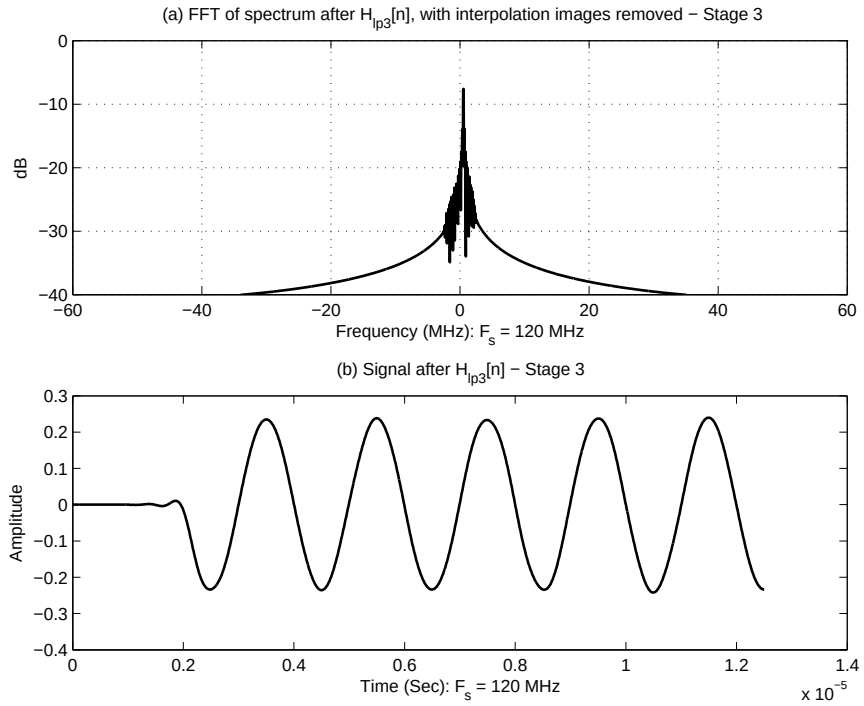

Fig. 3.12: Stage 3 low pass filtering, (a) shows the spectrum of the filtered signal, and (b) shows the filtered signal, $F_s = 120$ MHz.

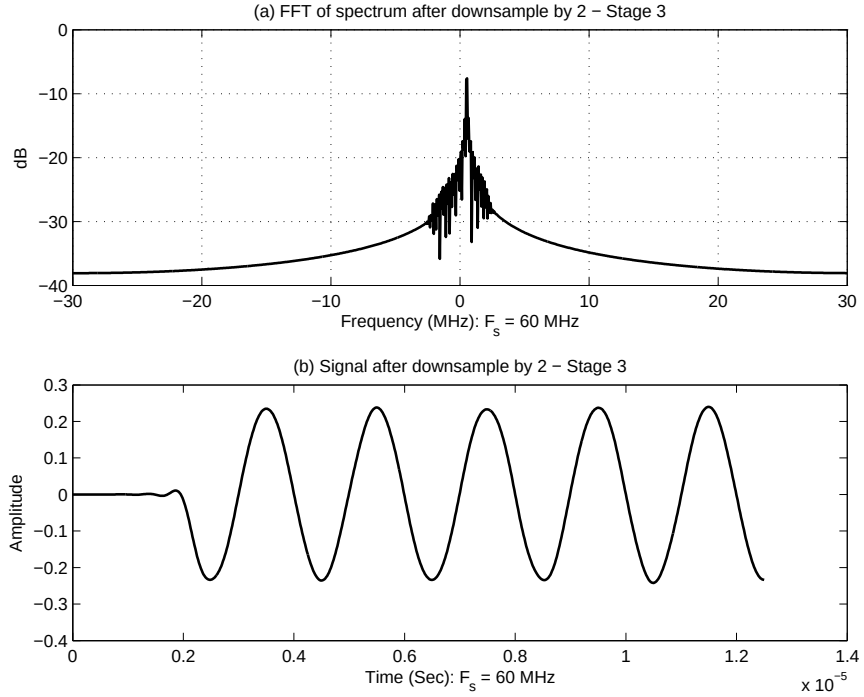

Fig. 3.13: Stage 3 downsample by 2, (a) shows the spectrum of the downsampled signal, and (b) shows the downsampled signal, $F_s = 60$ MHz.

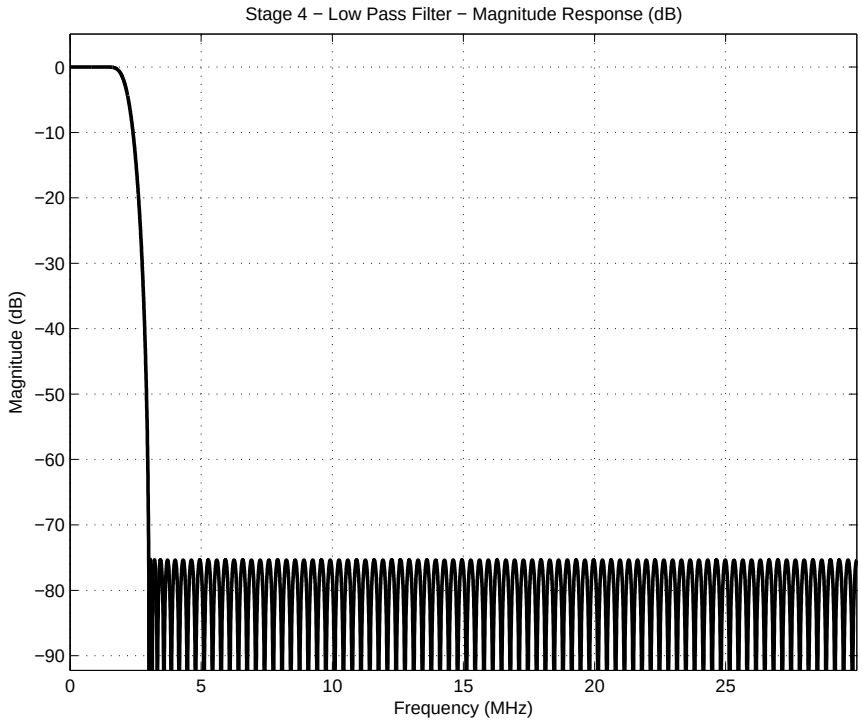

Fig. 3.14: Low pass filter used in stage 4 of the design, $F_s = 60$ MHz.

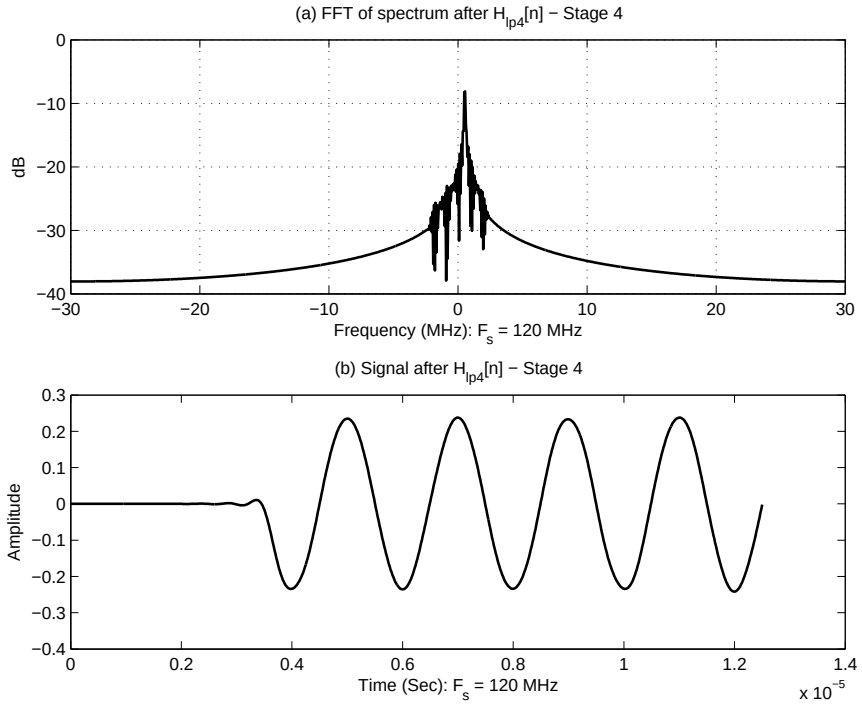

Fig. 3.15: Stage 4 low pass filtering, (a) shows the spectrum of the filtered signal, and (b) shows the filtered signal, $F_s = 60$ MHz.

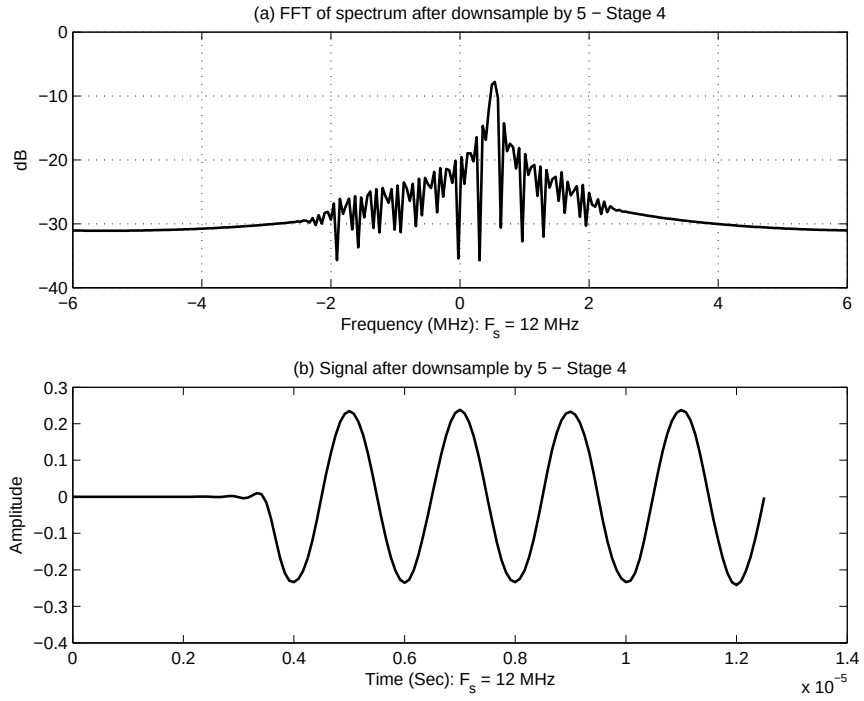

Fig. 3.16: Stage 4 downsample by 5, (a) shows the spectrum of the downsampled signal, and (b) shows the downsampled signal, $F_s = 12$ MHz.

by 1. This removes a single multiply before the filtering stage. The big change is in the low pass filter, as is shown in Figure 3.17(b), when the mixer is moved through the filter, it is then mixed up to a band pass filter, see Figure 3.18 for the in-phase branch optimized filter, and downsampling creates an image of the band where it should be at 68 MHz [5].

The next optimization for stage 1, is a polyphase decomposition of the optimized downsampler shown if Figure 3.17(d). Figure 3.19(a) shows the polyphase decomposition of the stage 1 low pass filter, and Figure 3.19(b) shows the decomposition with the noble identities applied, only for the in-phase branch of the sampling rate conversion, the same optimized structure will also apply to the quadrature-phase branch [3, 4].

Stage 2 optimization

The stage 2 optimizations consists, simply of a downsample by 5 polyphase decomposition. Figure 3.20(a) shows the polyphase filter decomposition, and Figure 3.20(b) shows the decomposition with the noble identities applied [3, 4].

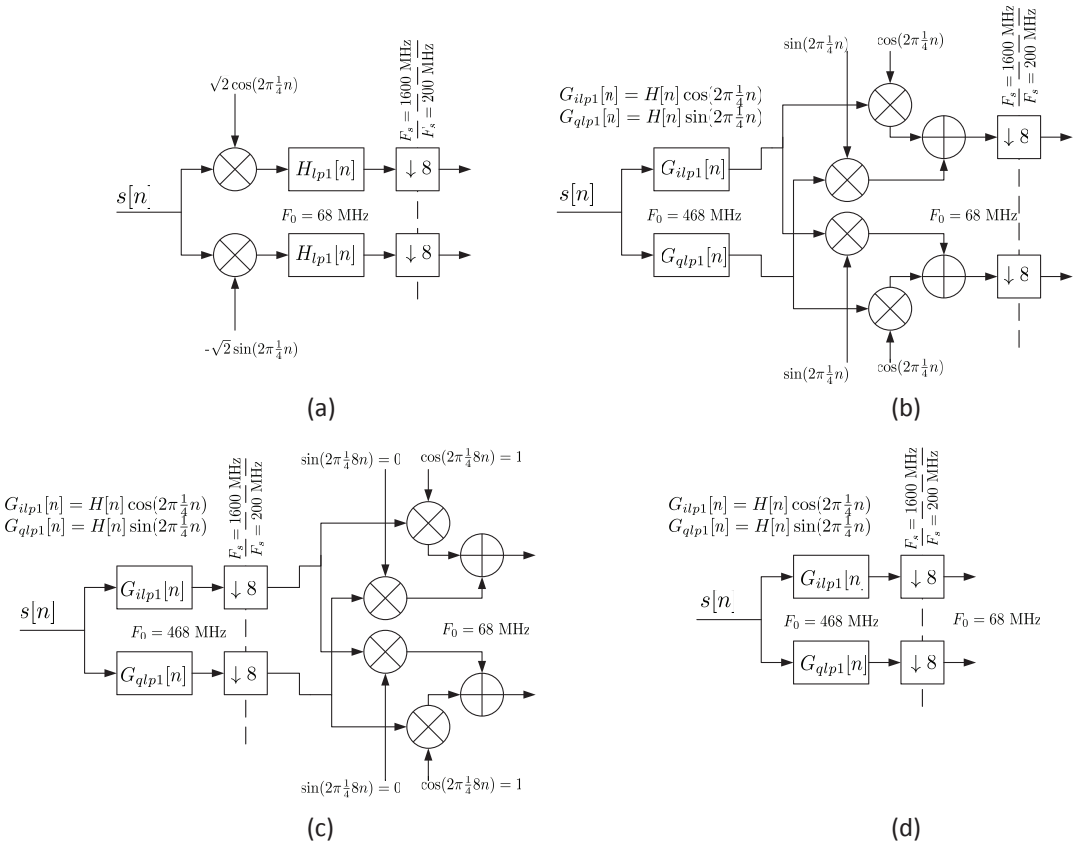

Fig. 3.17: Stage 1 optimizations, (a) is the original design, (b) is the original design with the transformed low pass filter, (c) shifts the downsampler through the complex multiply, and (d) is the optimized design.

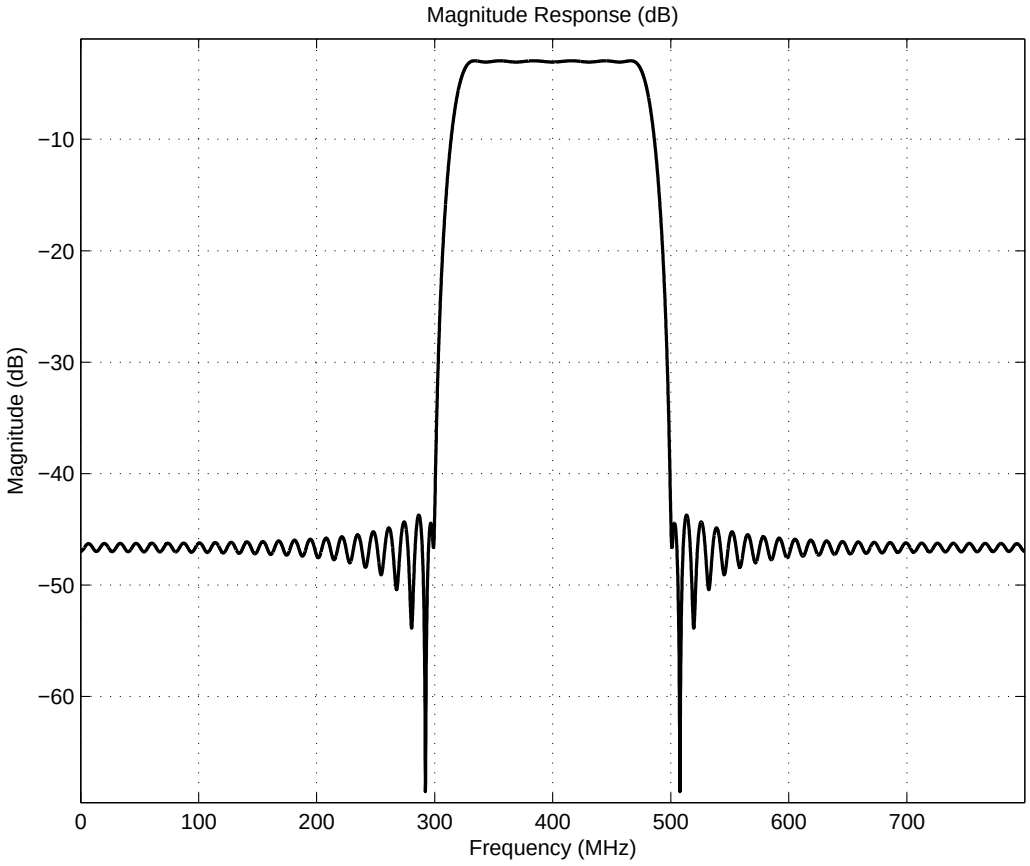

Fig. 3.18: Optimized version of the stage 1 low pass filter.

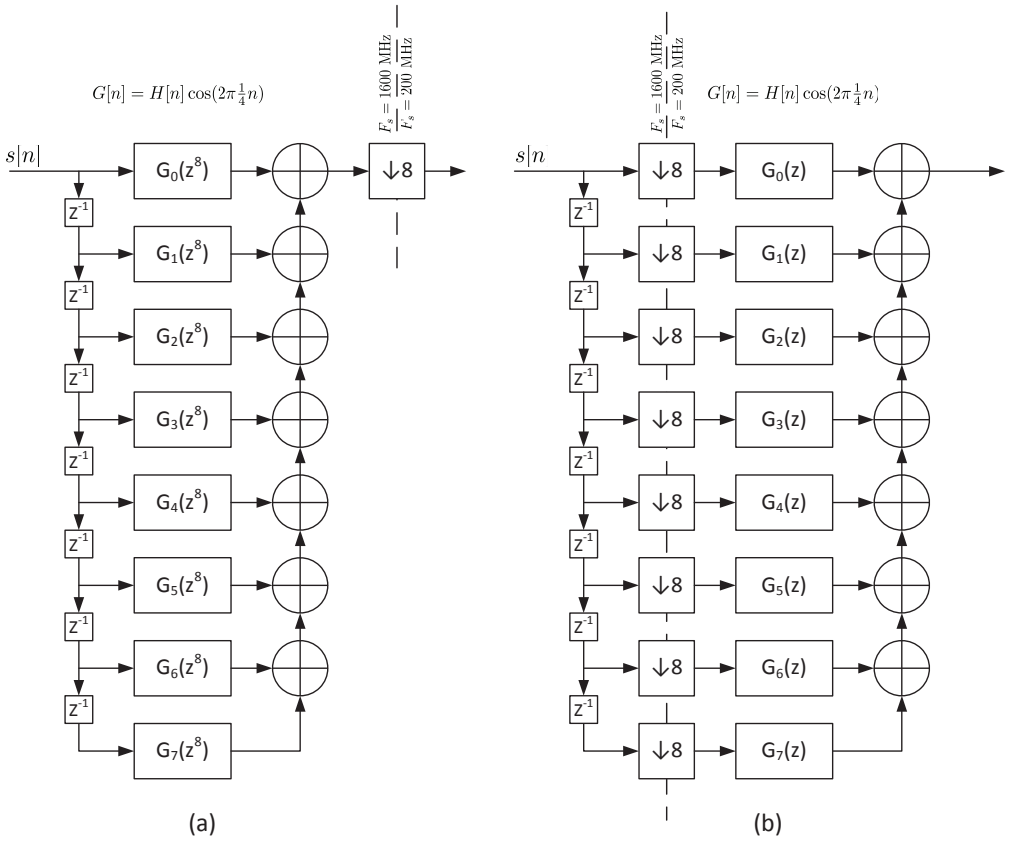

Fig. 3.19: Stage 1 optimization, with (a) the polyphase decomposition of the low pass filter, and (b) with the noble identities applied.

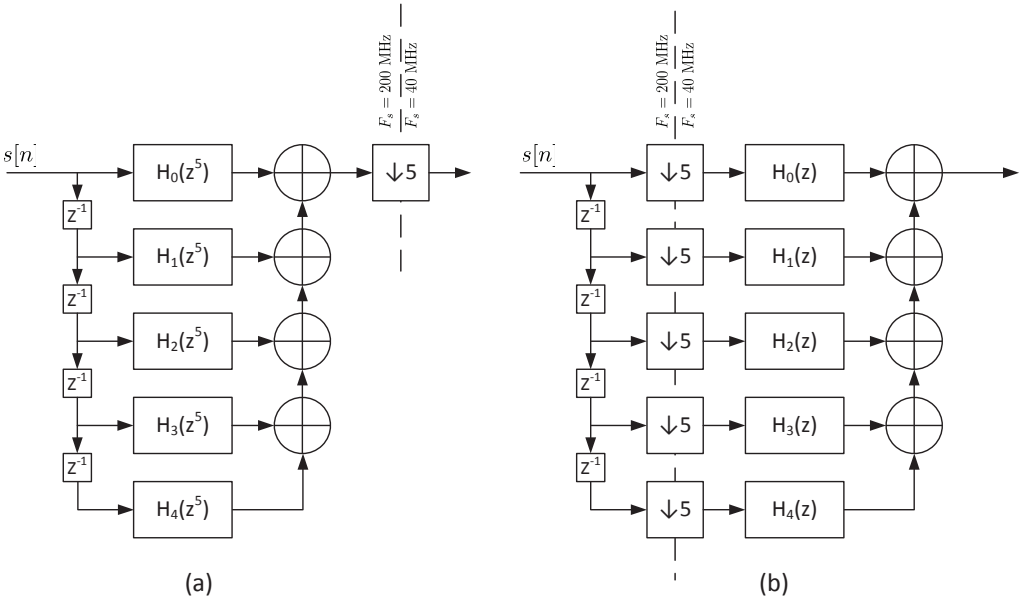

Fig. 3.20: Stage 2 optimization, with (a) the polyphase decomposition of the low pass filter, and (b) with the noble identities applied.

Stage 3 optimization

The stage 3 optimizations consists of a fractional rate optimization, with an upsample by 3 and a downsample by 2 structure. Figure 3.21 shows the polyphase filter decomposition with the noble identities applied [3, 4].

Stage 4 optimization

The stage 4 optimizations consist, simply of a downsample by 5 polyphase decomposition. Figure 3.22(a) shows the polyphase filter decomposition, and Figure 3.22(b) shows the decomposition with the noble identities applied [3, 4].

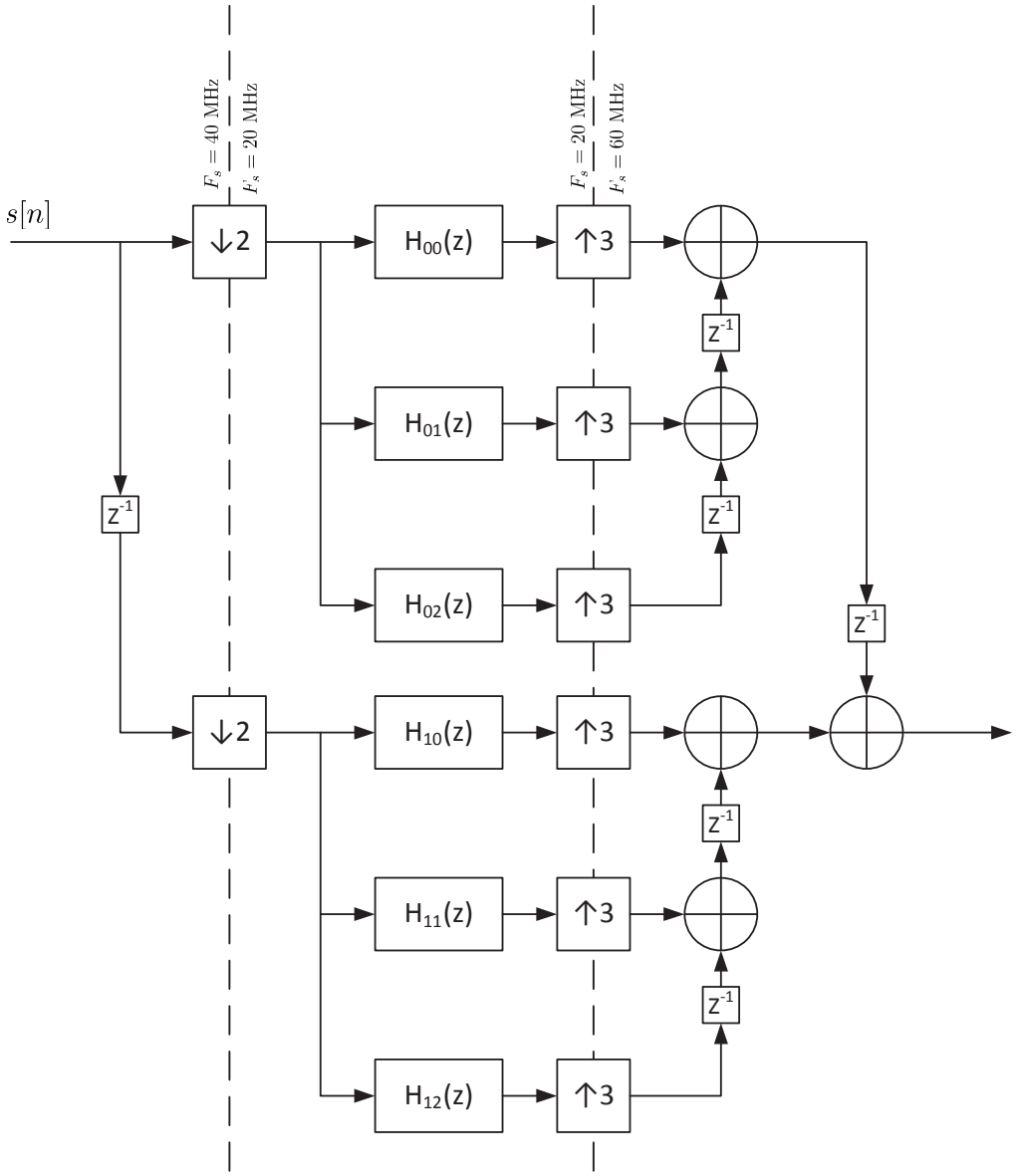

Fig. 3.21: Stage 3 optimization, with the polyphase decomposition of the low pass filter, and with the noble identities applied.

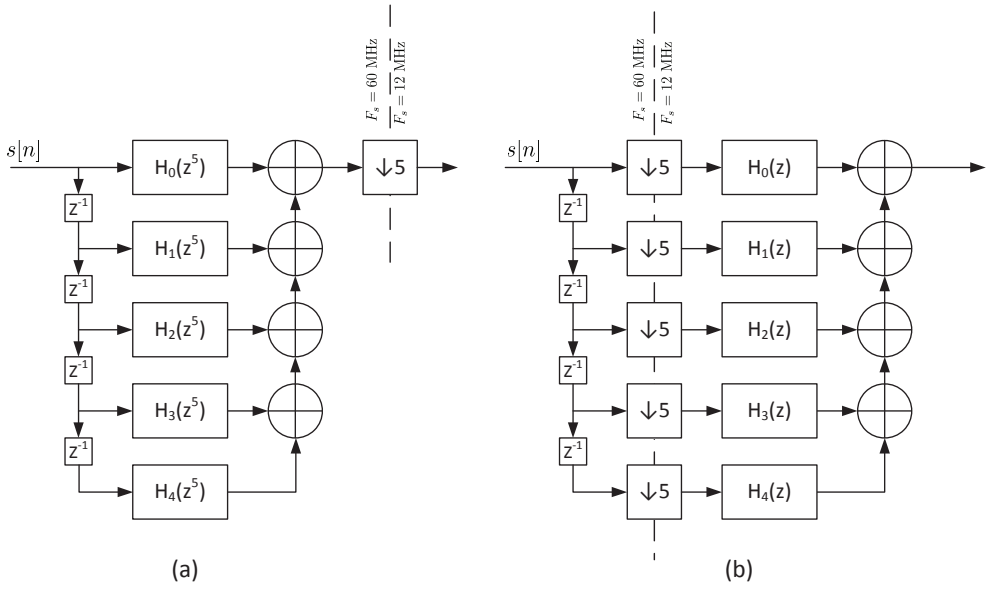

Fig. 3.22: Stage 4 optimization, with (a) the polyphase decomposition of the low pass filter, and (b) with the noble identities applied.

Chapter 4

Hardware solution

4.1 Hardware description

The hardware used for this project consists of a desktop personal computer (PC), with a digitizer card that connects to the PC using a peripheral component interconnect express (PCIe) bus for high-speed data transfer to the PC. The digitizer card used is innovative integrations X6-GSPS, where GSPS mean giga-sample per second (GSPS). As is implied from the name of the digitizer card, the X6-GSPS is capable of sampling up to 1.8 GSPS on two channels or 3.6 GSPS on a single channel. The following sections describe in more detail the hardware being used and the development tools used for developing digital signal processing (DSP) algorithms in firmware [6].

4.1.1 Innovative Integrations X6-GSPS

The innovative Integrations X6-GSPS is a high-speed digitizer card with a high-performance Xilinx Virtex-6 FPGA. The X6-GSPS also has 4 GB of onboard DRAM, and a PCIe interface capable of enumerating up to 8 PCIe lanes for high-speed data transfer. The X6-GSPS connects to the PC though the XMC to PCIe adapter board. Figure 4.1 shows the X6-GSPS board with the Xilinx Virtex-6 FPGA [6–9].

XMC to PCIe adapter board

The XMC to PCIe adapter board physically connects the X6-GSPS to the PC mother board PCIe slots. It also has a fan to help with thermal management of the FPGA on the X6-GSPS. The adapter board also provides power to the X6-GSPS, and is directly connected to the PC power supply. Figure 4.2 shows the the XMC to PCIe adapter board that enables the X6-GSPS to connect to a PCIe interface for high-speed data transfer.

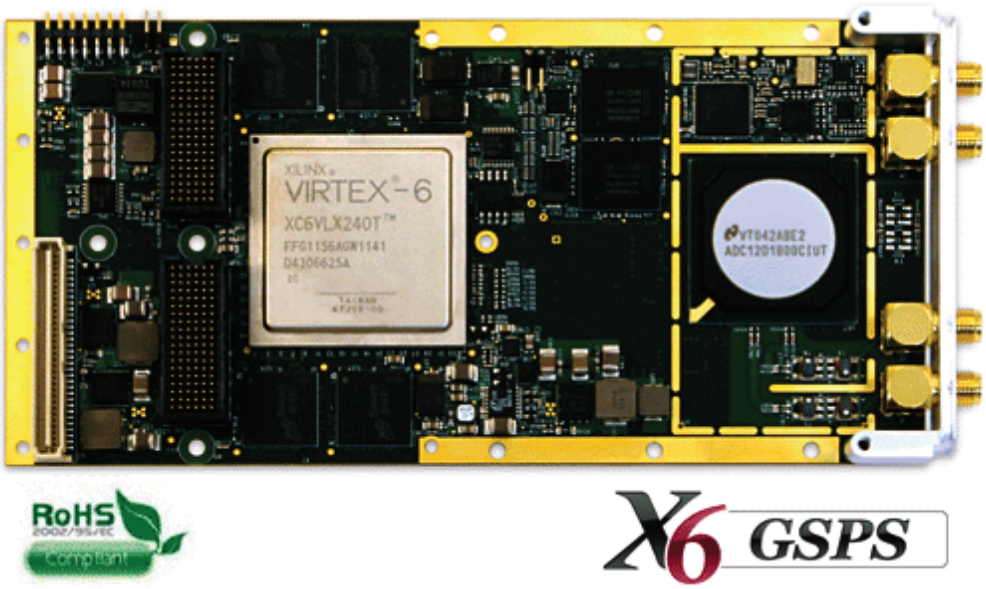

Fig. 4.1: Image showing the X6-GSPS board.

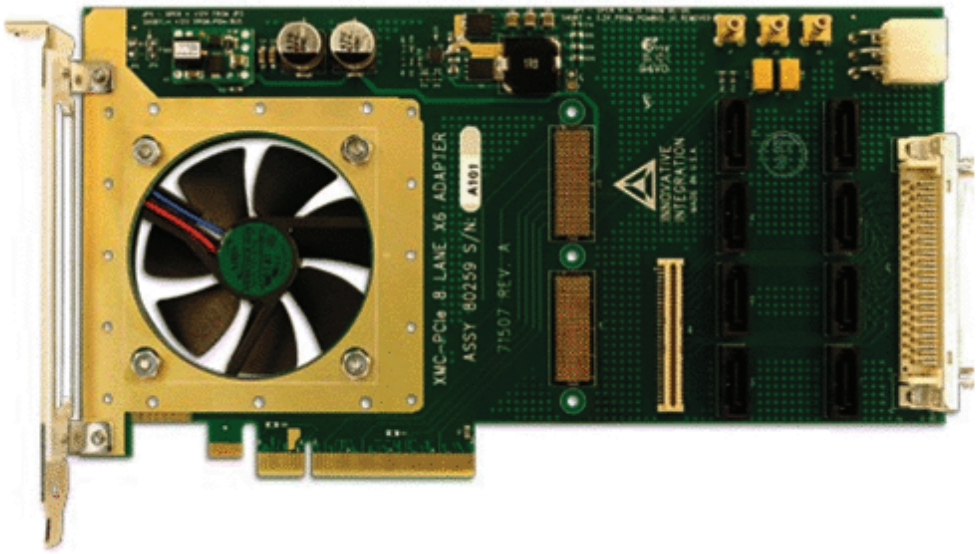

Fig. 4.2: Image showing the XMC to PCIe adapter board.

Xilinx Virtex-6 FPGA

Xilinx Virtex 6 FPGAs are high-performance, programmable logic, used in DSP and embedded systems that require high reconfigurability and real-time operations. The Virtex 6 family of FPGAs have many feature that are desirable in high-speed, high-performance DSP applications, such as DSP48E1 slices, high-speed GTX transceivers, and PCIe interfaces. The FPGA selected for this design is the Virtex 5 XC6VSX475T, which has 476,160 logic cells, 74,400 slices, 2,016 DSP48E1 slices, 2 PCIe interface blocks, and 38,304 Kb of block RAMs [10].

GSPS analog to digital converter

The ADC used in the design is an 12-bit 1.6 GSPS dual channel ADC, or 3.6 GSPS single channel ADC. The ADC is designed for a wide variety of applications, such as, radar, data acquisition, communications, and software defined radio. Figure 4.1, also shows the ADC on the X6-GSPS board [9].

4.2 Firmware development

Firmware development for the X6-GSPS is the process of implementing the DSP design with the provided supporting firmware provided by Innovative Integrations. The requires understanding the framework logic firmware, and how to interface custom designs to the framework logic firmware. Custom designs can be created using Matlab Sumulink and Xilinx System Generator, or can be written using VHSIC hardware description language (VHDL), where VHSIC stands for very high speed integrated circuits(VHSIC). The VHDL option is not the preferred method for development because of its long development and simulation time. However the VHDL option allows for greater control of timing and interfacing. The System Generator option is preferred because of the graphical method for implementing system designs. Simulation using Xilinx System Generator and Matlabs Simulink is also more useful and allows for better visualization of signal, especially discretized signals being processed. The following sections will discuss the firmware development tools, processes, and implementations strategies [8].

4.2.1 Framework logic

The framework logic firmware is a compilation of VHDL files, intellectual property (IP) cores, and binary netlist files or NGC files. These files are combined into an Xilinx ISE project that contains all of the required interfacing logic for the ADC, PCIe, and RAM. The files also contain data plane logic that allows the FPGA to get data from the ADC, send and receive data from the host PC, and transfer data within the FPGA for digital signal processing. To interface a custom design to the framework logic, an NGC netlist file, created by Xilinx System Generator in Matlab Simulink using Innovative Integrations board support package (BSP), will simply be added to the Xilinx ISE project. The steps for doing this will be discussed in Section 4.2.4, the board support package will be discussed in Section 4.2.2, and the Matlab Simulink and Xilinx System Generator will be discussed in Section 4.2.3 [7, 8].

4.2.2 X6 board support package

Innovative Integrations has created a board support package that is to be used with Matlab Simulink and Xilinx System Generator, for creating custom DSP designs that are easily plugged into the Xilinx ISE project. The BSP contains all of the input and output ports needed to interface with the Xilinx ISE project. Innovative Integrations has provided a default project file called "*matlab_project.mdl*", which contains the required blocks for integrating a design into the Xilinx ISE project. Figure 4.3 shows the default project with the System Generator token required for netlist generation [7].

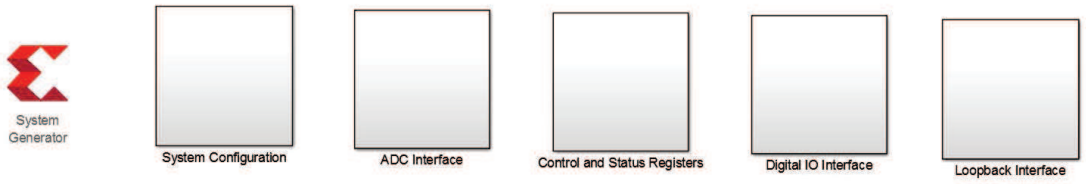

Fig. 4.3: Image showing top level of the board support package with the Xilinx System Generator Token.

System configuration

The system configuration block shown in Figure 4.3 contains ports for the resets and clock-locked signals. Figure 4.4 shows the contents of the system configuration block, and the Innovative Integration sub-block, System Configuration, is a wrapper block that contains Xilinx gateway block that act as input and output port for the component instantiation. Figure 4.5 shows the contents of the sub-blocks that contain the gateways for the input and output ports, in this example, the sub-block being shown is the ADC0_input block [7].

ADC interface

The ADC interface block shown in Figure 4.3, contains the default configuration of the ADC0 and ADC1 interfaces. This default connection allows for direct pass-through of data from the ADC to the host PC. Any DSP algorithms would be implemented between the ADC0_input and the ADC0_output, the naming of these blocks is not intuitive, the main thing to understand is that the block that is labeled input, is input to the ADC data processing, and output, is output from the ADC data processing to the host PC. Figure 4.6 shows the default connections for the ADC interface. The data from the ADC0_input is formated as 8, 16-bit samples from the ADC that are valid when the system clock has a rising edge and the ADC0_dvld signal is high. For more information see Innovative Integrations documentation on its BSP [7].

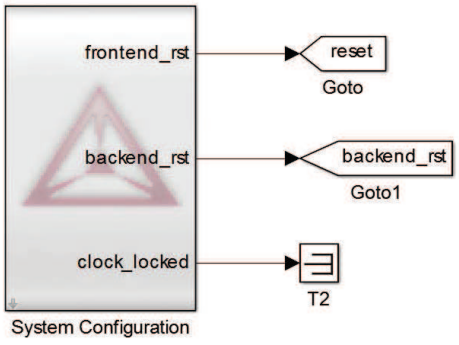

Fig. 4.4: Image showing the contents of the system configuration block.

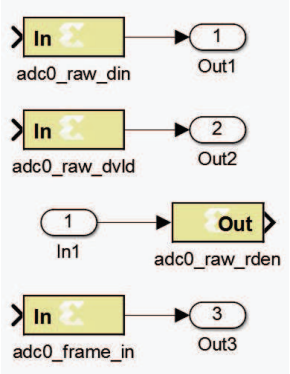

Fig. 4.5: Image showing the contents of the port wrapper block ADC0_input block.

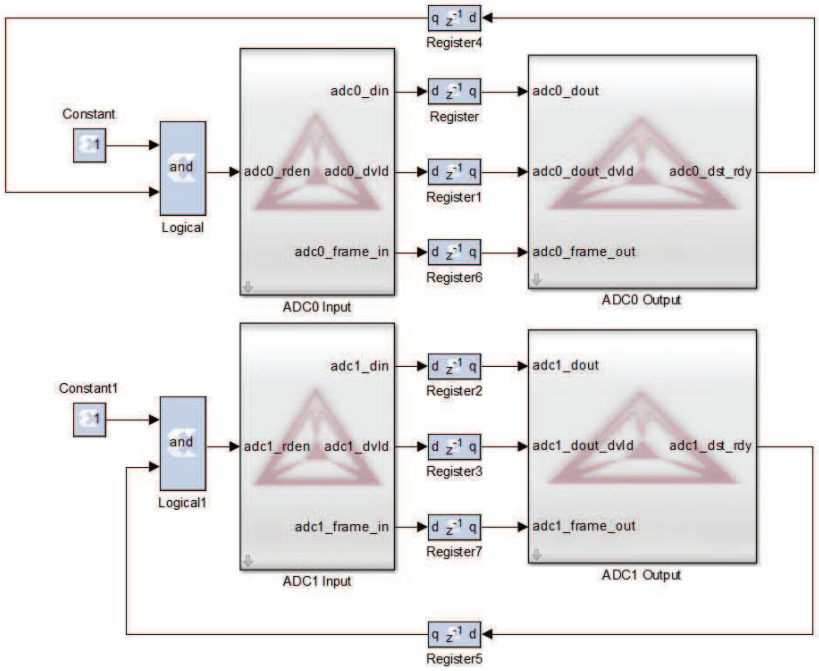

Fig. 4.6: Image showing the contents of the ADC interface block, with default connections.

Control and status registers

The control and status registers are used as a way for the user to configure, and verify register contents from the host PC. This is useful in more complicated designs that have different states that need to controlled by the user. For this design at this stage, these registers were not used. Figure 4.7 shows the contents of the control and status register block shown in figure 4.3 [7].

Digital IO interface

The digital input/output (DIO) interface is used to read and write to physical pins on the FPGA that are designated for general purpose input/output. Figure 4.8 shows the contents of the DIO interface block shown in Figure 4.3 [7].

Loopback interface

The loopback interface is used to pass data from the host PC for data processing and then back to the host PC. It operates in a similar manner to the the ADC interface, the only difference is the data source is the host PC. Figure 4.9 shows the contents of the loopback interface block, as shown in Figure 4.3 with default connections. This can be useful if there is a lot of data that needs to be processed from the host PC in real-time [7].

4.2.3 Simulink and Xilinx System Generator

Simulink paired with Xilinx System Generator is a graphical design environment that can be used to create complex designs that can be directly compiled to VHDL or NGC netlist files. The files created from this development environment can be directly instantiated in a larger design, and can even be a full design by itself. The use of this development environment was largely determined by Innovative Integrations BSP discussed in Section 4.2.2. When Xilinx System Generator is opened on the development PC, it opens a session of Matlab that can be used to develop hardware synthesizeable designs. Figure 4.10 show the Simulink Library Browser with some of the many available Xilinx block for use in the Simulink environment [11].

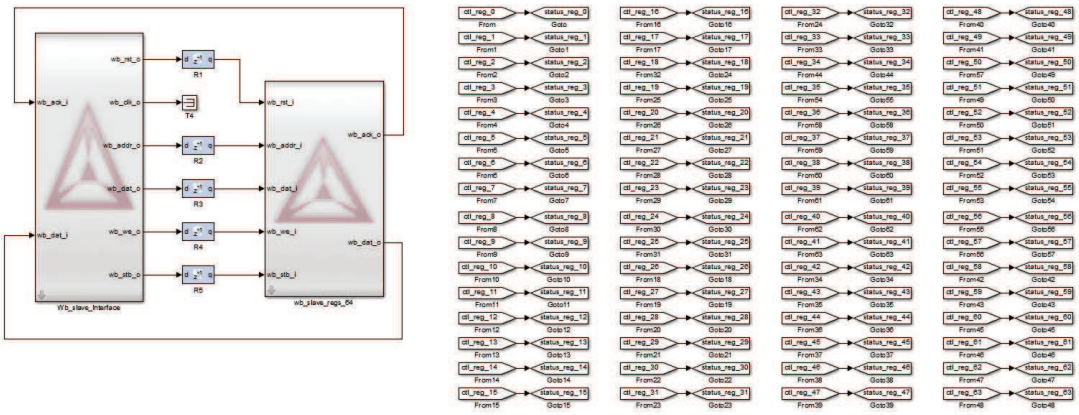

Fig. 4.7: Image showing the contents of the control and status registers block.

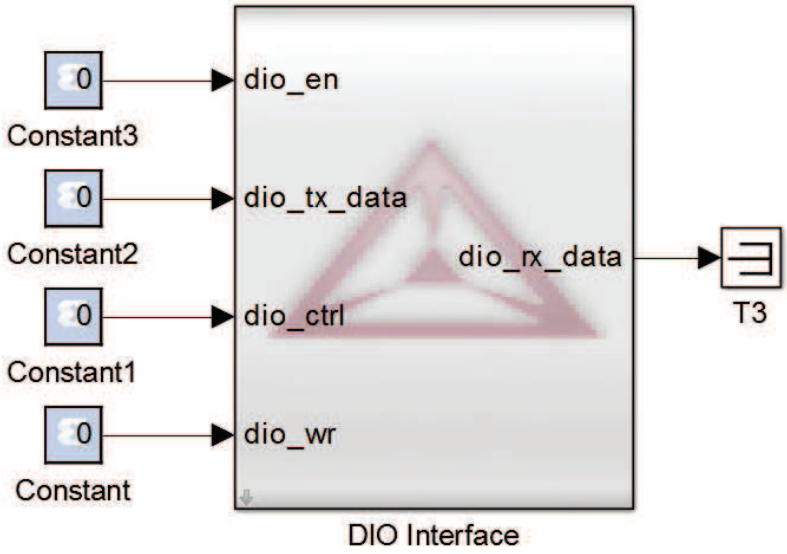

Fig. 4.8: Image showing the contents of the DIO interface block.

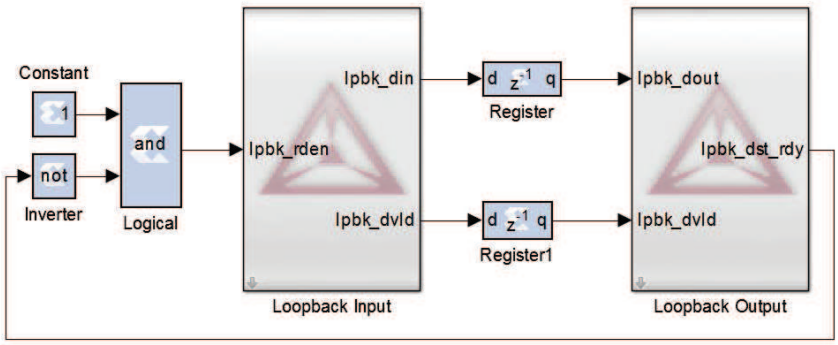

Fig. 4.9: Image showing the contents of the loopback interface block.

System Implementation

Implementing a design in Xilinx System Generator is a relatively simple task, however some care must be taken to ensure proper timing of signals to and from Xilinx library blocks. Most blocks placed in a design allow the user to access properties, by double-clicking on the block, that give the user control over data-types, and data-widths. It is recommended to read documents on Xilinx System Generator, and Innovative integrations BSP manual, and to work through the example projects provided by Innovative Integration [7, 8, 11].

System Verification and simulation

The Matlab Simulink development environment allows for quick system verification and simulation, which is important because it can take hours to compile a design and test on the hardware. There are two main methods for verifying a design in the Matlab Simulink environment. The first is to connect it to a spectrum analyzer scope, or time-axis scope, and the second is to use the Wavescope tool as a virtual logic analyzer. The Wavescope tool is very useful, as it allows you to view signals and buses, as either discrete signals or to view its analog representation. Figure 4.11 shows the output of a simulation result using the Wavescope tool. If a frequency domain analysis of the signal is needed then a spectrum analyzer scope can be used to verify the spectrum of a signal. Figure 4.12 shows the output of a spectrum analyzer scope, this updates in near real-time giving nearly immediate feedback on the system, where the Wavescope tool only shows the signals at the

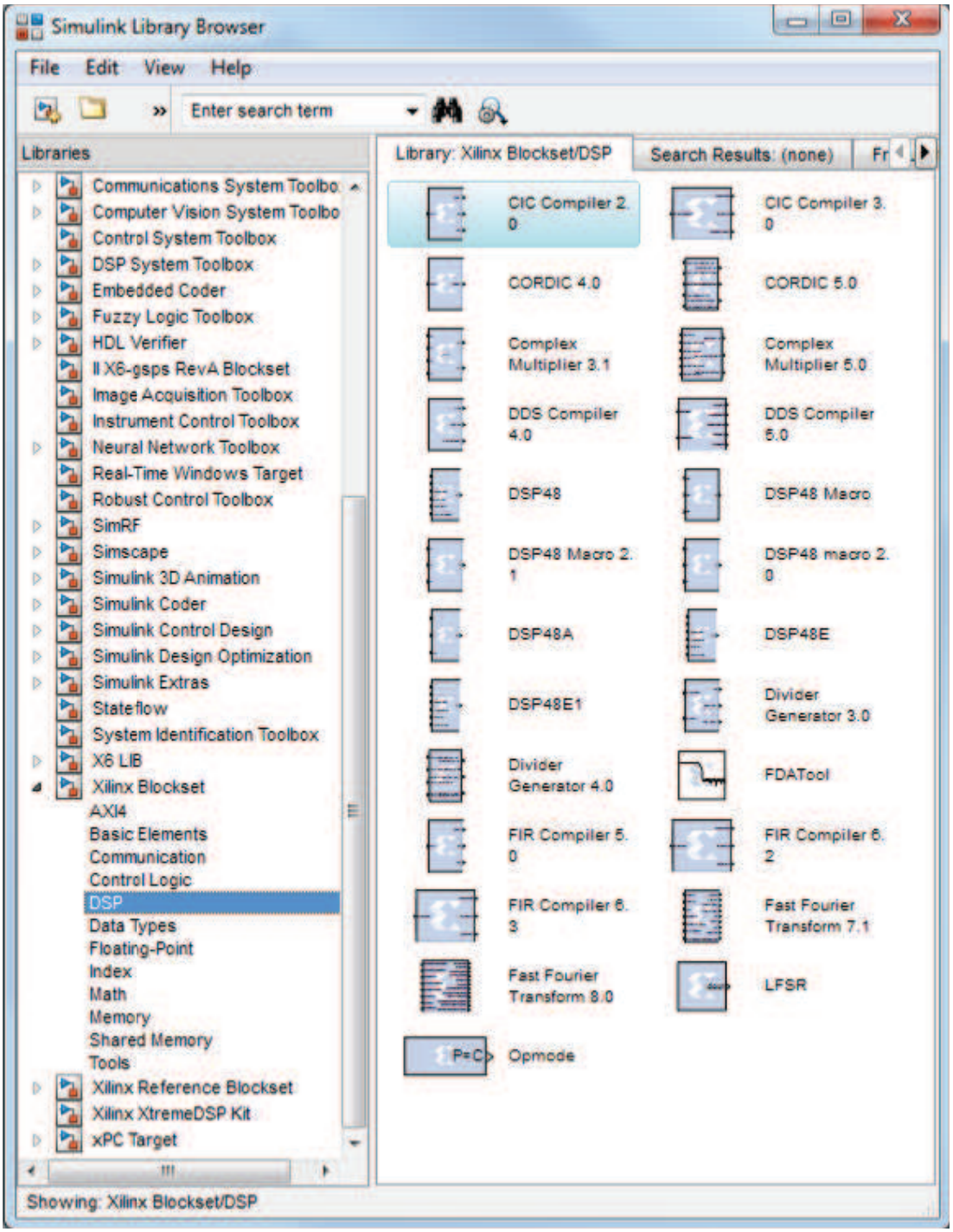

Fig. 4.10: Image showing the Simulink library browser.

end of a simulation [11].

4.2.4 Xilinx ISE

Xilinx ISE is a development environment designed for Xilinx products, and has all the tools required to develop and compile a design into a bit-file for hardware. The development environment uses project files to track all VHDL code files, IP cores, and netlists. When an object is instantiated in a file the Xilinx ISE development environment detects that, and uses its project browser to list the inherent hierarchy of the project.

Innovative Integration ISE project

Innovative Integration provides and ISE project as part of the framework logic BSP, which has all the required logic for interfacing with all of the peripherals on the X6-GSPS. The benefit of using the framework logic ISE project is that all of the foundation work required to get a project up and working is done, and effort can be focused on the DSP algorithm implementation.

Adding custom design to ISE project

Adding a custom design to the ISE project is fairly straightforward. In the Matlab Simulink project file, select the system generator token, and double click on the token, this will open a dialog box. From the dialog box, click generate, and the system will compile and create a NGC netlist file that can then be imported into ISE. The final step is to add the newly generated NGC netlist file to the ISE project. The project is now ready to compile and generate a programming file.

Generating Programming file

Generating a programming file is a time consuming process, for this project compilation took roughly 45-50 minutes to complete. It is recommended to thoroughly simulate the design before compilation for more efficient design flow. To generate a programming file, there are multiple steps in the compilation process that must be taken, synthesis, translate,

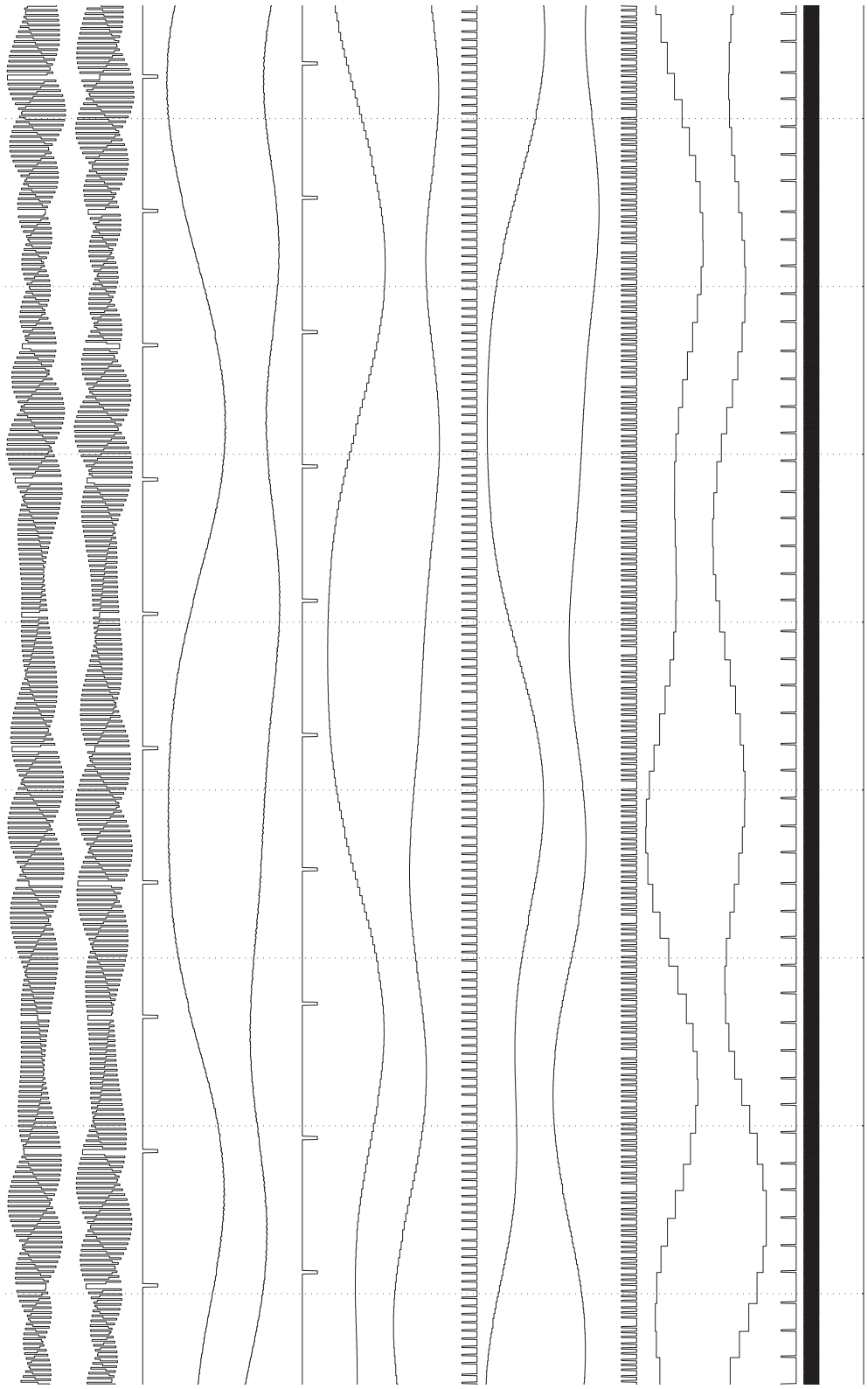

Fig. 4.11: Simulation results using Wavescope.

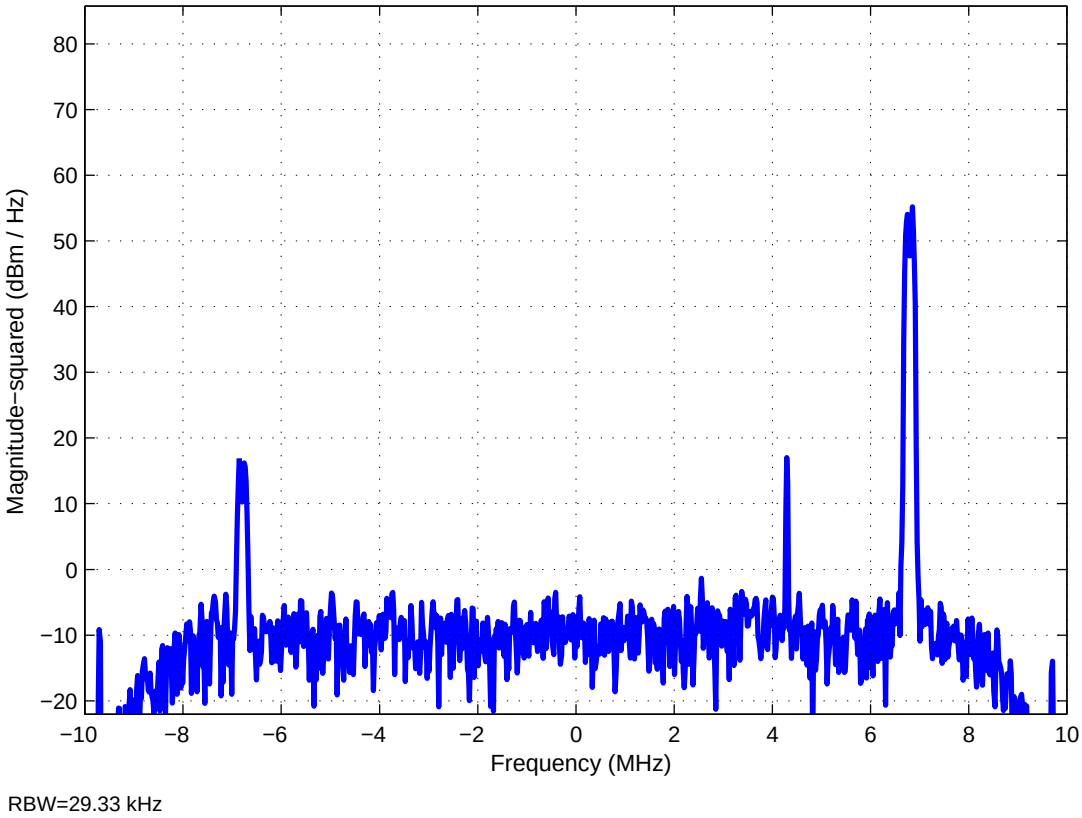

Fig. 4.12: Simulation results using the spectrum analyzer scope.

map, place and route, with the last step being to generate the programming file. While each step can be completed individually, in sequential order, the simplest approach is to double-click the generate programming file option in the processes menu of the ISE interface. Once the compilation is complete and the programming file is generated, a green check mark icon will appear next to the generate programming file option.

Correcting timing errors

Timing errors occur when logic set-up and hold times are not met, this can be caused by large levels of combinatorial logic, or signal that travel long relative distances without registers or synchronous logic. Timing errors are more prevalent the faster the system is running. There are a few ways to detect and correct timing errors, the first is the clock timing report. By looking at the report, any logic that has a failed timing score, will be shown with a red x. Look carefully at the signals, and where they are coming from and where they are going to find where the timing is not being met. Once the error is located, return to the design and add registers to pipeline the design. Another tool that is useful is Xilinx PlanAhead, which is a graphical tool that shows the FPGA logic utilizations and the connections. Looking at the connections of problem signals will show if the signal is traveling large distances in the FPGA without being registered. Again adding registers to the problem signal will usually solve the issue. There are also numerous resources on-line that will help to locate and resolve timing score issues. It is useful to note that even with timing errors the compilation process will still generate a valid programming file, a failing timing constraint means that at the specified clock rate the system will not work as expected. Lowering the clock rate will allow the system to function properly, however it will not be capable of the performance, and speed it was designed for.

Programming FPGA with Xilinx Impact and Xilinx Platform II JTAG cable

Once a programming bit file has been generated, the FPGA is ready to be programmed. The first step is to connect the Xilinx Platform II JTAG cable to the FPGA board. Follow the instructions in the Innovative Integration documentation to correctly connect the JTAG

cable, as the connection is not intuitive. Once the cable is correctly connected, open the Xilinx iMPACT utility to connect to and program the FPGA. Again, follow the instructions in the Innovative Integration documentation to correctly program the FPGA. The programming process takes about a minute, once the FPGA is correctly programmed restart the computer to re-enumerate the FPGA on the PCIe bus. Do not power off the computer or the FPGA will default to its original image, and the FPGA will have to be reprogrammed. Once the FPGA is correctly programmed and the computer restarted, the design can be tested [7, 8].

Chapter 5

Hardware implementation, firmware simulations, and real-time test results

5.1 Firmware design, firmware simulation, and test strategies

The sampling rate conversion system for the real-time ground station receiver was designed and simulated in Chapter 3, and Chapter 4 discusses the hardware being used for the actual implementation of the design. This chapter will discuss the implementation, simulation and real-time test results. Just as the design was segmented into four stages, in the design and simulation shown in Chapter 3, so is the actual implementation. The design was implemented starting with the first stage, and before moving on to a subsequent stage, each hardware implementation of that stage was thoroughly simulated. Once the current stage has been thoroughly simulated, the firmware was compiled into a bit-stream for programming the FPGA, and a real-time test was performed. Once the firmware was verified in hardware, the next stage of the design was added to the previous stage, and the whole system, up to that point, is simulated and tested again.

The simulation of each stage made used of time-domain scopes, spectrum analyzer scopes, and the Wavescope tool discussed in Section 4.2.3. These tools allowed for the viewing of signals in both digital and analog forms to verify the inputs and outputs of the system. Results of the simulation will be shown in the sections discussing the implementation and simulation of each stage [11].

The test signal generated for the real-time simulation was generated by a signal generator. The signal used for this test was a 468.5 MHz sine wave, with an amplitude of -60 dBm. At each stage of the design it will be shown that the signal entering the system was correctly processed, and that the results of the real-time simulation, and tests, match the

results of the theoretical simulations.

5.2 Test signal generation for simulation

The test signal generated for the real-time simulation in the Matlab Simulink development environment, were created to simulate sine waves in and out of the band of interest, and to simulate an OQPSK modulated signal with sine waves out of the band of interest. Each test signal was also introduced with band-limited white noise. The signals are then sampled by a simulated ADC and the samples are packed into a packet of 8, 16-bit signed integer values, this is done using an asymmetric FIFO, where the input rate is 1600 MHz with a 16 bit input, and the output rate is 200 MHz with a 128 bit output. This simulates the actual operation of the ADC0_input block from the BSP discussed in Section 4.2.2. Figure 5.1 shows the test signal generation module created for this project, and Figure 5.2 shows the module created to generate test OQPSK signal data. The Wavescope output of the simulated OQPSK signal is shown in Figure 5.3, and the spectrum of the simulated signal is shown in Figure 5.4.

5.3 System implementation overview

The sampling rate conversion system has four stages that are implemented, and also has an input FIFO and output FIFO. The system is running with a system clock speed of 205 MHz, this is to ensure that data is read from that ADC FIFO faster than data is put into the ADC FIFO, which prevents FIFO overflow and data loss. A data stream is created by connecting the valid_out signal port to the next valid_in port signal, this ensures that data is processed as it is received, in real-time. All synchronous elements in the design are registered to prevent timing issues, and they are all reset, to clear the processing pipeline at the beginning of each processing session.

The implementation is designed using the default *matlab_project.mdl* file so that it is compatible with the framework logic ISE project. Figure 5.5 shows the top-level implementation of the design with the simulated reset and ADC0_input block tied to the main real-time ground station (RTGS) processing block. The next level of the design, shown in

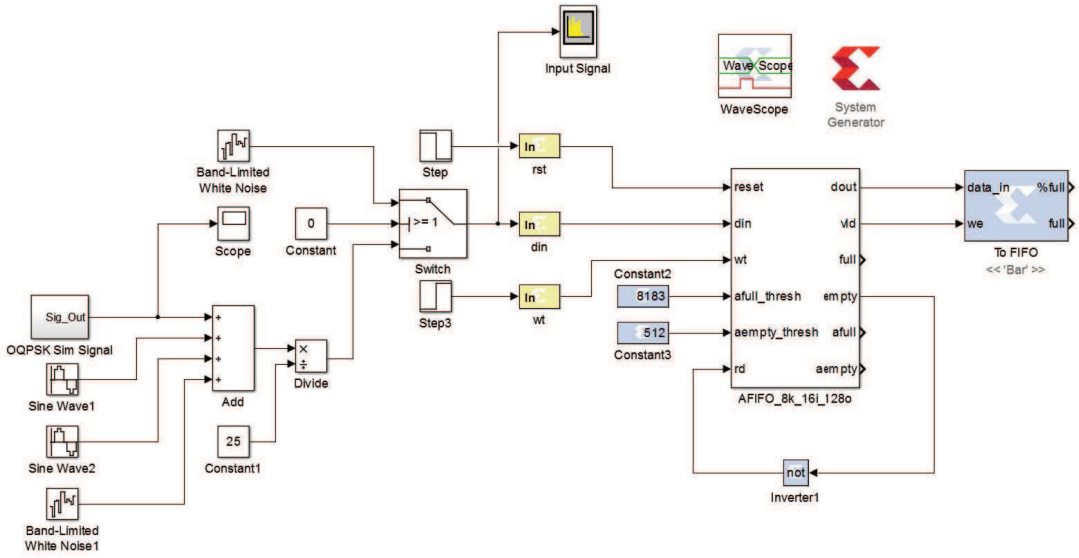

Fig. 5.1: Blocks used to generate test signal for simulation.

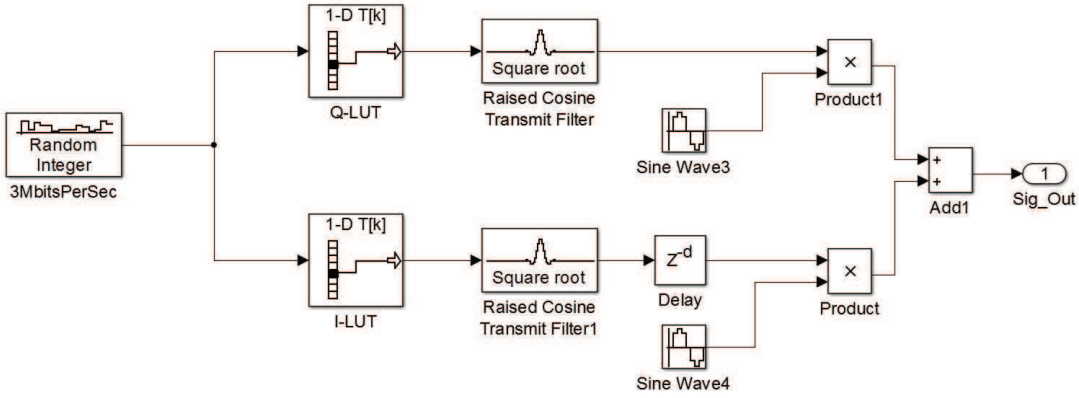

Fig. 5.2: Blocks used to generate OQPSK test signal for simulation.

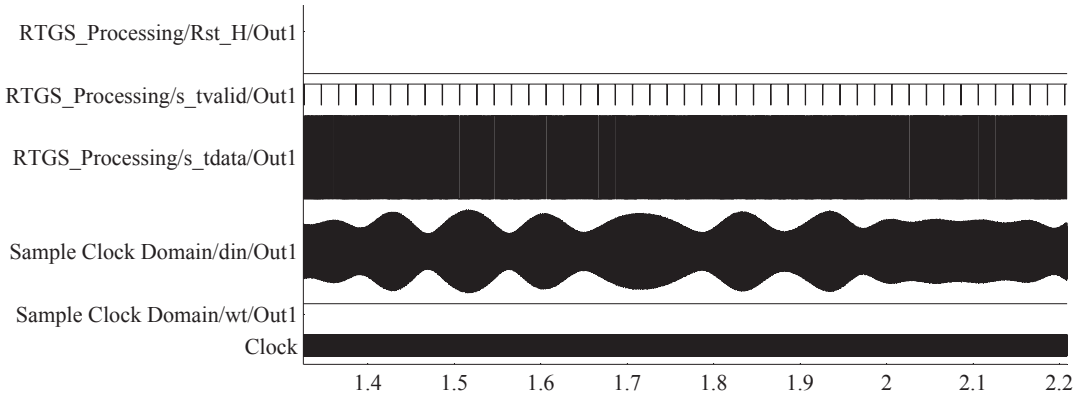

Fig. 5.3: Output of the Wavescope tool in the test signal generation module.

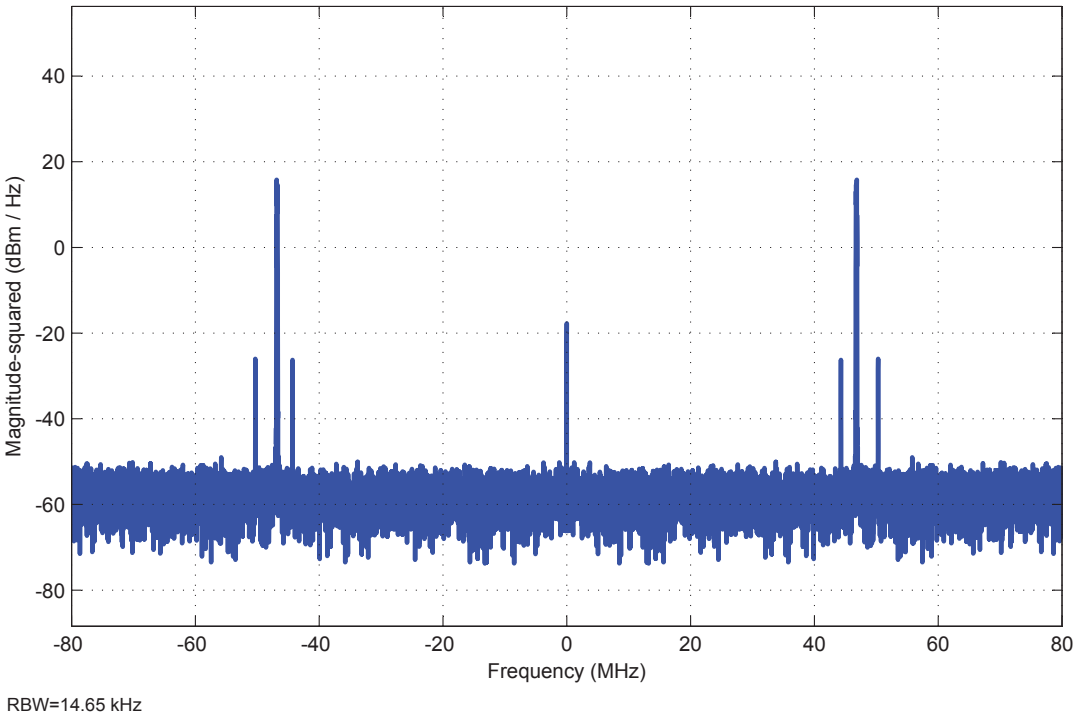

Fig. 5.4: Spectrum of test signal.

Figure 5.6, is the sampling rate conversion (SRC) processing block connecting to the output FIFO block, Figure 5.7 shows the contents of the output FIFO block. Figure 5.8 shows the contents of the SRC block with the input FIFO, and each stage of the SRC processing. The following sections will discuss the implementation and design of each stage, as well as simulation, and real-time testing.

5.4 Stage 1 implementation and simulation

Stage 1 was designed and implemented to take 128 bit samples from the ADC0_input at a rate of 200 MHz, with samples being valid when the system clock has a rising edge and the data valid signal is logic high. This requires that the data be immediately downsampled by 8 using a polyphase filter structure. Figure 5.9 shows the stage 1 implementation, with the polyphase down by 8 structure, and Figure 5.10 shows the contents of the sum8 block. In this stage we expect that the frequency of interest, 468 MHz, would be mixed down by 1/4 the sampling frequency, or 400 MHz, to 68 MHz, then filtered and downsampled

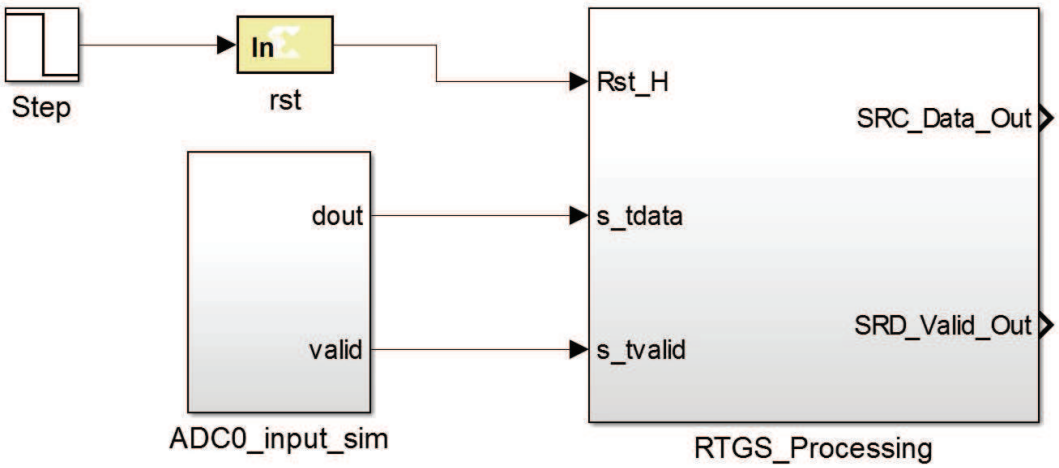

Fig. 5.5: Top level design with simulated inputs.

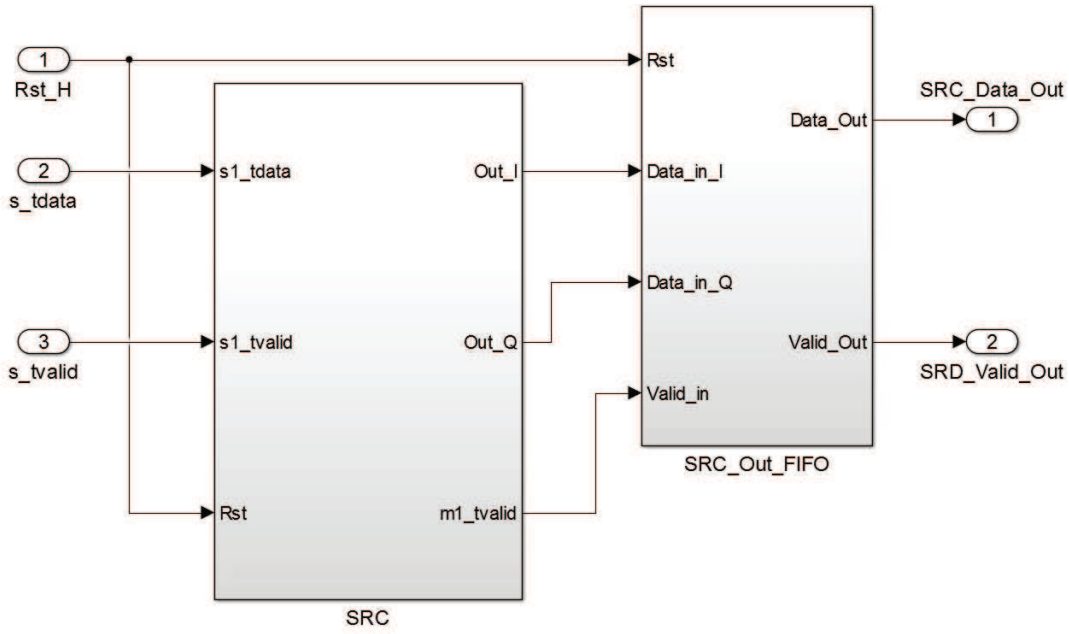

Fig. 5.6: Contents of the RTGS_processing block.

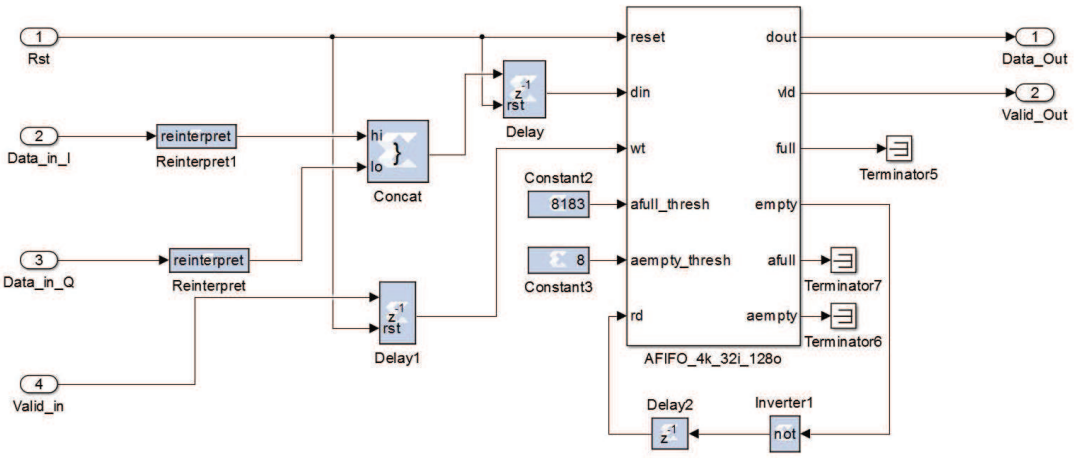

Fig. 5.7: Output FIFO taking the in-phase and quadrature-phase signals and concatenating them for input into the output FIFO.

by 8. Figure 5.11 shows the spectrum of the processed signal, and Figure 5.12 shows the Wavescope result of the simulation. Each Figure shows that stage 1 is working as expected.

5.5 Stage 1 real-time test results

The following shows the real-time test results of stage 1 of the system. The test signal used to perform the real-time test is a 468.5 MHz sine wave, and after processing the signal should be mixed down to 68.5 MHz and have a sampling rate of 200 MHz. Figure 5.13 shows a fast Fourier transform (FFT) of the results of the stage 1 real-time test, and Figure 5.14 shows a spectrogram of the results. It can be seen from the Figures that the results of the real-time test match the desired results.

5.6 Stage 2 implementation and simulation

Stage 2 was designed and implemented to take the outputs of stage 1, mix the signal to baseband, and downsample by 5. Stage 2 is separated into 2 separate modules, the first consists of the mix down to baseband, and the second is the downsample by 5 filter structure. The mix down to baseband block contains a direct digital synthesizer (DDS) and a complex multiply, as is shown in Figure 5.15. Figure 5.16 shows the stage 2 implementation of

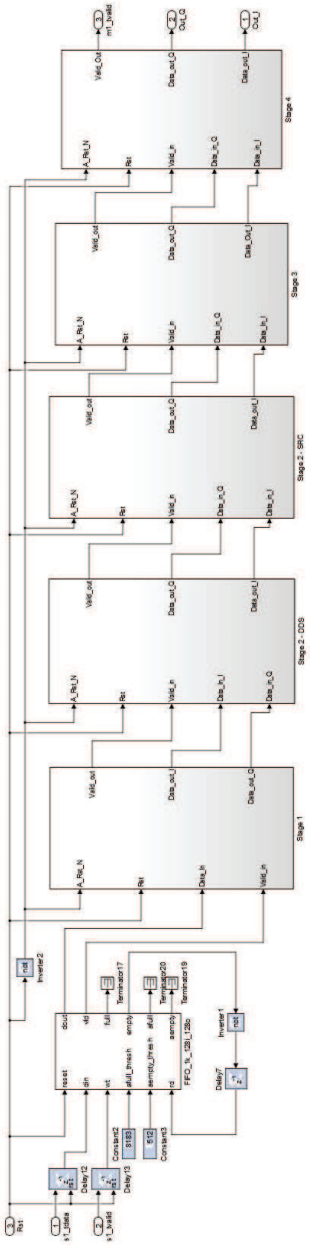

Fig. 5.8: Contents of the SRC block, showing the input FIFO, and the stages of the sampling rate conversion processing.

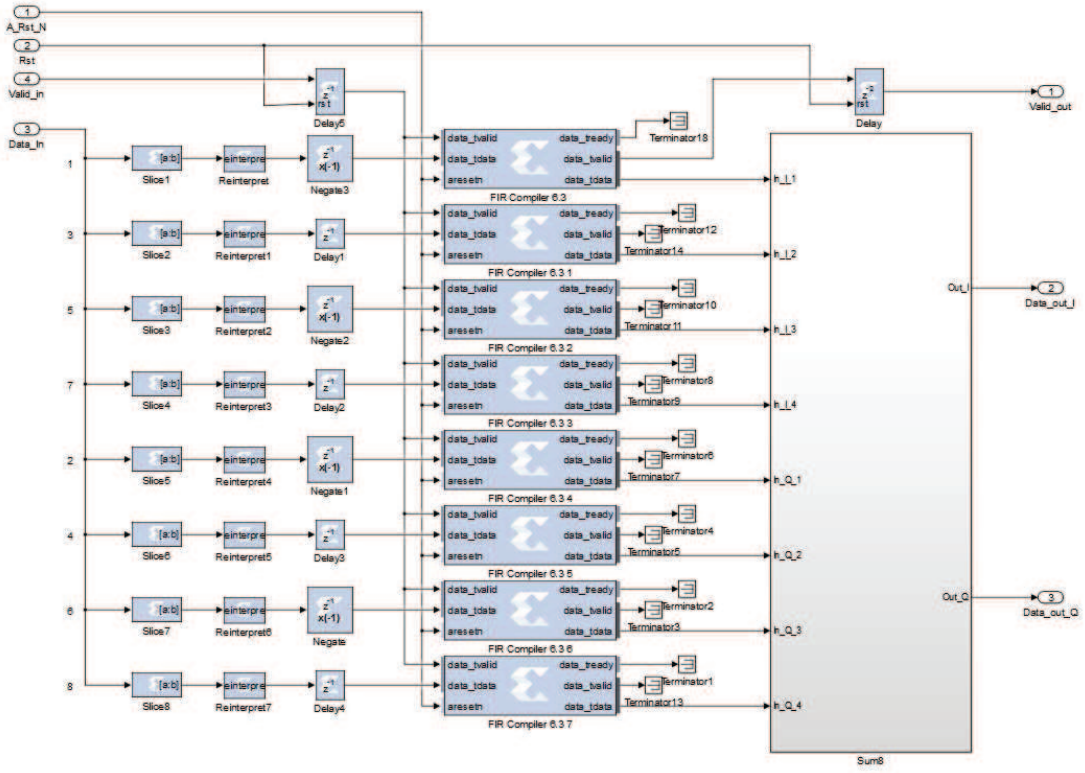

Fig. 5.9: Implementation of stage 1 with a polyphase down by 8 filter structure.

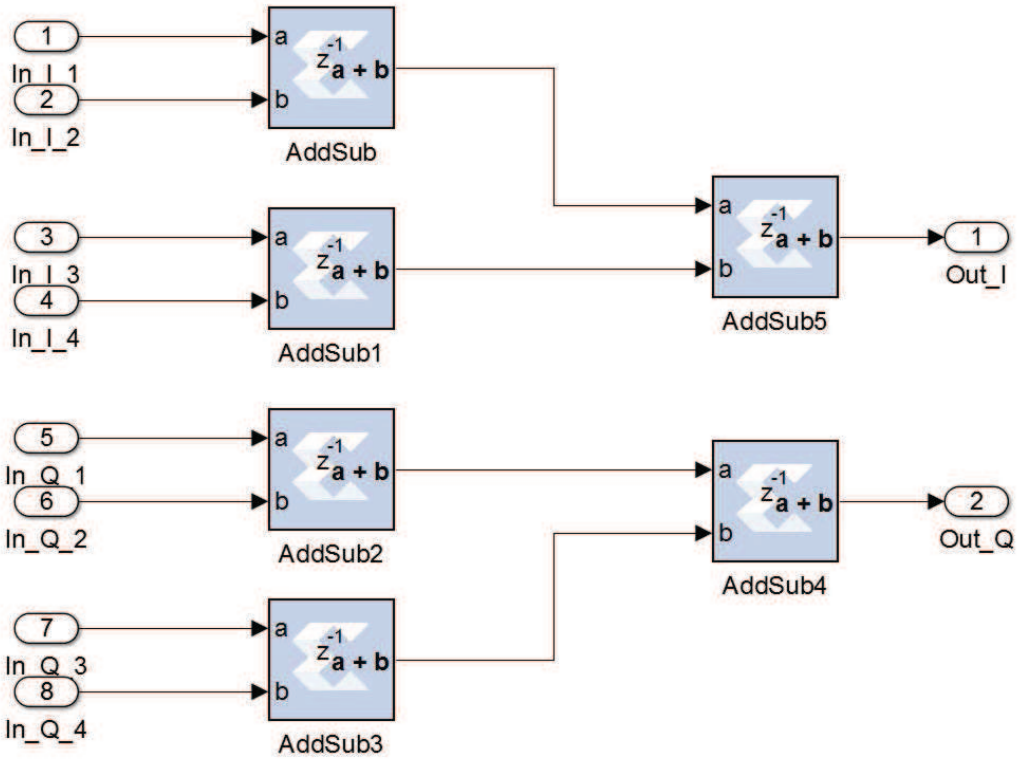

Fig. 5.10: Contents of the sum8 block contained in stage 1.

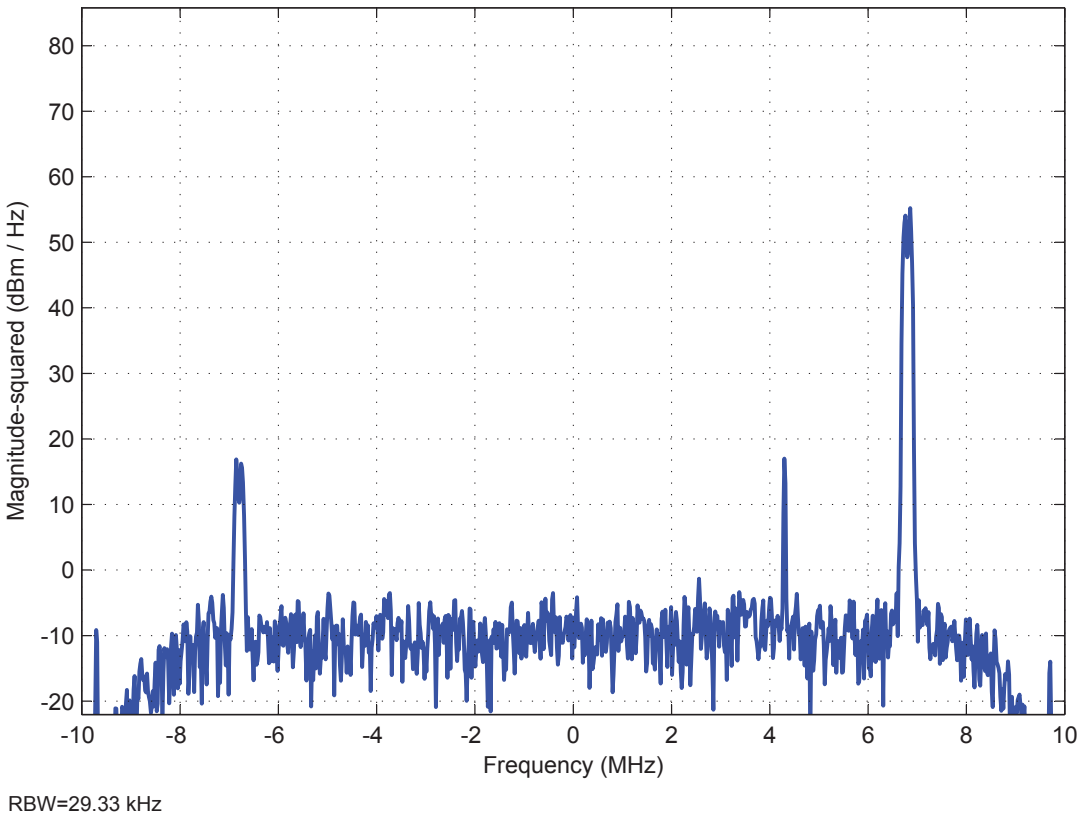

Fig. 5.11: Spectrum of the signal at the output of stage 1.

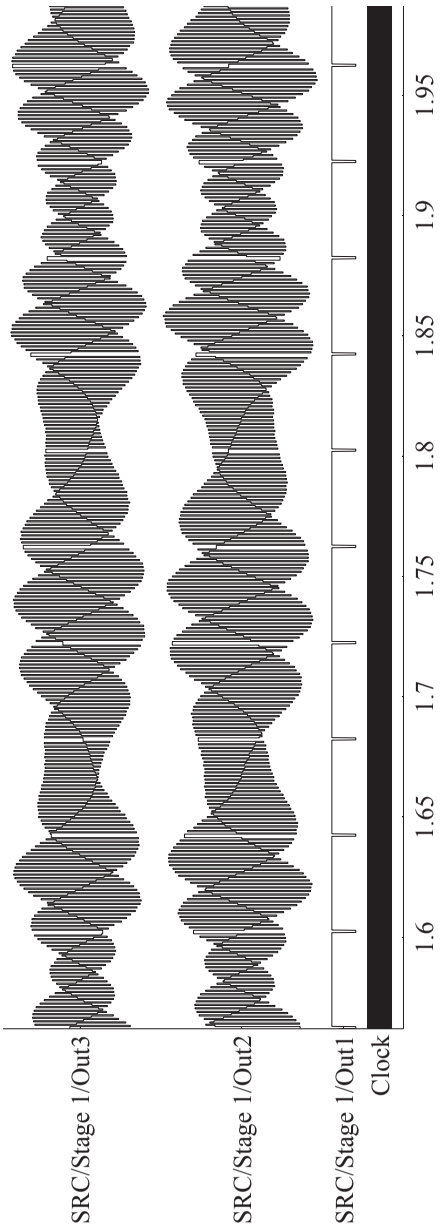

Fig. 5.12: Output of the Wavescope tool for stage1.

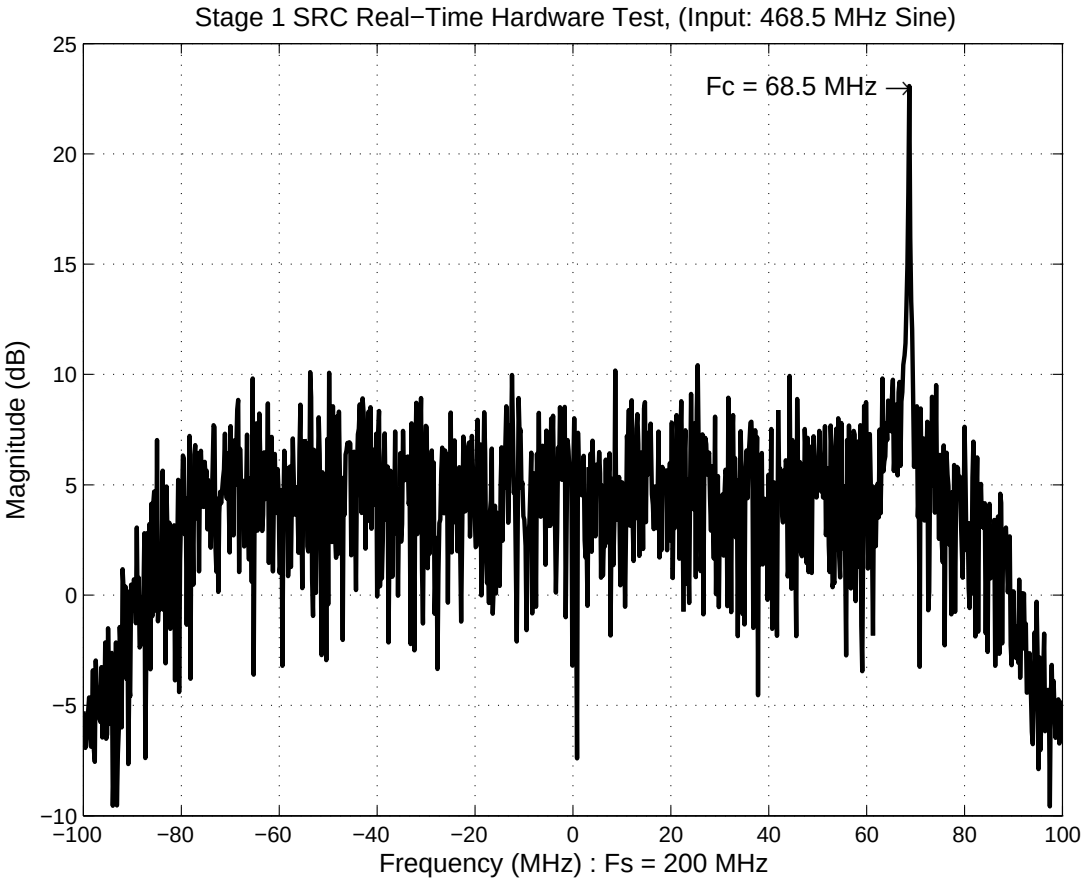

Fig. 5.13: FFT of the output of stage 1.

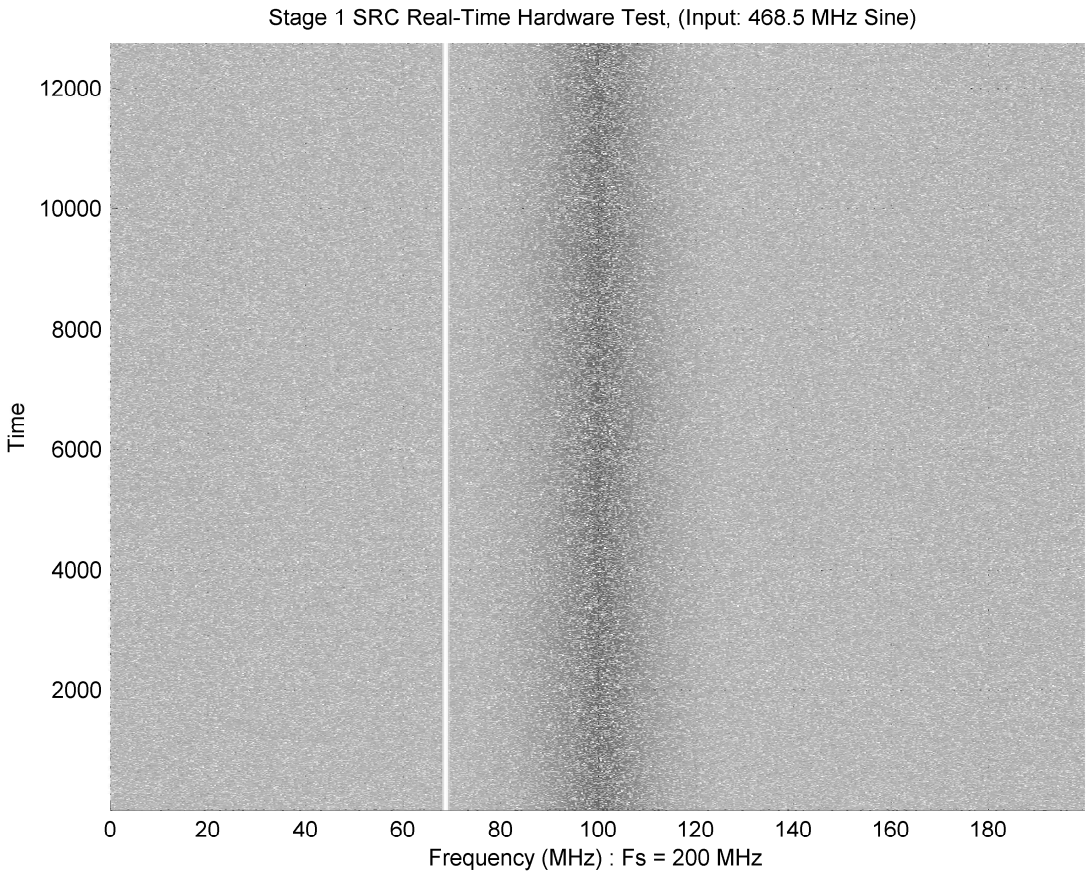

Fig. 5.14: Spectrogram of the output of stage 1.

the downsampler. In this stage we expect that the frequency of interest, 468 MHz with a bandwidth of 3 MHz, would be mixed down to baseband by 0.34 of the sampling frequency, or 68 MHz, to a 3 MHz band centered at 0 MHz, then filtered and downsampled by 5. Figure 5.17 shows the spectrum of the processed signal, and Figure 5.18 shows the Wavescope result of the simulation. Figure 5.19 shows the spectrum of the output of the DDS block. Each Figure shows that stage 2 is working as expected.

5.7 Stage 2 real-time test results

The following shows the real-time test results of stage 2 of the system. The test signal used to perform the real-time test is a 468.5 MHz sine wave, and after processing the signal should be mixed down to 500 KHz and have a sampling rate of 40 MHz. Figure 5.20 shows an FFT of the results of the stage 2 real-time test, and Figure 5.21 shows a spectrogram of the results. It can be seen from the Figures that the results of the real-time test match the desired results.

5.8 Stage 3 implementation and simulation

Stage 3 was designed and implemented to take the outputs of stage 3 and then upsample by 3, and downsample by 2. Figure 5.22 shows the stage 3 implementation of the sampling rate converter. In this stage we expect that the frequency of interest, 468 MHz with a bandwidth of 3 MHz, which was mixed to baseband in stage 2 and had a sampling frequency of 40 MHz, to now have sampling rate at 60 MHz and still have a bandwidth of 3 MHz. Figure 5.23 shows the spectrum of the processed signal, and Figure 5.24 shows the Wavescope result of the simulation. Each Figure shows that stage 3 is working as expected.

5.9 Stage 3 real-time test results

The following shows the real-time test results of stage 3 of the system. The test signal used to perform the real-time test is a 468.5 MHz sine wave, and after processing the signal should be mixed down to 500 KHz, and have a sampling rate of 60 MHz. Figure 5.25 shows an FFT of the results of the stage 3 real-time test, and Figure 5.26 shows a spectrogram of

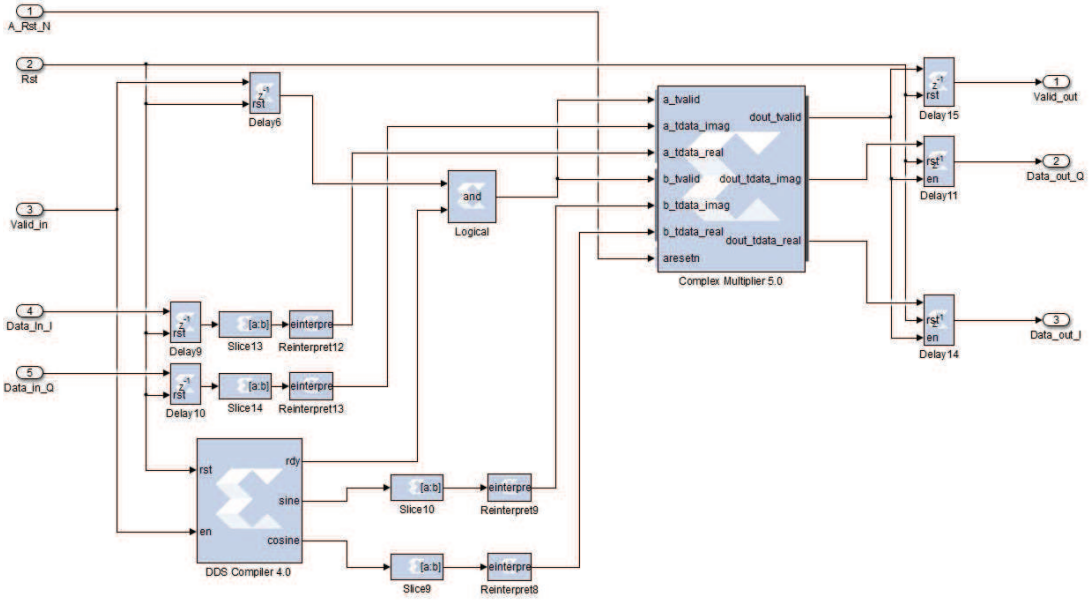

Fig. 5.15: Implementation of stage 2 DDS for mix down to baseband.

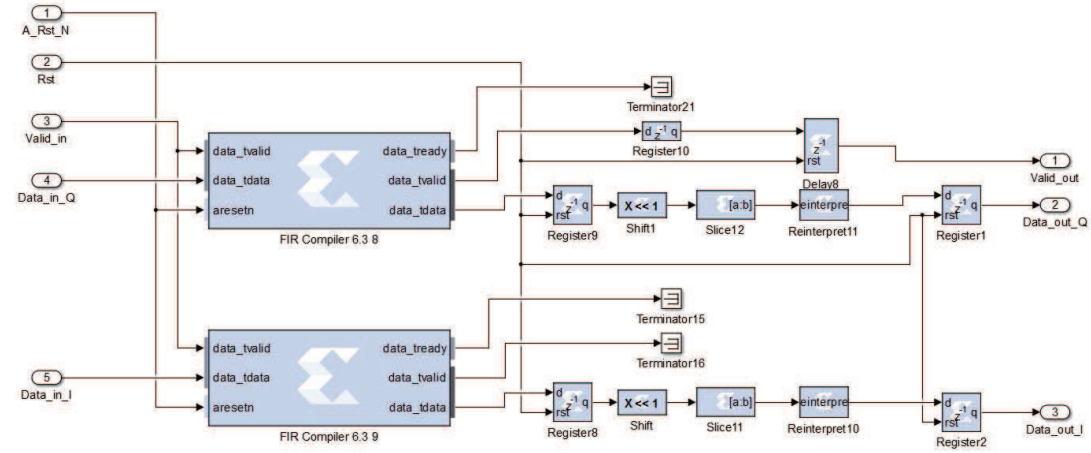

Fig. 5.16: Implementation of stage 2 with a polyphase down by 5 filter structure.

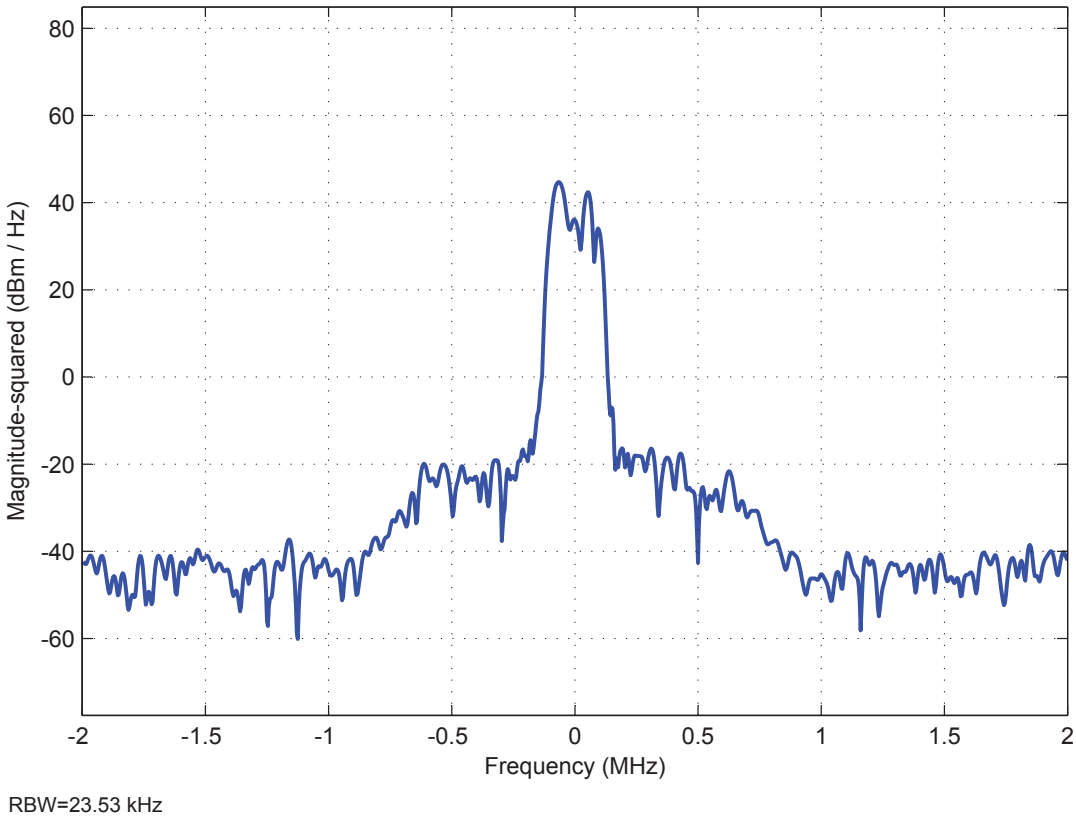

Fig. 5.17: Spectrum of the signal at the output of stage 2.

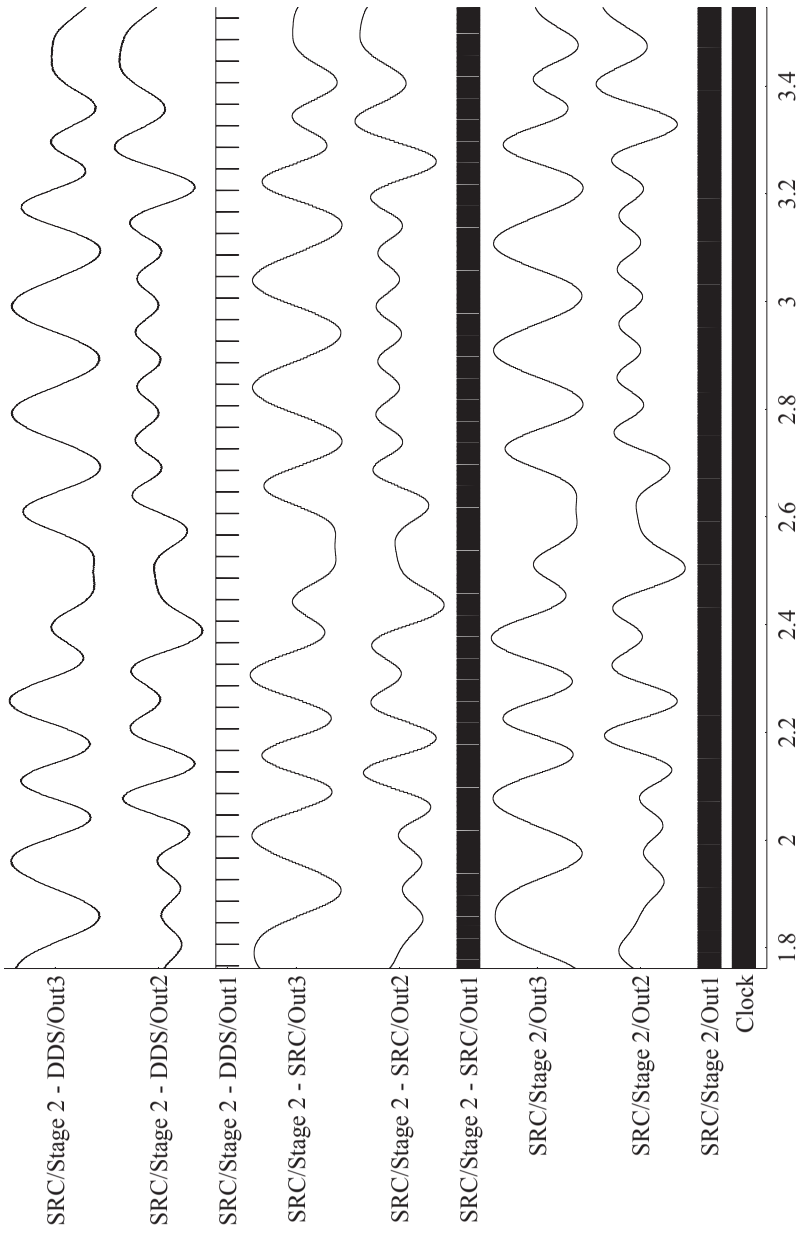

Fig. 5.18: Output of the Wavescope tool for stage 2.

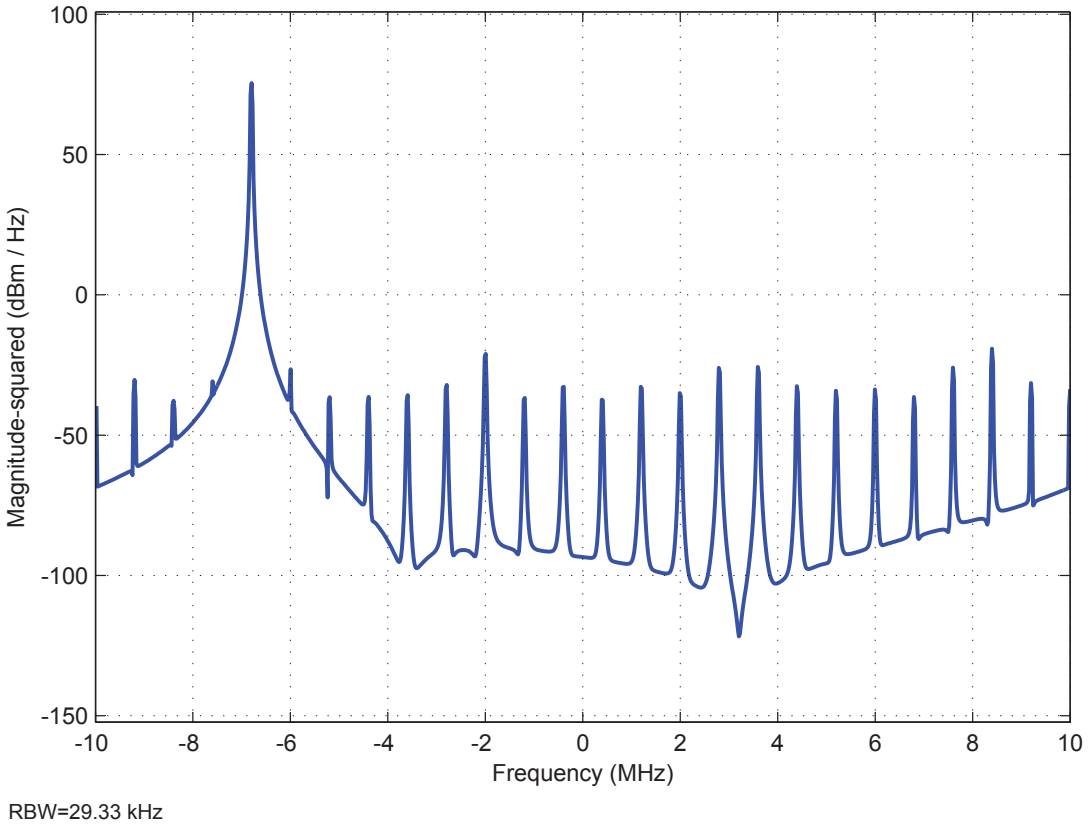

Fig. 5.19: Spectrum of the signal at the output of stage 2 DDS.

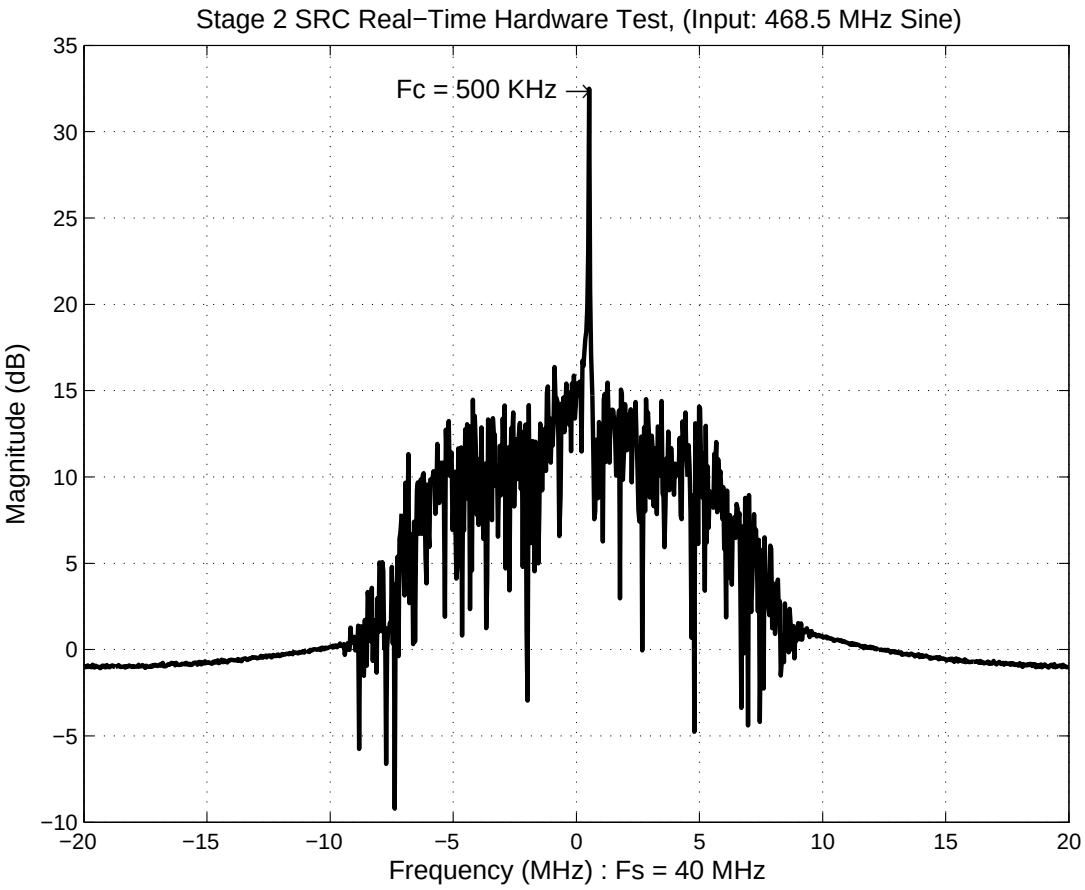

Fig. 5.20: FFT of the output of stage 2.

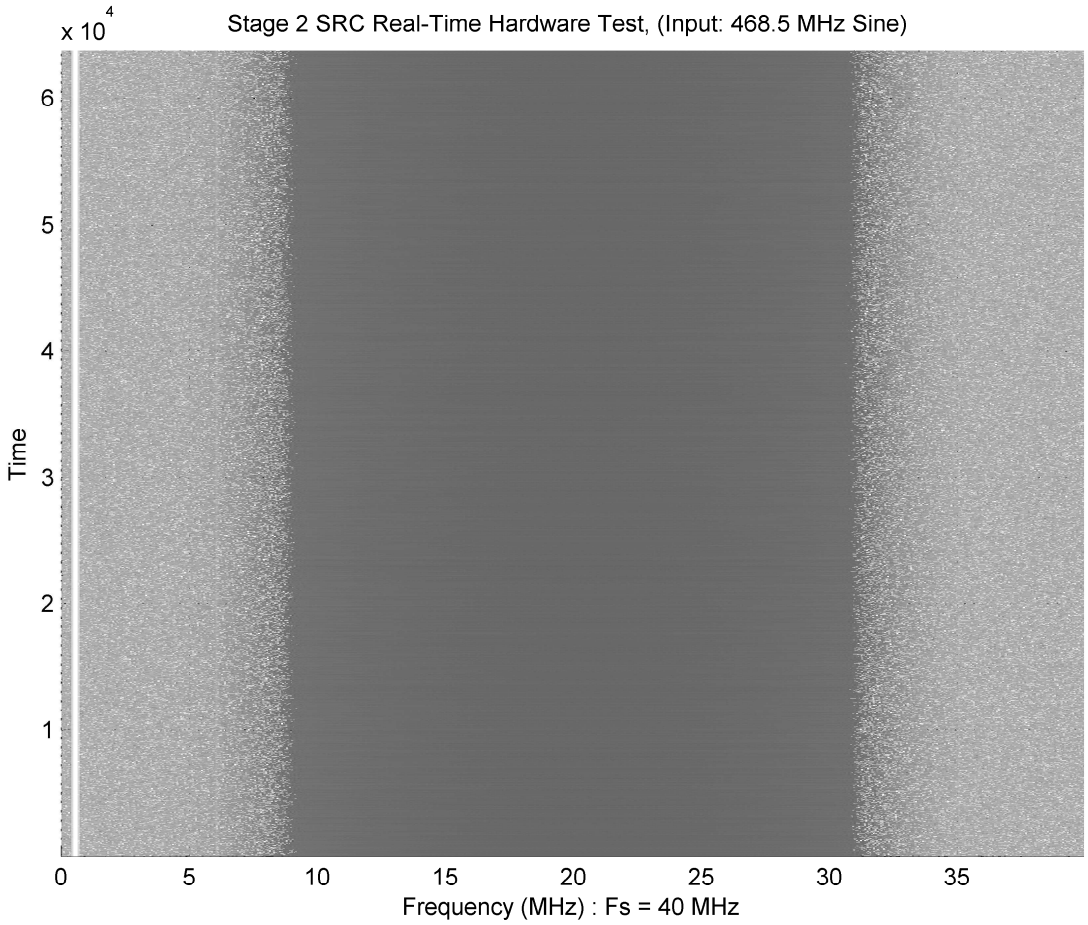

Fig. 5.21: Spectrogram of the output of stage 2.

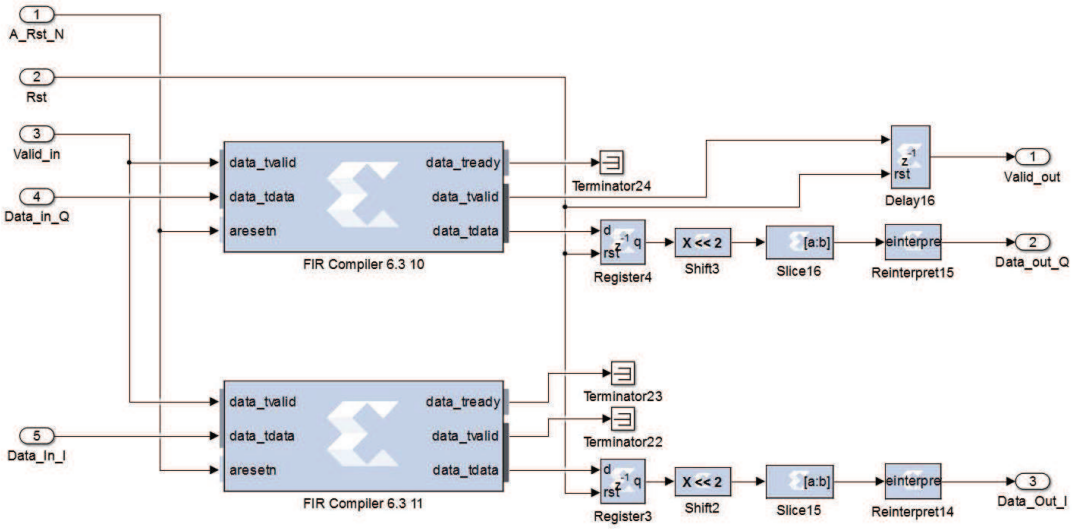

Fig. 5.22: Implementation of stage 3 with a polyphase up by 3, down by 2 filter structure.

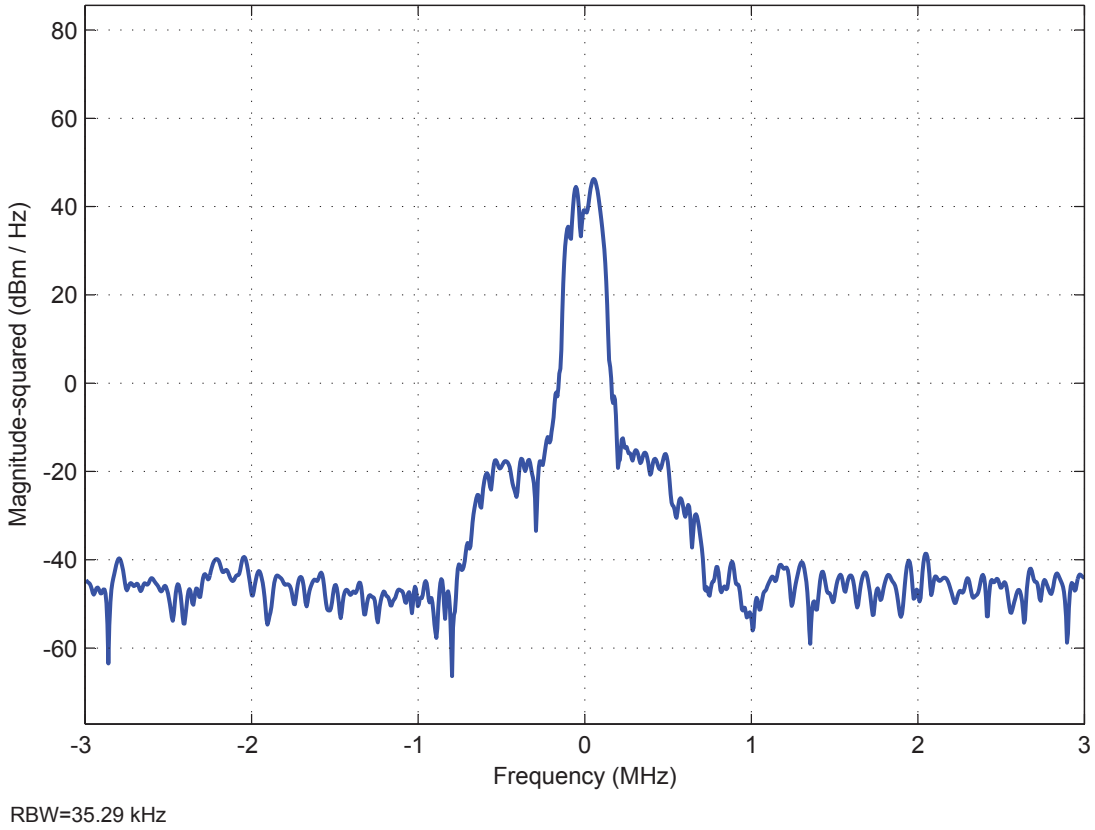

Fig. 5.23: Spectrum of the signal at the output of stage 3.

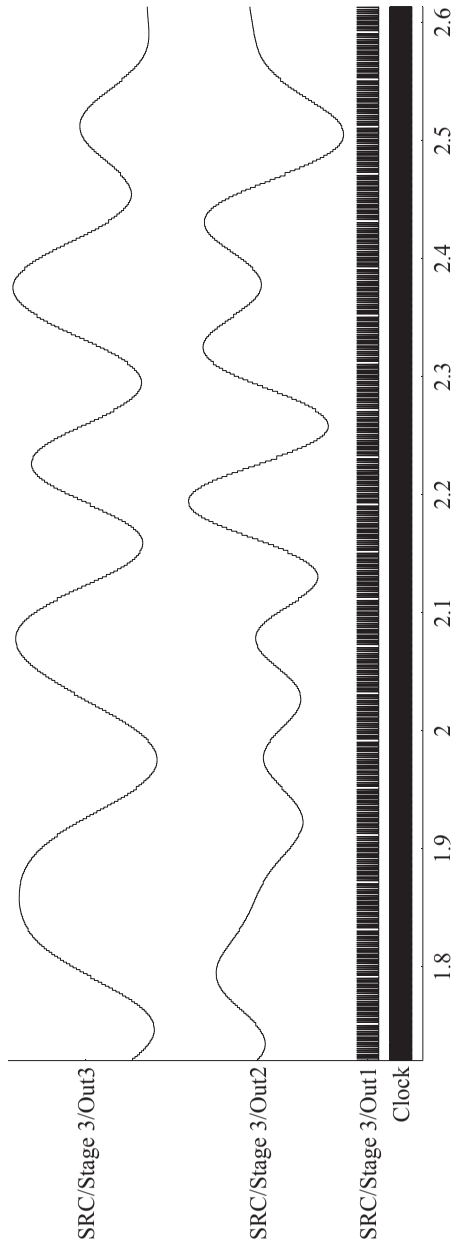

Fig. 5.24: Output of the Wavescope tool for stage 3.

the results. It can be seen from the Figures that the results of the real-time test match the desired results.

5.10 Stage 4 implementation and simulation

Stage 4 was designed and implemented to take the outputs of stage 3 and downsample by 5. Figure 5.27 shows the stage 4 implementation of the sampling rate converter. In this stage we expect that the frequency of interest, 468 MHz with a bandwidth of 3 MHz, which was mixed to baseband in stage 2 and now has a sampling frequency of 60 MHz, to now have sampling rate at 12 MHz and still have a bandwidth of 3 MHz. Figure 5.28 shows the spectrum of the processed signal, and Figure 5.29 shows the Wavescope result of the simulation. Each Figure shows that stage 4 is working as expected.

5.11 Stage 4 real-time test results

The following shows the real-time test results of stage 4 of the system. The test signal used to perform the real-time test is a 468.5 MHz sine wave, and after processing the signal should be mixed down to 500 KHz and have a sampling rate of 12 MHz. Figure 5.30 shows an FFT of the results of the stage 4 real-time test, and Figure 5.31 shows a spectrogram of the results. It can be seen from the Figures that the results of the real-time test match the desired results.

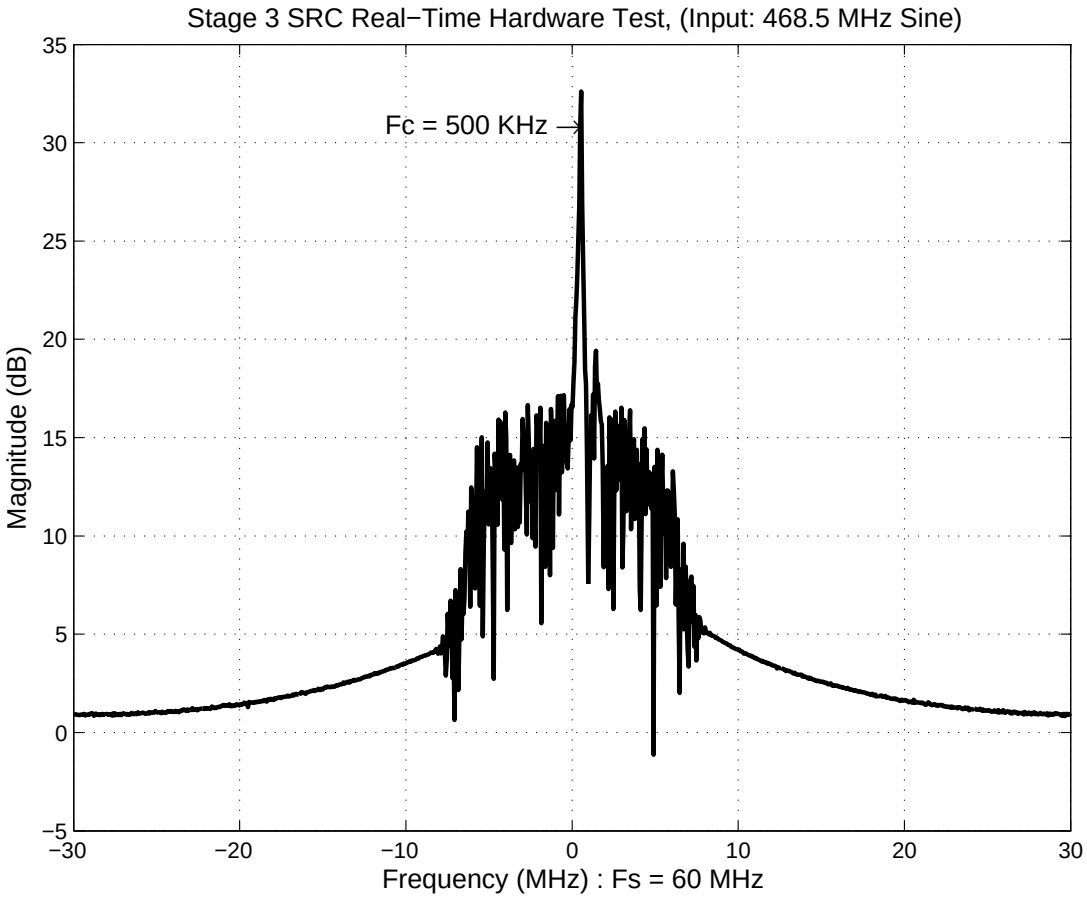

Fig. 5.25: FFT of the output of stage 3.

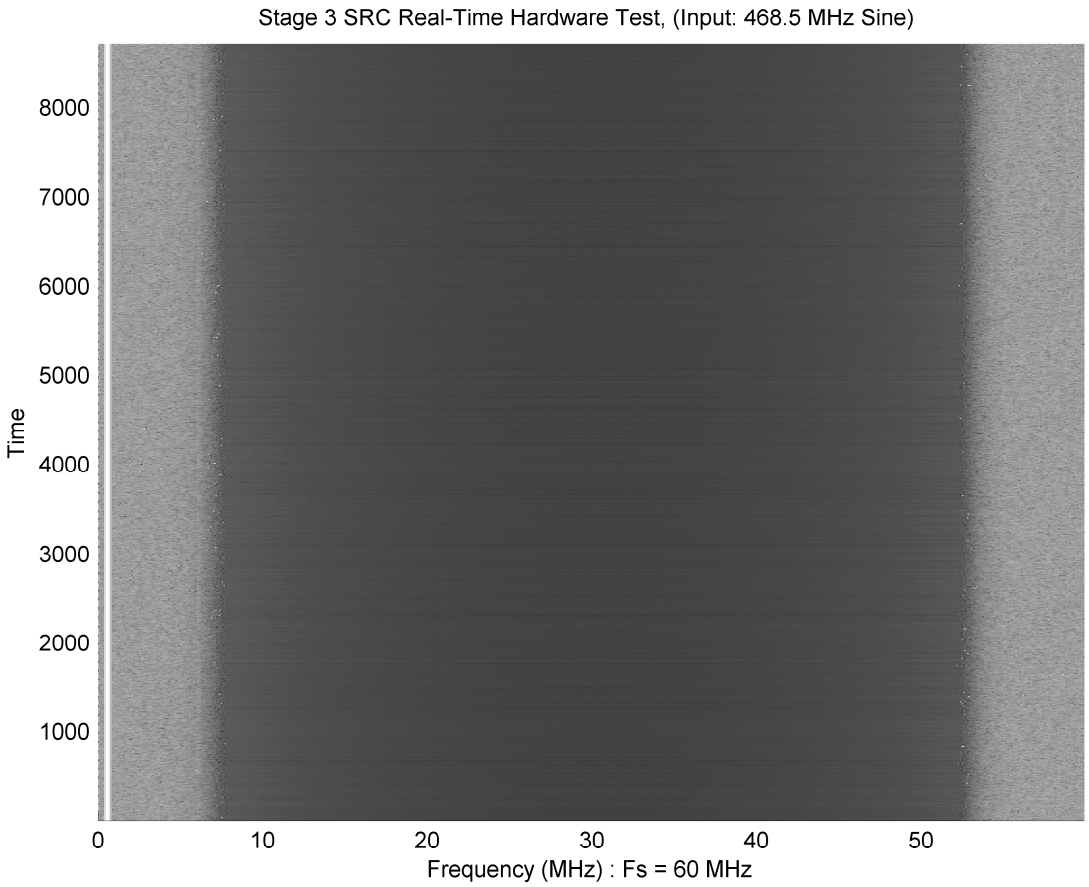

Fig. 5.26: Spectrogram of the output of stage 3.

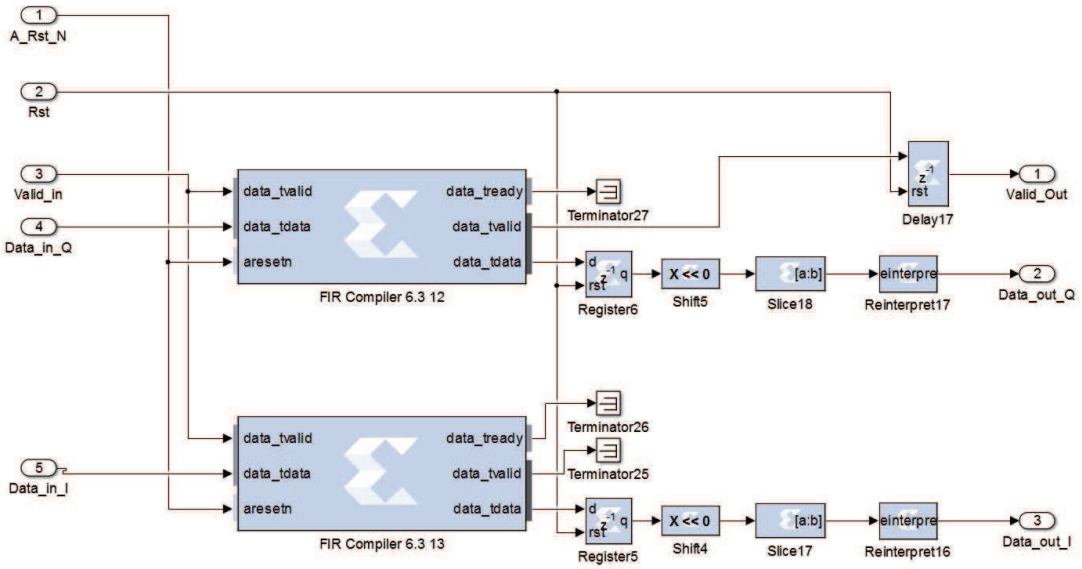

Fig. 5.27: Implementation of stage 4 with a polyphase down by 5 filter structure.

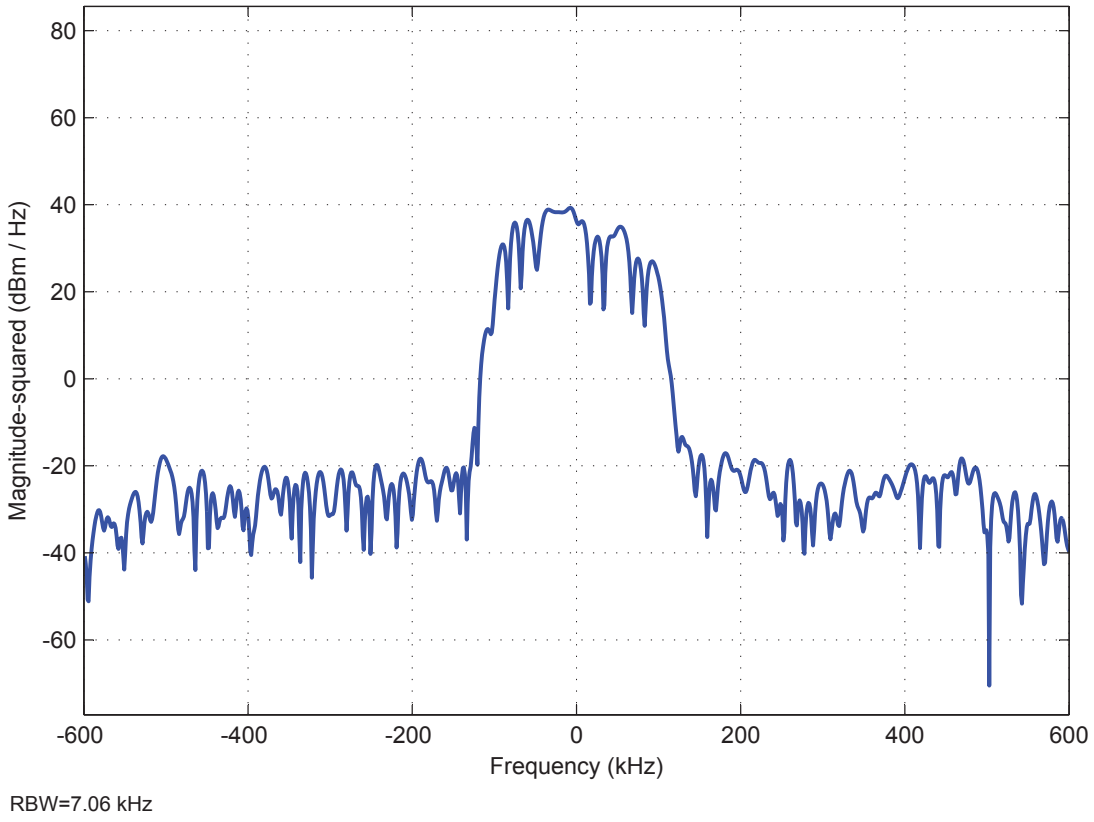

Fig. 5.28: Spectrum of the signal at the output of stage 4.

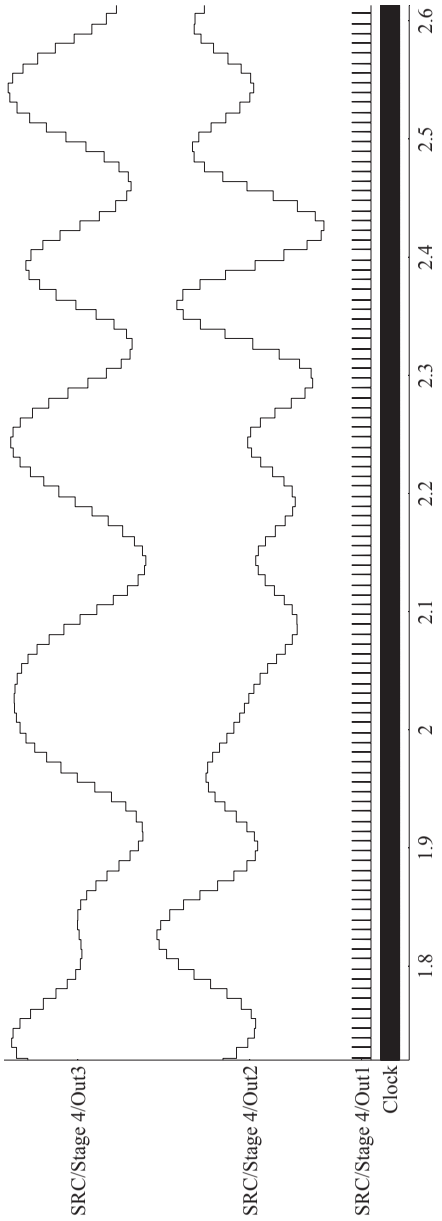

Fig. 5.29: Output of the Wavescope tool for stage 4.

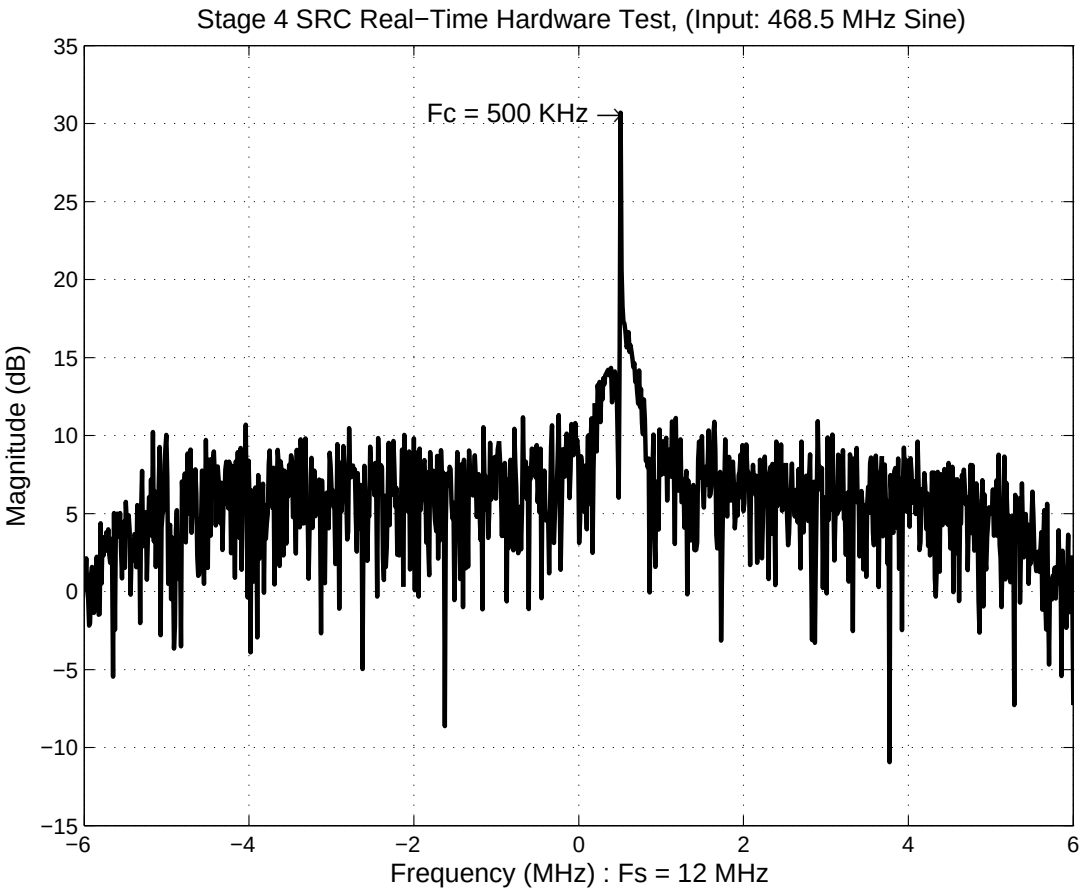

Fig. 5.30: FFT of the output of stage 4.

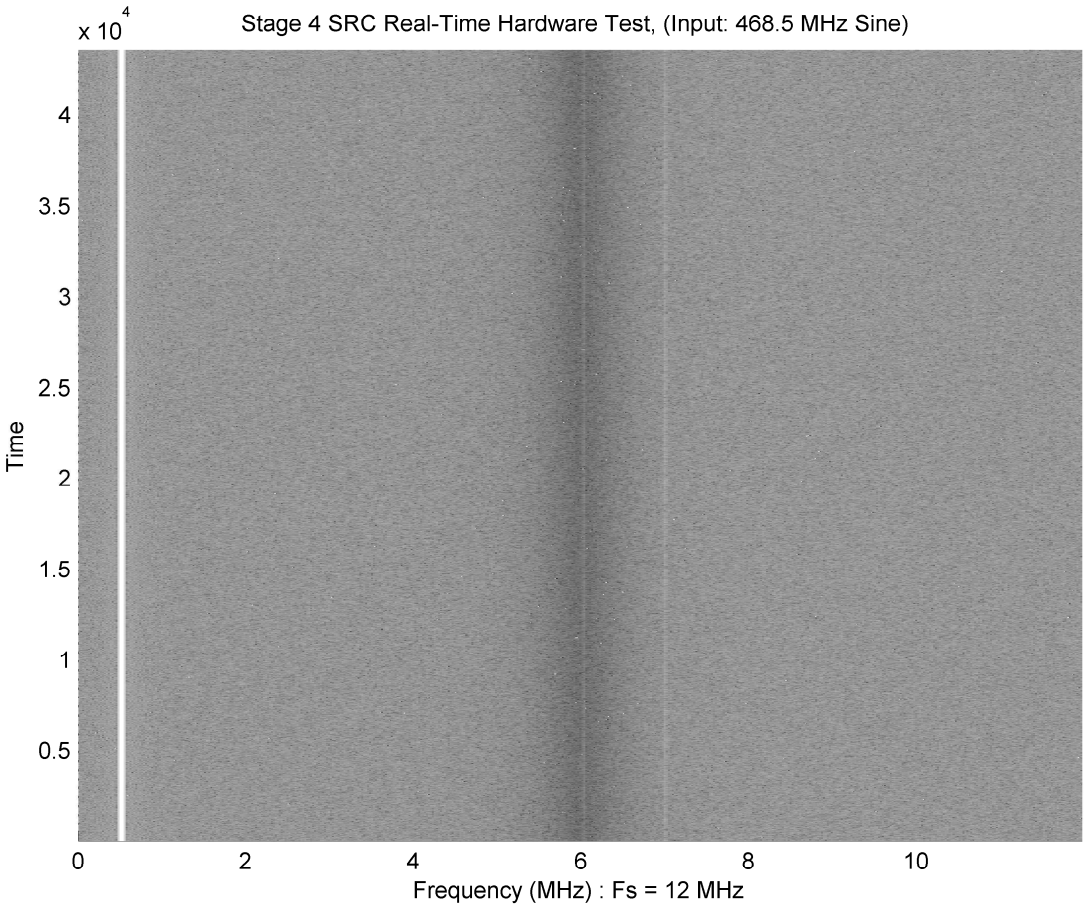

Fig. 5.31: Spectrogram of the output of stage 4.

Chapter 6

Results and conclusion

The purpose of this project was to design and develop a sampling rate conversion system for a real-time ground station receiver. This report has discussed the theory behind rational sampling rate conversion, the signal processing design and optimization, the hardware for the project, and finally the implementation of the design in hardware. This chapter will discuss the project and its implementation, as well as, the results of the real-time tests. The last section in this chapter will discuss future work on the real-time ground station receiver.

6.1 Project overview

The sampling rate conversion system for the real-time ground station receiver is the first step required to demodulate a received signal. While this project focuses mainly on the sampling rate conversion of the system, there is other work needing to be completed for the real-time ground station to work. The SRC system was design to take a signal, sampled at 1600 MHz, and downsample the signal to 12 MHz for the demodulator. The designed system provides at its output a signal, sampled at 12 MHz, with both an in-phase and quadrature-phase signal that can be directly connected to the OQPSK demodulator. The system was designed to run in real-time on an FPGA that is connected to a computer using a high-speed PCIe interface for data aquisition.

6.2 Real-time test results

The purpose of this project was to design and implement a sampling rate conversion system that could run in real-time. The impementation of this project was performed in stages, so that each stage could be simulated and tested in hardware to verify its functionality. Chapter 5 discussed the implementation, simulation, and real-time testing of the

sampling rate conversion system. It is shown that each stage of the design was thoroughly tested, as was the sampling rate conversion system as a whole. The sampling rate conversion system operated as required, and is ready to be attached to the demodulator.

6.3 Future work

As this project is part of a larger design, there is a lot of work that must be completed for this project to be considered complete. In Chapter 1, the introduction to this document, it is shown that the real-time ground station receiver must be capable of the following:

- Downsample signal to 12 MHz.
- Shift signal to baseband and preserve spectrum.
- Perform OQPSK demodulation.
- Implement automatic gain control.
- Doppler phase correction.
- Interference cancellation.
- Forward error correction.
- Convert symbols to bits.
- Store bits to hard disk.

The first and second items on this list have been completed with this project. However this leaves the remaining items as future work needing to be completed for the real-time ground station receiver.

References

[1] B. Klofas, "Cubesat communications system table," 2015. [Online]. Available: http://www.klofas.com/comm-table/

[2] C. Fish, C. Swenson, T. Neilson, B. Bingham, J. Gunther, E. Stromberg, S. Burr, R. Burt, and M. Whitely, "DICE mission design, development, and implementation: Success and challenges," *Confrence on Small Satellites*, 2012.

[3] S. K. Mitra, *Digital Signal Processing: A Computer Based Approach*. McGraw-Hill Companies, Inc, 2011.

[4] D. G. Manolakis and V. K. Ingle, *Applied Digital Signal Processing*. Cambridge University Press, 2011.

[5] M. Rice, *Digital Communications: A Discrete-Time Approach*. Pearson Education. Inc, 2009.

[6] *X6-GSPS*, Innovative Integration. [Online]. Available: http://www.innovative-dsp.com/cgi-bin/dlDocs.cgi?product=X6-GSPS

[7] *Matlab X6-GSPS BSP Manual*, 2nd ed., Innovative Integration, May 2012. [Online]. Available: http://www.innovative-dsp.com/cgi-bin/dlDocs.cgi?product=X6-GSPS

[8] *X6-GSPS FrameWork Logic User Guide*, Innovative Integration. [Online]. Available: http://www.innovative-dsp.com/cgi-bin/dlDocs.cgi?product=X6-GSPS

[9] "Innovative Integration." [Online]. Available: http://www.innovative-dsp.com/products.php?product=X6-GSPS

[10] *Virtex-6 Family overview*, Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

[11] *Xilinx System Generator*, 14th ed., Xilinx. [Online]. Available: http://www.innovative-dsp.com/cgi-bin/dlDocs.cgi?product=X6-GSPS

Appendices

Appendix A

System simulation using Matlab

A.1 Description of code

The code for the simulation of the designed project was written using Matlab. It is broken up into logical sections based on the stages of the design. Multiple plots are created to verify the signal at each stage of the simulation.

A.2 Simulation code

```
N = 20000; % number of samples to generate
scale = 1; % scale factor
Fs = 1600e6/scale; % sample rate
Ts = 1/Fs;
t = 0:Ts:N*Ts;
% generate test frequencies
freq = 468.5e6/scale;
freq2 = 588.5e6;
freq3 = 428.5e6;
noise = .2*randn(N+1,1)';
hist(noise)
sig = sin(2*pi*freq*t)+.75*sin(2*pi*freq2*t)+.75*sin(2*pi*freq3*t)+noise;
figure
subplot(211)
stairs(t,sig,'LineWidth',1.5);
title('(a) Test signal for simulation');
ylabel('Amplitude');
xlabel('Time (Sec): F_s = 1600 MHz');

```

```
% plot test signal spectrum
L = length(sig);
NFFT = 2^nextpow2(L);
Y = fft((sig),NFFT)/L;
Y1 = 10*log10(fftshift((Y)));
f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
subplot(212)
plot(f,Y1,'LineWidth',1.5);
title('(b) FFT of test signal, F_1 = 468.5, F_2 = 588.5, F_3 = 428.5, and gaussian noise');
ylabel('dB');
xlabel('Frequency (MHz): F_s = 1600 MHz');
axis([-Fs/2 Fs/2 -25 0])
grid on;


% mix down by 1/4 Fs
n = 0:1:N;
cplx = sqrt(2)*exp(-1j*pi/2*n);
sig1 = sig.*cplx;

% plot mixed signal
L = length(sig1);
NFFT = 2^nextpow2(L);
Y = fft((sig1),NFFT)/L;
Y1 = 10*log10(fftshift((Y)));
f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
figure;
subplot(211)
plot(f,Y1,'LineWidth',1.5);
title('(a) FFT of spectrum mixed by (1/4) F_s - Stage 1');
ylabel('dB');
xlabel('Frequency (MHz): F_s = 1600 MHz');
axis([-Fs/2 Fs/2 -25 0])
grid on;

```

```
% stage 1 low pass filter
lpf1 = LPF1;
LPF1n = lpf1.numerator;
sig2 = filter(LPF1n,1,sig1);

% plot filtered signal
L = length(sig2);
NFFT = 2^nextpow2(L);
Y = fft((sig2),NFFT)/L;
Y1 = 10*log10(fftshift((Y)));
f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);

subplot(212)
plot(f,Y1,'LineWidth',1.5);
title('(b) FFT of spectrum after H_{lp1}[n] - Stage 1');
ylabel('dB');
xlabel('Frequency (MHz): F_s = 1600 MHz');
axis([-Fs/2 Fs/2 -45 0])
grid on;



% downsample by 8
m  = 8;
Fs = Fs/m;
sig3 = downsample(sig2,m);
t1 = downsample(t,m);

% plot downsampled signal
L = length(sig3);
NFFT = 2^nextpow2(L);
Y = fft((sig3),NFFT)/L;
Y1 = 10*log10(fftshift((Y)));
f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
figure;
```

```
subplot(211)
plot(f,Y1,'LineWidth',1.5);
title('(a) FFT of spectrum after downsample by 8 – Stage 1');
ylabel('dB');
xlabel('Frequency (MHz): F_s = 200 MHz');
axis([-Fs/2 Fs/2 -35 0])
grid on;
subplot(212)
plot(t1,real(sig3),'LineWidth',1.5);
title('(b) Signal after downsample by 8 – Stage 1');
ylabel('Amplitude');
xlabel('Time (Sec): F_s = 200 MHz');


% mix down by 68 MHz
dFreq = .34;
n = 0:1:length(sig3)-1;
cplx = exp(-1j*2*pi*dFreq*n);

% plot baseband signal
L = length(sig4);
NFFT = 2^nextpow2(L);
Y = fft((sig4),NFFT)/L;
Y1 = 10*log10(fftshift((Y)));
f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
figure;
subplot(211)
plot(f,Y1);
title('(a) Mixed down by (.34) F_s');
ylabel('dB');
xlabel('Frequency (MHz): F_s = 200 MHz – Stage 2');
axis([-Fs/2 Fs/2 -35 0])
grid on;
subplot(212)
plot(t1,real(sig4),'LineWidth',1.5);
```

```
125 title('(b) Signal after mix down by (.34) F_s – Stage 2');
126 ylabel('Amplitude');
127 xlabel('Time (Sec): F_s = 200 MHz');
128
129
130 % stage 2 lowpass filter
131 s2 = S2LPF;
132 s2n = s2.numerator;
133 fvtool(s2n);
134 sig5 = filter(s2n,1,sig4);
135
136 % plot filtered signal
137 L = length(sig5);
138 NFFT = 2^nextpow2(L);
139 Y = fft((sig5),NFFT)/L;
140 Y1 = 10*log10(fftshift((Y)));
141 f = (Fs/2*linspace(–Fs/2,Fs/2,NFFT))/(Fs/2);
142 figure;
143 subplot(211)
144 plot(f,Y1,'LineWidth',1.5);
145 title('(a) FFT of spectrum after H_{lp2}[n] – Stage 2');
146 ylabel('dB');
147 xlabel('Frequency (MHz): F_s = 200 MHz');
148 axis([–Fs/2 Fs/2 –40 0])
149 grid on;
150 subplot(212)
151 plot(t1,real(sig5),'LineWidth',1.5);
152 title('(b) Signal after H_{lp2}[n] – Stage 2');
153 ylabel('Amplitude');
154 xlabel('Time (Sec): F_s = 200 MHz');
155
156
157 % downsample by 5
158 m  = 5;
159 Fs = Fs/m;
```

```
160 sig6 = downsample(sig5,m);
161 t1 = downsample(t1,m);
162
163 % plot downsampled signal
164 L = length(sig6);
165 NFFT = 2^nextpow2(L);
166 Y = fft((sig6),NFFT)/L;
167 Y1 = 10*log10(fftshift((Y)));
168 f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
169 figure;
170 subplot(211)
171 plot(f,Y1,'LineWidth',1.5);
172 title('(a) FFT of spectrum after downsample by 5 - Stage 2');
173 ylabel('dB');
174 xlabel('Frequency (MHz): F_s = 40 MHz');
175 axis([-Fs/2 Fs/2 -30 0])
176 grid on;
177 subplot(212)
178 plot(t1,real(sig6),'LineWidth',1.5);
179 title('(b) Signal after downsample by 5 - Stage 2');
180 ylabel('Amplitude');
181 xlabel('Time (Sec): F_s = 40 MHz');
182
183
184 % upsample by 3
185 m  = 3;
186 Fs = Fs*m;
187 sig7 = upsample(sig6,m);
188 t1 = 0:1/120e6:N*Ts;%upsample(t1,m);
189 sig7 = sig7(1:length(t1));
190
191 % plot upsampled signal
192 L = length(sig7);
193 NFFT = 2^nextpow2(L);
194 Y = fft((sig7),NFFT)/L;
```

```
195 Y1 = 10*log10(fftshift((Y)));
196 f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
197 figure;
198 subplot(211)
199 plot(f,Y1,'LineWidth',1.5);
200 title('(a) FFT of spectrum after upsample by 3, with images from upsampling - Stage 3');
201 ylabel('dB');
202 xlabel('Frequency (MHz): F_s = 120 MHz');
203 axis([-Fs/2 Fs/2 -40 0])
204 grid on;
205 subplot(212)
206 plot(t1,real(sig7),'LineWidth',1.5);
207 title('(b) Signal after upsample by 3 - Stage 3');
208 ylabel('Amplitude');
209 xlabel('Time (Sec): F_s = 120 MHz');
210
211
212 % stage 3 lowpass filter
213 s3 = S3LPF;
214 s3n = s3.numerator;
215 fvtool(s3n);
216 sig8 = filter(s3n,1,sig7);
217
218 % plot filtered signal
219 L = length(sig8);
220 NFFT = 2^nextpow2(L);
221 Y = fft((sig8),NFFT)/L;
222 Y1 = 10*log10(fftshift((Y)));
223 f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
224 figure;
225 subplot(211)
226 plot(f,Y1,'LineWidth',1.5);
227 title('(a) FFT of spectrum after H_{lp3}[n], with interpolation images removed - Stage 3');
228 ylabel('dB');
229 xlabel('Frequency (MHz): F_s = 120 MHz');
```

```
230 axis([-Fs/2 Fs/2 -40 0])
231 grid on;
232 subplot(212)
233 plot(t1,real(sig8),'LineWidth',1.5);
234 title('(b) Signal after H_{lp3}[n] - Stage 3');
235 ylabel('Amplitude');
236 xlabel('Time (Sec): F_s = 120 MHz');
237
238 % downsample by 2
239 m  = 2;
240 Fs = Fs/m;
241 sig9 = downsample(sig8,m);
242 t1 = downsample(t1,m);
243
244 % plot downsampled signal
245 L = length(sig9);
246 NFFT = 2^nextpow2(L);
247 Y = fft((sig9),NFFT)/L;
248 Y1 = 10*log10(fftshift((Y)));
249 f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
250 figure;
251 subplot(211)
252 plot(f,Y1,'LineWidth',1.5);
253 title('(a) FFT of spectrum after downsample by 2 - Stage 3');
254 ylabel('dB');
255 xlabel('Frequency (MHz): F_s = 60 MHz');
256 axis([-Fs/2 Fs/2 -40 0])
257 grid on;
258 subplot(212)
259 plot(t1,real(sig9),'LineWidth',1.5);
260 title('(b) Signal after downsample by 2 - Stage 3');
261 ylabel('Amplitude');
262 xlabel('Time (Sec): F_s = 60 MHz');
263
264
```

```
% stage 4 lowpass filter
s4 = S4LPF;
s4n = s4.numerator;
fvtool(s4n)
sig10 = filter(s4n,1,sig9);

% plot filtered signal
L = length(sig10);
NFFT = 2^nextpow2(L);
Y = fft((sig10),NFFT)/L;
Y1 = 10*log10(fftshift((Y)));
f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
figure;
subplot(211)
plot(f,Y1,'LineWidth',1.5);
title('(a) FFT of spectrum after H_{lp4}[n] - Stage 4');
ylabel('dB');
xlabel('Frequency (MHz): F_s = 120 MHz');
axis([-Fs/2 Fs/2 -40 0])
grid on;
subplot(212)
plot(t1,real(sig10),'LineWidth',1.5);
title('(b) Signal after H_{lp4}[n] - Stage 4');
ylabel('Amplitude');
xlabel('Time (Sec): F_s = 120 MHz');


% downsample by 5
m  = 5;
Fs = Fs/m;
sig11 = downsample(sig10,m);
t1 = downsample(t1,m);

% plot downsampled signal
L = length(sig11);
```

```
300 NFFT = 2^nextpow2(L);
301 Y = fft((sig11),NFFT)/L;
302 Y1 = 10*log10(fftshift((Y)));
303 f = (Fs/2*linspace(-Fs/2,Fs/2,NFFT))/(Fs/2);
304 figure;
305 subplot(211)
306 plot(f,Y1,'LineWidth',1.5);
307 title('(a) FFT of spectrum after downsample by 5 - Stage 4');
308 ylabel('dB');
309 xlabel('Frequency (MHz): F_s = 12 MHz');
310 axis([-Fs/2 Fs/2 -40 0])
311 grid on;
312 subplot(212)
313 plot(t1,real(sig11),'LineWidth',1.5);
314 title('(b) Signal after downsample by 5 - Stage 4');
315 ylabel('Amplitude');
316 xlabel('Time (Sec): F_s = 12 MHz');
```

Appendix B

Project files

All project files relating to the design and simulation of the real-time ground station receiver sampling rate conversion system are provided in an attached compact disk (CD). Not all files are provided due to file sizes being to large to fit on a CD, and some files are licensed products that require a license from Innovative Integration to use. The files provided can easily be introduced into projects.

B.1 Directory structure

- 00-Documentation

  This folder contains the documentation for this project.

  - 00-Proposal

    This folder contains the proposal for this project and the supporting Tex files.

  - 01-Report

    This folder contains the report for this project and the supporting Tex files.

  - 02-Presentation

    This folder contains the final presentation for this project.

  - 03-DemodulatorDesign

    This folder contains a system block diagram for the OQPSK demodulator.

  - 04-DownsamplerDesign

    This folder contains a system block diagram for the sampling rate conversion system.

- 01-DigitalDesign

This folder contains the simulation and implementation of the system as designed for the hardware being used.

  - 00-SystemSimulation

    This folder contains the firmware simulation.

  - 01-SystemImplementation

    This folder contains the firmware implementaion.

  - 02-BitFiles

    This folder contains the programming bit file for the FPGA.

- 02-SignalProcessing

  This folder contains a Matlab script with supporting files to simulate the full sampling rate conversion system.

- 03-InnovativeIntegrationInfo

  This folder contains information on Innovative Integrations X6-GSPS hardware, and development environment setup.

  - 00-DevelopmentEnvSetup

    This folder contains videos, and documents created by Josh Hunsaker for setting up the development environment required for the X6-GSPS.

  - 01-Documentation

    This folder contains information from Innovative Integrations for the X6-GSPS.

  - 02-RMA

    This folder contains RMA information for board repairs.

  - 03-RemovePacketHeaderScript

    This folder contains a Matlab script that will remove packet headers from files downloaded using the SNAP utility for processing and signal verification.